

MQSeries® for OS/390®



System Management Guide

Version 2 Release 1

MQSeries® for OS/390®



System Management Guide

Version 2 Release 1

Note!

Before using this information and the product it supports, be sure to read the general information under Appendix E, "Notices" on page 485.

First edition (January 1999)

This edition applies to MQSeries for OS/390 Version 2 Release 1 and to any subsequent releases and modifications until otherwise indicated in new editions.

This book is based on the *System Management Guide* for MQSeries for MVS/ESA® 1.2, SC33-0806-06.

Order publications through your IBM® representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

At the back of this publication is a page titled "Sending your comments to IBM". If you want to make comments, but the methods described are not available to you, please address them to:

IBM United Kingdom Laboratories,
Information Development,
Mail Point 095,
Hursley Park,
Winchester,
Hampshire,
England,
SO21 2JN

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

© **Copyright International Business Machines Corporation 1993,1999. All rights reserved.**

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

About this book	xv
Who this book is for	xv
How to use this book	xv
Conventions used in this book	xv
What you need to know to understand this book	xvi
MQSeries publications	xvi
MQSeries cross-platform publications	xvi
MQSeries platform-specific publications	xix
MQSeries Level 1 product publications	xxi
Softcopy books	xxi
MQSeries information available on the Internet	xxii
Related publications	xxiii
OS/390	xxiii
CICS Transaction Server for OS/390	xxiii
CICS for MVS/ESA Version 4	xxiii
IMS	xxiv
DFSMS/MVS	xxiv
Security Server	xxiv
Other products	xxiv
What is new for this version	xxv

Part 1. Introduction	1
Chapter 1. Introduction to MQSeries for OS/390	3
MQSeries and message queuing	3
Messages and queues	4
Objects and commands	7
An overview of storage management	15
Logs and recovery	17
Events	18
Managing MQSeries with commands and utilities	18
MQSeries and other products	22
Making MQSeries available	26
CSA storage requirement	28

Part 2. Customizing MQSeries after installation	31
Chapter 2. Preparing for customization	33
Installable features	33
Libraries that exist after installation	34
Chapter 3. Customizing the queue managers	37
Introducing the customization tasks	37
Task 1: Choose which language you want to use	39
Task 2: Choose the distributed queuing facility	40
Task 3: Define the MQSeries subsystem to OS/390	41
Task 4: Update the OS/390 link list	46
Task 5: APF authorize the MQSeries load libraries	47
Task 6: Update the OS/390 program properties table	48
Task 7: Create procedures for the MQSeries subsystem	49
Task 8: Create procedures for the channel initiator	50
Task 9: Implement your ESM security controls	51
Task 10: Customize the initialization input data sets	52
Task 11: Create the bootstrap and log data sets	61
Task 12: Define your page sets	62
Task 13: Tailor your system parameter module	67
Task 14: Tailor the channel initiator parameter module	82
Task 15: Set up Batch, TSO, and RRS adapters	87
Task 16: Set up the operations and control panels	88
Task 17: Include the MQSeries dump formatting member	90
Task 18: Suppress information messages	91
Chapter 4. Migrating from previous versions of MQSeries for MVS/ESA	93
Migrating from Version 1.2 to Version 2.1	93
Migrating from Version 1.1.4 to Version 2.1	96
Migrating from Version 1.1.3 to Version 2.1	96
Migrating from Version 1.1.2 or earlier to Version 2.1	97
Coexistence with earlier versions of MQSeries for MVS/ESA	98
Chapter 5. Testing your queue manager	99
Basic function testing	99
Testing for C, C++, COBOL, PL/I, and CICS	102
Testing for distributed queuing	102
Part 3. MQSeries and CICS	107
Chapter 6. The MQSeries-CICS adapter	109
CICS adapter overview	109
Adapter components	111
Other adapter features	112
CICS adapter performance considerations	115
CICS adapter conventions	117
Setting up the CICS adapter	118
Customizing the CICS adapter	122
Chapter 7. Operating the CICS adapter	123
Invoking the adapter's control functions	123

Preparing to use the CICS adapter	125
Accessing the CICS adapter control panels	126
Starting a connection	127
Stopping a connection	130
Modifying a connection	132
Displaying details of connections and CICS tasks	135
Starting an instance of the task initiator CKTI	136
Stopping an instance of CKTI	138
Displaying the current instances of CKTI	140
Displaying CICS task information	141
Purging tasks that are using the CICS adapter	143
Shutting down a connection between MQSeries and the CICS adapter	144
Chapter 8. The MQSeries-CICS bridge	147
Introduction to the CICS bridge	147
Customizing the CICS bridge	152
Starting the CICS bridge	154
Shutting down the CICS bridge	155
Part 4. MQSeries and IMS	157
Chapter 9. The MQSeries-IMS adapter	159
Introduction to the IMS adapter	159
Setting up the IMS adapter	161
Chapter 10. Operating the IMS adapter	169
Controlling IMS connections	169
Connecting from the IMS control region	170
Displaying in-doubt units of recovery	172
Controlling IMS dependent region connections	174
Disconnecting from IMS	176
Controlling the IMS trigger monitor	177
Chapter 11. The MQSeries-IMS bridge	179
Introduction to the IMS bridge	179
Customizing the IMS bridge	181
Controlling the IMS bridge	182
Security	185
Part 5. Operating and administering MQSeries	187
Chapter 12. Operating MQSeries for OS/390	189
Issuing commands	189
Starting and stopping MQSeries	191
Using the operations and control panels	197
Chapter 13. Writing programs to administer MQSeries	211
Before you begin	211
Understanding how it all works	212
Using the command server	213
Preparing queues for administration programs	214
Sending commands to the command server	216

Putting messages on the system-command input queue	217
Retrieving replies to your commands	218
Interpreting the replies	220
If you do not receive a reply	226
Passing commands using MGCR or MGCRE	226
Chapter 14. Using the MQSeries utilities	227
How to read syntax diagrams	228
MQSeries utility program (CSQUTIL)	229
Page set management functions	231
MQSeries command management functions	238
Queue management functions	245
The change log inventory utility (CSQJU003)	256
The print log map utility (CSQJU004)	265
The log print utility (CSQ1LOGP)	266

Part 6. Backup, recovery, and restart 269

Chapter 15. Introducing some recovery concepts	273
How changes are made to data	273
How consistency is maintained	275

Chapter 16. Understanding termination and restart	279
What happens during termination	279
Connections and threads	280
What happens during restart and recovery	281
How in-doubt units of recovery are resolved	283
Recovering CICS units of recovery manually	286
Recovering IMS units of recovery manually	288
Recovering RRS units of recovery manually	290
What happens when the CICS adapter restarts	291
What happens when the IMS adapter restarts	293
Using the OS/390 Automatic Restart Manager (ARM)	294

Chapter 17. Understanding the log and the bootstrap data set	299
What logs are	299
How the log is structured	301
How the logs are written	302
What the bootstrap data set is for	305
Managing the logs and BSDS	306

Chapter 18. Planning for backup and recovery	307
Introduction	307
General tips for backup and recovery	307
Planning your logging environment	309
Planning your archive storage	310
Other recovery considerations	311

Chapter 19. Managing the logs and the bootstrap data set	315
Archiving logs with the ARCHIVE LOG command	315
Discarding archive log data sets	317
Printing log records	319
Finding out what the BSDS contains	319

Changing the BSDS	321
Recovering logs	323
Chapter 20. Managing page sets	325
Adding a page set to a queue manager	325
When one of your page sets becomes full	325
How to balance loads on page sets	327
How to reduce a page set	330
Backing up and recovering page sets	331
Backing up and restoring queues using CSQUTIL	335
Chapter 21. Example recovery scenarios	337
Dealing with active log problems	338
Dealing with archive log problems	343
Dealing with BSDS errors	346
BSDS recovery	350
Dealing with page set problems	353
Restarting if you have lost your log data sets	355
Performing a cold start of MQSeries	356
Dealing with IMS-related problems	357
Dealing with hardware errors	359
<hr/>	
Part 7. Monitoring performance and resource usage	361
Chapter 22. Monitoring performance and resource usage	363
Getting snapshots of MQSeries	364
Using CICS adapter statistics	364
Investigating performance problems	364
Using System Management Facility	366
Using other products with MQSeries	367
Using MQSeries trace	368
Using MQSeries events	370
Chapter 23. Interpreting MQSeries performance statistics	371
SMF type 115 record layout	371
Message manager statistics	374
Data manager statistics	375
Buffer manager statistics	376
Log manager statistics	380
Sample SMF statistics records	382
Chapter 24. Interpreting MQSeries accounting data	385
SMF type 116 record layout	385
Message manager accounting	388
Sample SMF accounting record	390
<hr/>	
Part 8. Security	391
Chapter 25. Introduction to MQSeries security	393
Why you need to protect MQSeries resources	393
Implementing MQSeries security	394
Resources you can protect	395

Chapter 26. Using RACF classes and profiles	399
Using RACF security classes	400
RACF profiles	401
Switch profiles	401
Profiles used to protect MQSeries resources	404
Profiles for command security	421
Profiles for command resource security	422
Using the RESLEVEL security profile	425
User IDs for security checking	430
Auditing considerations	437
Chapter 27. MQSeries security implementation	439
Security implementation checklist	439
MQSeries security management	441
Customizing security	444
Security considerations for using MQSeries with CICS	445
Security considerations for using MQSeries with IMS	449
Security considerations for distributed queuing	453
Security considerations for cluster support	455
Security installation tasks	456
Example security scenario	458
Security problem determination	464

Part 9. Appendixes 467

Appendix A. Macros intended for customer use	469
General-use programming interface macros	469
General-use programming interface copy files	470
Product-sensitive programming interface macros	471
General-use programming interface include files	471
Appendix B. Using OTMA exits in IMS	473
Exit names	473
A sample scenario	473
Appendix C. Upgrading and applying service to TCP/IP, Language Environment, or OS/390 Callable Services	477
Appendix D. Enabling distributed queuing using CICS ISC	479
Defining MQSeries programs and data sets as CICS resources	479
Defining the channel definitions	480
Defining the CKMQ transient data queue	481
Defining MQSeries queues, triggers, and processes	481
Defining CICS resources used by distributed queuing	481
Defining access security	482
Setting up communications	482
Security considerations for distributed queuing (using CICS ISC)	482
Appendix E. Notices	485
Programming interface information	487
Trademarks	488

Part 10. Glossary and index	489
Glossary of terms and abbreviations	491
Index	501

Figures

1.	Representation of an MQSeries message	5
2.	MQSeries for OS/390 overview	9
3.	Mapping queues to page sets through storage classes	16
4.	Buffers, buffer pools, and page sets	18
5.	Sample IEFSSNss statements for defining subsystems	41
6.	PPT additional entries needed for MQSeries	48
7.	Example job for migrating queue objects	95
8.	RACF commands for CSQ4IVP1	100
9.	RACF commands for CSQ4IVPX	104
10.	Example output from CSQ4IVPX	105
11.	How CICS, the CICS adapter, and an MQSeries subsystem are related	111
12.	JCL fragment for upgrading the CICS CSD	119
13.	Sample PLT for use with the CICS adapter	120
14.	Sample INITPARM statement to set the default connection values for CICS	120
15.	Linking to the adapter connect program, CSQCQCON, from a PLT program	122
16.	Padding adapter commands	124
17.	Starting a queue manager that is identified by the CPF '+cpf'	125
18.	The CICS adapter control initial panel	126
19.	Starting a connection	127
20.	Starting a connection from the command line	128
21.	Starting a connection from the command line specifying parameters	128
22.	Specifying lowercase queue names	128
23.	Linking to the adapter connect program, CSQCQCON, from a CICS program	129
24.	Stopping a connection from the CKQC initial panel	130
25.	Stopping a connection from the command line—a quiesced shutdown	130
26.	Stopping a connection from the command line—a forced shutdown	131
27.	Stopping a connection from a CICS application program—a quiesced shutdown	131
28.	Stopping a connection from a CICS application program—a forced shutdown	131
29.	Modifying a connection	132
30.	Format of command to modify connection parameters from the command line	133
31.	Resetting connection statistics from the command line	133
32.	Changing the adapter's trace number and disabling the API-crossing exit from the command line	133
33.	Format of the MODIFY command issued from a CICS adapter application program	133
34.	Resetting connection statistics from a CICS program	134
35.	Linking to the adapter reset program, CSQCRST, from a CICS program	134
36.	The display connection panel	135
37.	Starting an instance of CKTI	136
38.	Starting an instance of CKTI—for the default initiation queue	137
39.	Starting an instance of CKTI—for a specified initiation queue	137
40.	Linking to the adapter task-initiator program CSQCSSQ from CICS	137
41.	Linking to the adapter task-initiator program CSQCSSQ from CICS	137
42.	Stopping an instance of the task initiator CKTI	138

43.	Stopping an instance of CKTI from the command line—for the default initiation queue	138
44.	Stopping an instance of CKTI from the command line—for a specified initiation queue	139
45.	Stopping an instance of CKTI from a program—for the default initiation queue from CICS	139
46.	Stopping an instance of CKTI from a program—for a specified initiation queue from CICS	139
47.	The CKQC Display CKTI panel	140
48.	The CKQC Display Task panel	141
49.	Message showing the status of a connection	142
50.	Displaying the status of a connection	142
51.	Linking to the adapter program CSQCDSPL from a CICS program	142
52.	Components and data flow to run a CICS DPL program	148
53.	Components and data flow to run a CICS 3270 transaction	150
54.	Sample JCL to link-edit the dynamic call stub	162
55.	CSQQDEFX macro syntax	165
56.	Layout of a subsystem definition table	166
57.	Example CSQQTAPL transaction definition for CSQQTRMN	167
58.	Example CSQQTPSB PSB definition for CSQQTRMN	167
59.	The MQSeries-IMS bridge	180
60.	Issuing a DISPLAY command from the OS/390 console	190
61.	Starting the MQSeries subsystem from an OS/390 console	192
62.	MQSeries startup messages for subsystem CSQ1	193
63.	Stopping MQSeries	195
64.	The MQSeries operations and control initial panel	202
65.	Listing queues	203
66.	Defining a local queue - first panel	204
67.	Defining a local queue - second panel	205
68.	Defining a local queue - trigger conditions	205
69.	Defining a local queue - event control	206
70.	Defining a local queue - backout reporting	206
71.	Specifying the queue manager that a utility is to work with	229
72.	Sample JCL for the FORMAT function of CSQUTIL	232
73.	Sample JCL showing the use of the COPYPAGE function	234
74.	Sample JCL showing the use of the RESETPAGE function	236
75.	Sample JCL for issuing MQSeries commands using CSQUTIL	240
76.	Sample JCL for using the MAKEDEF option of the COMMAND function	240
77.	Sample JCL for using the MAKECLNT option of the COMMAND function	241
78.	Sample JCL for the SDEFS function of CSQUTIL	243
79.	Sample JCL for the CSQUTIL COPY functions	247
80.	Sample JCL for the CSQUTIL SCOPY functions	250
81.	Sample JCL for the CSQUTIL EMPTY function	252
82.	Sample JCL for the CSQUTIL LOAD function	254
83.	Sample JCL to invoke the CSQJU003 utility	256
84.	Sample JCL to invoke the CSQJU004 utility	265
85.	Sample JCL to invoke the CSQ1LOGP utility using a BSDS	266
86.	Sample JCL to invoke the CSQ1LOGP utility using active log data sets	267
87.	Sample JCL to invoke the CSQ1LOGP utility using archive logs	267
88.	A unit of recovery within an application program	273
89.	A unit of recovery showing back out	274
90.	The two-phase commit process	276
91.	Example restart messages	292
92.	Sample ARM policy	295

Figures

93.	The logging process	302
94.	The off-loading process	303
95.	Sample input statements for CSQJU003	314
96.	Extract from a load balancing job	329
97.	Part of an SMF record showing the header and self-defining sections	372
98.	SMF record 115, subtype 1	383
99.	SMF record 115, subtype 2	384
100.	Part of an SMF record showing the header and self-defining sections	386
101.	SMF record 116	390
102.	Sample output from RACFRW showing RESLEVEL general audit records	438
103.	Typical output from the MQSeries command DISPLAY SECURITY	444
104.	Example security scenario	458
105.	OTMA pre-routing exit assembler sample	474
106.	Sample assembler DRU exit	476
107.	Adding the distributed queuing definitions to the CICS CSD	480
108.	Adding a DD statement to the CICS startup procedure	480

Tables

1.	Summary of MQSeries administrator commands	19
2.	System control commands	20
3.	MQSeries libraries that exist after installation	34
4.	National language feature libraries	39
5.	Subsystem name to CPF associations	42
6.	Example of CPF subset and superset rules	43
7.	Valid character set for CPF strings	43
8.	Default values of CSQ6SYSP parameters	68
9.	Default values of CSQ6LOGP parameters	74
10.	Default values of CSQ6ARVP parameters	76
11.	Default values of CSQ6CHIP parameters	83
12.	TCP/IP settings	86
13.	Shutting down a CICS adapter connection	144
14.	SSM specifications options	165
15.	Valid operations and control panels actions for MQSeries objects	199
16.	A summary of MQSeries utilities	227
17.	How to read syntax diagrams	228
18.	Termination using QUIESCE, FORCE, and RESTART	279
19.	SMF record header description	371
20.	Offsets to self-defining sections	373
21.	Structure of the message manager statistics record QMST	374
22.	Structure of the data manager statistics record QIST	375
23.	Structure of the buffer manager statistics record QPST	376
24.	Structure of the log manager statistics record QJST	380
25.	Problem symptoms that can be examined using log manager statistics	381
26.	SMF record header description	385
27.	Offsets to self-defining sections	387
28.	Structure of the message manager accounting record QWHS	388
29.	Structure of the message manager accounting record QMAC	388
30.	Structure of the thread cross reference record for a CICS system	389
31.	Structure of the thread cross reference record for an IMS system	390
32.	RACF classes used by MQSeries	400
33.	Switch profiles	402
34.	Access levels for queue security	406
35.	Access levels for close options on permanent dynamic queues	410
36.	RACF authority to the dead-letter queue and its alias	412
37.	Access levels for process security	414
38.	Access levels for namelist security	414
39.	Access levels for alternate user security	415
40.	Access levels for context security	417
41.	MQOPEN, MQPUT1, and MQCLOSE options and the security authorization required	419
42.	Commands, profiles, and their access levels	423
43.	Checks made at different RACF access levels for the Batch/TSO adapter	428
44.	Checks made at different RACF access levels for the CICS adapter	428
45.	Checks made at different RACF access levels for the IMS adapter	429
46.	Checks made at different RACF access levels for channel initiator connections	429
47.	User ID checking for Batch/TSO-type user IDs	431
48.	User ID checking for CICS-type user IDs	432

Tables

49.	User ID checking for IMS-type user IDs	432
50.	How the second user ID is determined for the IMS adapter	432
51.	User IDs checked for TCP/IP channels	433
52.	User IDs checked for LU 6.2 channels	434
53.	User IDs checked for LU 6.2 and TCP/IP server-connection channels	435
54.	CICS bridge monitor security	447
55.	CICS bridge task security	448
56.	RACF access to data sets associated with a queue manager	457
57.	RACF access to data sets associated with distributed queuing	457
58.	Security profiles for the example scenario	460
59.	Sample security profiles for the batch application on queue manager QM1	461
60.	Sample security profiles for queue manager QM2 using TCP	462
61.	Sample security profiles for queue manager QM2 using LU 6.2	462
62.	Sample security profiles for the CICS application on queue manager QM1	463
63.	Service has been applied or the product has been upgraded to a new release	477
64.	One of the products has been updated to a new release in a new SMP/E environment and libraries	477

About this book

MQSeries for OS/390 (referred to in this book as MQSeries) is part of the IBM MQSeries family of products. It provides application programming services that allow a new style of programming. This style enables you to code indirect program-to-program communication using *message queues*. MQSeries is an OS/390 *message-queue manager*.

MQSeries is an independent OS/390 subsystem using OS/390 cross-memory services, the CICS® task-related user exit product interface, and the IMS™ External Subsystem Attach Facility (ESAF). MQSeries supports application programs that can be run in:

- CICS
- MVS batch
- MVS/TSO
- IMS

Who this book is for

This book is intended for system programmers, system administrators, and system operators. It explains how to customize, operate, administer, and monitor MQSeries.

How to use this book

This book takes you through the steps required to make MQSeries available to application programmers. Read those sections that relate to your task, whether it be installing adapters, tuning, or communication.

The latter part of the book deals with termination, recovery, and restart. Read these sections when you need to perform such tasks.

Conventions used in this book

- Throughout this book, the term *object* refers to any MQSeries queue manager, queue, namelist, channel, storage class, or process.
- In the examples in this book, the string `+cpf` has been used to show the command prefix (CPF), and the commands are shown in UPPERCASE.
- CICS means both CICS Transaction Server for OS/390 and CICS for MVS/ESA unless otherwise stated. IMS means IMS/ESA® unless otherwise stated.
- Throughout this book, the default value `th1qua1` is used to indicate the target library high-level qualifier for MQSeries data sets in your installation.
- Throughout this book, the term *distributed queuing* refers to the distributed queuing feature (also known as the “non-CICS mover”). The term *distributed queuing using CICS ISC* is used to refer to the optional CICS distributed queuing feature (also known as the “CICS mover”).

What you need to know to understand this book

This book assumes you are familiar with the basic concepts of:

- CICS
- IMS
- OS/390 job control language (JCL)
- OS/390 Time Sharing Option (TSO)

If you want to write programs to administer MQSeries, this book assumes that you can write programs in one of the supported languages:

- COBOL
- C
- C++
- Assembler
- PL/I

You do not need to have written message-queuing programs previously.

MQSeries publications

This section describes the documentation available for all current MQSeries products.

MQSeries cross-platform publications

Most of these publications, which are sometimes referred to as the MQSeries “family” books, apply to all MQSeries Level 2 products. The latest MQSeries Level 2 products are:

- MQSeries for AIX V5.1
- MQSeries for AS/400 V4R2M1
- MQSeries for AT&T GIS UNIX V2.2
- MQSeries for Digital OpenVMS V2.2
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for OS/390 V2.1
- MQSeries for SINIX and DC/OSx V2.2
- MQSeries for Sun Solaris V5.1
- MQSeries for Tandem NonStop Kernel V2.2
- MQSeries for VSE/ESA V2.1
- MQSeries for Windows V2.0
- MQSeries for Windows V2.1
- MQSeries for Windows NT V5.1

Any exceptions to this general rule are indicated. (Publications that support the MQSeries Level 1 products are listed in “MQSeries Level 1 product publications” on page xxi. For a functional comparison of the Level 1 and Level 2 MQSeries products, see the *MQSeries Planning Guide*.)

MQSeries Brochure

The *MQSeries Brochure*, G511-1908, gives a brief introduction to the benefits of MQSeries. It is intended to support the purchasing decision, and describes some authentic customer use of MQSeries.

MQSeries: An Introduction to Messaging and Queuing

MQSeries: An Introduction to Messaging and Queuing, GC33-0805, describes briefly what MQSeries is, how it works, and how it can solve some classic interoperability problems. This book is intended for a more technical audience than the *MQSeries Brochure*.

MQSeries Planning Guide

The *MQSeries Planning Guide*, GC33-1349, describes some key MQSeries concepts, identifies items that need to be considered before MQSeries is installed, including storage requirements, backup and recovery, security, and migration from earlier releases, and specifies hardware and software requirements for every MQSeries platform.

MQSeries Intercommunication

The *MQSeries Intercommunication* book, SC33-1872, defines the concepts of distributed queuing and explains how to set up a distributed queuing network in a variety of MQSeries environments. In particular, it demonstrates how to (1) configure communications to and from a representative sample of MQSeries products, (2) create required MQSeries objects, and (3) create and configure MQSeries channels. The use of channel exits is also described.

MQSeries Clients

The *MQSeries Clients* book, GC33-1632, describes how to install, configure, use, and manage MQSeries client systems.

MQSeries System Administration

The *MQSeries System Administration* book, SC33-1873, supports day-to-day management of local and remote MQSeries objects. It includes topics such as security, recovery and restart, transactional support, problem determination, and the dead-letter queue handler. It also includes the syntax of the MQSeries control commands.

This book applies to the following MQSeries products only:

- MQSeries for AIX V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1

MQSeries Command Reference

The *MQSeries Command Reference*, SC33-1369, contains the syntax of the MQSC commands, which are used by MQSeries system operators and administrators to manage MQSeries objects.

MQSeries Programmable System Management

The *MQSeries Programmable System Management* book, SC33-1482, provides both reference and guidance information for users of MQSeries events, Programmable Command Format (PCF) messages, and installable services.

MQSeries Messages

The *MQSeries Messages* book, GC33-1876, which describes “AMQ” messages issued by MQSeries, applies to these MQSeries products only:

- MQSeries for AIX V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1

MQSeries publications

- MQSeries for Windows V2.0
- MQSeries for Windows V2.1

This book is available in softcopy only.

MQSeries Application Programming Guide

The *MQSeries Application Programming Guide*, SC33-0807, provides guidance information for users of the message queue interface (MQI). It describes how to design, write, and build an MQSeries application. It also includes full descriptions of the sample programs supplied with MQSeries.

MQSeries Application Programming Reference

The *MQSeries Application Programming Reference*, SC33-1673, provides comprehensive reference information for users of the MQI. It includes: data-type descriptions; MQI call syntax; attributes of MQSeries objects; return codes; constants; and code-page conversion tables.

MQSeries Application Programming Reference Summary

The *MQSeries Application Programming Reference Summary*, SX33-6095, summarizes the information in the *MQSeries Application Programming Reference* manual.

MQSeries Using C++

MQSeries Using C++, SC33-1877, provides both guidance and reference information for users of the MQSeries C++ programming-language binding to the MQI. MQSeries C++ is supported by these MQSeries products:

- MQSeries for AIX V5.1
- MQSeries for AS/400 V4R2M1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for OS/390 V2.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1

MQSeries C++ is also supported by MQSeries clients supplied with these products and installed in the following environments:

- AIX®
- HP-UX
- OS/2®
- Sun Solaris
- Windows NT®
- Windows® 3.1
- Windows 95 and Windows 98

MQSeries Using Java®

MQSeries Using Java, SC34-5456, provides both guidance and reference information for users of the MQSeries Bindings for Java and the MQSeries Client for Java. MQSeries Java is supported by these MQSeries products:

- MQSeries for AIX V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1

MQSeries Administration Interface Programming Guide and Reference

The *MQSeries Administration Interface Programming Guide and Reference*, SC34-5390, provides information for users of the MQAI. The MQAI is a

programming interface that simplifies the way in which applications manipulate Programmable Command Format (PCF) messages and their associated data structures.

This book applies to the following MQSeries products only:

- MQSeries for AIX V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1

MQSeries Queue Manager Clusters

MQSeries Queue Manager Clusters, SC34-5349, describes MQSeries clustering. It explains the concepts and terminology and shows how you can benefit by taking advantage of clustering. It details changes to the MQI, and summarizes the syntax of new and changed MQSeries commands. It shows a number of examples of tasks you can perform to set up and maintain clusters of queue managers.

This book applies to the following MQSeries products only:

- MQSeries for AIX V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for OS/390 V2.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1

MQSeries platform-specific publications

Each MQSeries product is documented in at least one platform-specific publication, in addition to the MQSeries family books.

MQSeries for AIX

MQSeries for AIX Version 5 Release 1 Quick Beginnings, GC33-1867

MQSeries for AS/400®

MQSeries for AS/400 Version 4 Release 2.1 Administration Guide, GC33-1956

MQSeries for AS/400 Version 4 Release 2 Application Programming Reference (RPG), SC33-1957

MQSeries for AT&T GIS UNIX®

MQSeries for AT&T GIS UNIX Version 2 Release 2 System Management Guide, SC33-1642

MQSeries for Digital OpenVMS

MQSeries for Digital OpenVMS Version 2 Release 2 System Management Guide, GC33-1791

MQSeries for Digital UNIX

MQSeries for Digital UNIX Version 2 Release 2.1 System Management Guide, GC34-5483

MQSeries for HP-UX

MQSeries for HP-UX Version 5 Release 1 Quick Beginnings, GC33-1869

MQSeries publications

MQSeries for OS/2 Warp

MQSeries for OS/2 Warp Version 5 Release 1 Quick Beginnings, GC33-1868

MQSeries for OS/390

MQSeries for OS/390 Version 2 Release 1 Licensed Program Specifications, GC34-5377

MQSeries for OS/390 Version 2 Release 1 Program Directory

MQSeries for OS/390 Version 2 Release 1 System Management Guide, SC34-5374

MQSeries for OS/390 Version 2 Release 1 Messages and Codes, GC34-5375

MQSeries for OS/390 Version 2 Release 1 Problem Determination Guide, GC34-5376

MQSeries link for R/3

MQSeries link for R/3 Version 1 Release 2 User's Guide, GC33-1934

MQSeries for SINIX and DC/OSx

MQSeries for SINIX and DC/OSx Version 2 Release 2 System Management Guide, GC33-1768

MQSeries for Sun Solaris

MQSeries for Sun Solaris Version 5 Release 1 Quick Beginnings, GC33-1870

MQSeries for Tandem NonStop Kernel

MQSeries for Tandem NonStop Kernel Version 2 Release 2 System Management Guide, GC33-1893

MQSeries for VSE/ESA™

MQSeries for VSE/ESA Version 2 Release 1 Licensed Program Specifications, GC34-5365

MQSeries for VSE/ESA Version 2 Release 1 System Management Guide, GC34-5364

MQSeries for Windows

MQSeries for Windows Version 2 Release 0 User's Guide, GC33-1822

MQSeries for Windows Version 2 Release 1 User's Guide, GC33-1965

MQSeries for Windows NT

MQSeries for Windows NT Version 5 Release 1 Quick Beginnings, GC34-5389

MQSeries for Windows NT Using the Component Object Model Interface, SC34-5387

MQSeries LotusScript Extension, SC34-5404

MQSeries Level 1 product publications

For information about the MQSeries Level 1 products, see the following publications:

MQSeries: Concepts and Architecture, GC33-1141

MQSeries Version 1 Products for UNIX Operating Systems Messages and Codes, SC33-1754

MQSeries for UnixWare Version 1 Release 4.1 User's Guide, SC33-1379

Softcopy books

Most of the MQSeries books are supplied in both hardcopy and softcopy formats.

BookManager® format

The MQSeries library is supplied in IBM BookManager format on a variety of online library collection kits, including the *Transaction Processing and Data* collection kit, SK2T-0730. You can view the softcopy books in IBM BookManager format using the following IBM licensed programs:

BookManager READ/2
 BookManager READ/6000
 BookManager READ/DOS
 BookManager READ/MVS
 BookManager READ/VM
 BookManager READ for Windows

HTML format

Relevant MQSeries documentation is provided in HTML format with these MQSeries products:

- MQSeries for AIX V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1 (compiled HTML)
- MQSeries link for R/3 V1.2

The MQSeries books are also available in HTML format from the MQSeries product family Web site at:

<http://www.software.ibm.com/ts/mqseries/>

Portable Document Format (PDF)

PDF files can be viewed and printed using the Adobe Acrobat Reader.

If you need to obtain the Adobe Acrobat Reader, or would like up-to-date information about the platforms on which the Acrobat Reader is supported, visit the Adobe Systems Inc. Web site at:

<http://www.adobe.com/>

PDF versions of relevant MQSeries books are supplied with these MQSeries products:

- MQSeries for AIX V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1

MQSeries on the Internet

- MQSeries for Sun Solaris V5.1
- MQSeries for Windows NT V5.1
- MQSeries link for R/3 V1.2

PDF versions of all current MQSeries books are also available from the MQSeries product family Web site at:

<http://www.software.ibm.com/ts/mqseries/>

PostScript format

The MQSeries library is provided in PostScript (.PS) format with many MQSeries Version 2 products. Books in PostScript format can be printed on a PostScript printer or viewed with a suitable viewer.

Windows Help format

The *MQSeries for Windows User's Guide* is provided in Windows Help format with MQSeries for Windows Version 2.0 and MQSeries for Windows Version 2.1.

MQSeries information available on the Internet

MQSeries Web site

The MQSeries product family Web site is at:

<http://www.software.ibm.com/ts/mqseries/>

By following links from this Web site you can:

- Obtain latest information about the MQSeries product family.
- Access the MQSeries books in HTML and PDF formats.
- Download MQSeries SupportPacs.

Related publications

For information about other products that are referred to in this book, see the following books:

OS/390

- *MVS IPCS Customization*, GC28-1755
- *MVS Initialization and Tuning Reference*, SC28-1752
- *MVS Planning: Workload Management*, GC28-1761
- *MVS Programming: Callable Services for High Level Languages*, GC28-1768
- *MVS Routing and Descriptor Codes*, GC28-1778
- *MVS Setting up a Sysplex* GC28-1779
- *MVS System Management Facilities (SMF)*, GC28-1783
- *MVS Diagnosis: Tools and Service Aids*, LY28-1085
- *OpenEdition Planning*, SC28-1890
- *MVS Resource Measurement Facility User's Guide*, SC28-1916
- *MVS Planning: APPC/MVS Management*, GC28-1807

CICS Transaction Server for OS/390

- *XRF Guide*, SC33-0661
- *Migration Guide*, GC33-1571
- *System Definition Guide*, SC33-1682
- *Customization Guide*, SC33-1683
- *Resource Definition Guide*, SC33-1684
- *Operations and Utilities Guide*, SC33-1685
- *CICS-Supplied Transactions*, SC33-1686
- *Application Programming Guide*, SC33-1687
- *Messages and Codes*, GC33-1694
- *Intercommunication Guide*, SC33-1695
- *Recovery and Restart Guide*, SC33-1698
- *Performance Guide*, SC33-1699
- *RACF Security Guide*, SC33-1701
- *Internet and External Interfaces Guide*, SC33-1944

CICS for MVS/ESA Version 4

- *XRF Guide*, SC33-0661
- *Migration Guide*, GC33-1162
- *System Definition Guide*, SC33-1164
- *Customization Guide*, SC33-1165
- *Resource Definition Guide*, SC33-1166
- *Operations and Utilities Guide*, SC33-1167
- *CICS-Supplied Transactions*, SC33-1168
- *Application Programming Guide*, SC33-1169
- *Messages and Codes*, SC33-1177
- *Intercommunication Guide*, SC33-1181
- *Recovery and Restart Guide*, SC33-1182
- *Performance Guide*, SC33-1183
- *CICS-RACF Security Guide*, SC33-1185

Related publications

IMS

- *Customization Guide*, SC26-8020
- *Operator's Reference*, SC26-8030
- *Administration Guide: System*, SC26-8013
- *Open Transaction Manager Access Guide*, SC26-8026
- *Messages and Codes*, SC26-8028
- *Failure Analysis Structure Tables (FAST) for Dump Analysis*, LY27-9621

DFSMS/MVS

- *Access Method Services for VTAM*, SC26-4905
- *Access Method Services for the Integrated Catalog Facility*, SC26-4906
- *Macro Instructions for Data Sets*, SC26-4913

Security Server

- *Security Server (RACF) Security Administrator's Guide*, SC28-1915
- *Security Server (RACF) Auditor's Guide*, SC28-1916
- *Security Server (RACF) System Programmer's Guide*, SC28-1913
- *Security Server (RACF) External Security Interface (RACROUTE) Macro Reference*, GC28-1922

Other products

- *DFP Storage Administration Reference*, SC26-4566
- *ISPF Dialog Developer's Guide and Reference*, SC28-1273
- *TCP/IP OpenEdition: Planning and Release Guide*, SC31-8303
- *MQSeries Workflow for OS/390: Customization and Administration*, SC33-7030
- *MQSeries Workflow: Concepts and Architecture*, GH12-6285
- *Performance Reporter for OS/390 Administration Guide*, SH19-6816-03
- *Data Facility Hierarchical Storage Manager User's Guide*, SH35-0093
- *APPC Security: MVS/ESA, CICS/ESA, and OS/2*, GG24-3960 (redbook)

What is new for this version

- The minimum levels for many of the items of software required to use MQSeries for OS/390 have changed. These are described in the *MQSeries Planning Guide*.
- There are several changes to the installation process, and some new libraries. These are described in the *MQSeries for OS/390 Program Directory*.
- The OS/390 Automatic Restart Manager (ARM) is now supported. This is described in “Using the OS/390 Automatic Restart Manager (ARM)” on page 294.
- MQSeries now supports clustering. This is described in the *MQSeries Queue Manager Clusters* manual.
- MQSeries now supports C++. This is described in the *MQSeries Using C++* manual.
- MQSeries Batch and TSO applications can now participate in 2-phase commit protocols with other RRS-enabled products, coordinated by OS/390 Resource Recovery Services (RRS). This is described in “Task 15: Set up Batch, TSO, and RRS adapters” on page 87.
- There are new system parameters in CSQZPARM (EXITLIM, EXITTCB, and WLMTIME) and channel initiator parameters in CSQXPARM (TCPTYPE, LU62ARM, and LSTRTMR). If you do not use the supplied default parameter modules, consider whether you need to use these new parameters, and change and link-edit your parameter modules again accordingly.
- If you are using TCP/IP for distributed queuing, you can now use the OpenEdition® sockets interface. This is described “Task 2: Choose the distributed queuing facility” on page 40.
- The CICS bridge is now incorporated into MQSeries for OS/390. This is described in “Introduction to the CICS bridge” on page 147.
- MQSeries can now be used in conjunction with the OS/390 Workload Manager. This is described in “MQSeries, the Workload Manager, and Workflow” on page 26.
- The operations and control panels have been extended and reorganized to accommodate the new clustering function. The panels can now also be used for the security commands. A number of actions have been consolidated into a new PERFORM action. This is described in “Using the operations and control panels” on page 197.
- The supplied default for storage class SYSTEM has been changed to page set 01, so that messages are not put on page set 00. This is described in “Migrating from Version 1.2 to Version 2.1” on page 93.
- The supplied sample input initialization data sets have been reorganized and renamed. This is described in “Task 10: Customize the initialization input data sets” on page 52.
- New code pages are supported for data conversion, including those with the Euro currency symbol. These are described in the *MQSeries Application Programming Reference* manual.

What is new

Part 1. Introduction

Chapter 1. Introduction to MQSeries for OS/390	3
MQSeries and message queuing	3
Time-independent applications	3
Event-driven processing	4
Data integrity and resource protection	4
Application environments	4
Messages and queues	4
What messages are	4
What queues are	5
Objects and commands	7
Object names	7
MQSeries queue managers	8
MQSeries queues	9
Specific local queues used by MQSeries	10
Namelists	13
Channels	13
Process definitions	14
Storage classes	14
An overview of storage management	15
Page sets	15
Storage classes - mapping queues to page sets	15
Page set zero	16
Buffers and buffer pools	17
Logs and recovery	17
Bootstrap data set (BSDS)	17
Events	18
Managing MQSeries with commands and utilities	18
Issuing commands	19
Administrator commands	19
System control commands	20
Utilities	21
Operations and control panels	21
MQSeries and other products	22
Address spaces	22
MQSeries and OS/390 Batch and TSO	22
MQSeries and CICS	23
MQSeries and IMS	25
MQSeries and security	25
MQSeries and SMS	26
MQSeries and ARM	26
MQSeries, the Workload Manager, and Workflow	26
Making MQSeries available	26
Customizing MQSeries and its adapters	27
Operating MQSeries	27
Administering MQSeries	27
Tuning MQSeries	27
Preparing for recovery	28
CSA storage requirement	28
Private region storage usage	28

Chapter 1. Introduction to MQSeries for OS/390

This chapter introduces IBM MQSeries for OS/390¹ from an administrator's perspective and describes:

- The basic concepts of MQSeries and messaging.
- The system-specific things you need to know.
- An outline of the facilities MQSeries for OS/390 provides for system management and administration.
- The relationship between MQSeries and other products.

It contains these sections:

- “MQSeries and message queuing”
- “Messages and queues” on page 4
- “Objects and commands” on page 7
- “An overview of storage management” on page 15
- “Logs and recovery” on page 17
- “Events” on page 18
- “Managing MQSeries with commands and utilities” on page 18
- “MQSeries and other products” on page 22
- “Making MQSeries available” on page 26
- “CSA storage requirement” on page 28

MQSeries and message queuing

MQSeries for OS/390 lets OS/390 applications use message queuing to participate in message-driven processing. With message-driven processing, applications can communicate across different platforms by using the appropriate message queuing software products. For example, OS/390 and OS/400® applications can communicate through MQSeries for OS/390 and MQSeries for AS/400 respectively.

MQSeries products implement a common application programming interface (Message Queue Interface or MQI) whatever platform the applications run on. The calls made by the applications and the messages they exchange are common.

Time-independent applications

With message queuing, the exchange of messages between the sending and receiving programs is time independent. This means that the sending and receiving applications are decoupled so that the sender can continue processing without having to wait for the receiver to acknowledge the receipt of the message.

¹ In the rest of this book, where the context allows, this is referred to simply as MQSeries.

Messages and queues

Event-driven processing

Applications can be started by messages arriving on a queue and, if necessary, terminated when the message or messages have been processed.

Data integrity and resource protection

MQSeries applications can transfer data with an extremely high degree of confidence. Message delivery can involve a *syncpoint* mechanism. This allows for the recovery of important data—using MQSeries logs—if the system fails.

All MQSeries resources, including MQSeries commands, messages, and queues, can be protected using an external security manager such as Resource Access Control Facility (RACF®) (also known as the OS/390 Security Server).

Application environments

With MQSeries for OS/390 you can create applications in these environments:

- An OS/390 Batch and TSO environment
- A CICS transaction environment
- An IMS transaction environment

Applications (or transactions) connect to MQSeries through an *adapter*. There are three types of adapter; one for each of these environments.

Messages and queues

Messages and queues are the basic components of any queuing system.

What messages are

A *message* is a string of bytes that has meaning to the applications that use the message. Messages are used for transferring information from one application to another (or different parts of the same application). The applications can be running in the same environment, or in a different environment. They can be running on the same platform, or on a different platform.

In MQSeries, messages have two parts; the *application data* and a *message descriptor*. The content and structure of the application data is defined by the application programs that use them. The message descriptor identifies the message and contains other control information, such as the message length, the type of message, and the priority assigned to the message by the sending application.

The format of the message descriptor is defined by MQSeries. For a complete description of the message descriptor, see the *MQSeries Application Programming Reference* manual. Figure 1 on page 5 represents an MQSeries message that is logically divided into message descriptor and application data.

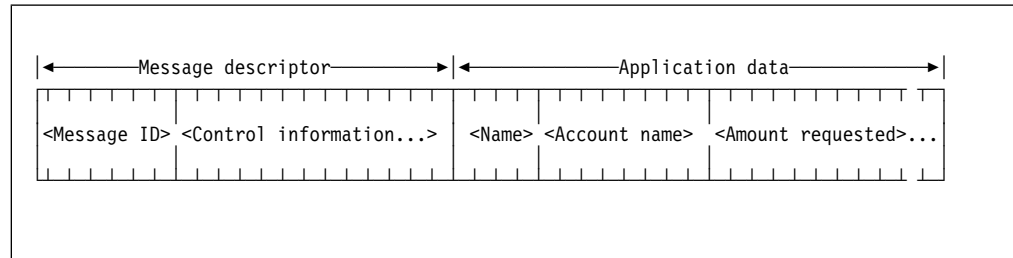


Figure 1. Representation of an MQSeries message. The message descriptor and application data are shown as separate parts. Information that is specific to the application, such as the Account name in this example, is in the application data part of the message.

All applications that participate in message queuing use messages that conform to this common specification. In MQSeries, there are four types of message:

Datagram A simple message for which no reply is expected.

Request A message for which a reply is expected.

Reply A reply to a request message.

Report A message that describes an event, such as the occurrence of an error.

The message type is part of the message descriptor. For more information, see the *MQSeries Application Programming Reference* manual.

Message lengths

In MQSeries for OS/390, the maximum message length is 4 MB (megabytes). In practice, the message length might be limited by:

- The maximum message length defined for the receiving queue.
- The maximum message length defined for the queue manager.
- The maximum message length defined by the applications, when one of the applications is operating in a non-OS/390 platform.
- The amount of storage available for the message.

It might take several messages to send all the information that an application requires.

What queues are

A queue is a data structure that is used to store messages until they are retrieved by an application.

In MQSeries there are three types of queues that hold messages, as defined by the DEFTYPE attribute in the queue definition. These are:

- Predefined
- Permanent dynamic
- Temporary dynamic

Predefined queues are created by the MQSeries command DEFINE QLOCAL. Dynamic queues are created by applications making an **MQOPEN** call against a model queue. (Model queues are described “Using queue objects” on page 10.) Permanent queues, that is, predefined queues and permanent dynamic queues, exist independently of the applications that use them and survive MQSeries

Messages and queues

restarts. Temporary dynamic queues are deleted when the application that created them is stopped, or if MQSeries is stopped.

Queues can exist in main storage, but might be paged out to DASD depending on the usage of that queue. If a queue must be kept for recovery purposes, it is written to DASD. This means that if it needs to be recovered it can be retrieved from DASD.

Each queue belongs to a *queue manager*, which is responsible for maintaining it. The queue manager puts the messages it receives on the appropriate queues.

Applications send and receive messages using MQI calls. For example, to put a message on a queue, the queue is opened for 'puts' by the sending application making an **MQOPEN** call. Then, the application issues an **MQPUT** call to put the message onto that queue. The receiving application must open the same queue for 'gets'. It then issues an **MQGET** to retrieve a message from the queue.

In MQSeries, messages can be retrieved from a queue by suitably authorized applications according to these retrieval algorithms:

- First-in-first-out (FIFO).
- Message priority, as defined in the message descriptor. Messages having the same priority are retrieved on a FIFO basis.
- A program request for a specific message.

Queues can be local or remote. MQSeries for OS/390 can send your messages to queues on another queue manager using its distributed queuing facility. Distributed queuing involves communications links called channels between the queue manager systems; communications can be handled by any of the following:

- TCP/IP
- APPC/MVS (LU 6.2)
- CICS
- SNS/TCPaccess

You can group queue managers in a cluster. Queue managers within the cluster can make the queues that they host available to the whole cluster. Using clusters enables you to:

- Simplify your system administration
- Increase the availability of your queues
- Distribute workload among your queue managers

This is described in the *MQSeries Queue Manager Clusters* manual.

Objects and commands

Many of the tasks described in this book involve manipulating MQSeries *objects*. These objects are common across the different platforms, although some implementations might support a subset of the types listed here.

In MQSeries for OS/390, there are six different types of object:

1. Channels

(Note that if you are using CICS for distributed queuing, channels are not objects, and cannot be manipulated using MQSeries commands.)

2. Queues, subdivided into:

- Local queues
- Remote queues
- Alias queues
- Model queues
- Cluster queues

3. Queue managers

4. Namelists

5. Process definitions

6. Storage classes

These objects can be manipulated by the MQSeries commands described in the *MQSeries Command Reference* manual. For example, the MQSeries command DEFINE QLOCAL, with the appropriate attributes, defines a local queue object. This means that if you issue this command, and the command is successful, you create a new queue. Depending on the attributes provided on the queue definition, applications can then open this queue for putting and getting messages.

Object names

Each queue manager is identified by a unique subsystem name. This name is associated with a *command prefix* string (CPF). See “Using command prefix strings” on page 42. Depending on the task you are performing, a queue manager might be identified by its subsystem name or by its CPF. (For example, you use the CPF if you are entering commands at the OS/390 console, and you use the subsystem name when you are using the operations and control panels.)

For the other types of objects, each object has a name associated with it and can be referenced in MQSeries commands by that name. In general, names must be unique within each of these object types. For example, you can have a queue and a process with the same name, but you cannot have two queues with the same name. This means that you cannot have a local queue with the same name as a model queue, a remote queue, or an alias queue. There are a few exceptions to this rule; these are described where they arise.

MQSeries queue managers

Each MQSeries instance is a queue manager. It is the queue manager that provides queuing services to applications that are connected to it through an adapter. For example, the application can be a CICS transaction, which is connected to the MQSeries queue manager by the CICS adapter.

Figure 2 on page 9 shows two queue managers with applications connected through adapters. The queue manager manages the queues that are local to it, and ensures that outgoing messages are routed correctly. *Distributed queuing* is the part of MQSeries that handles communications with other queue managers. (MQSeries applications can also access CICS and IMS applications directly through bridges; these are introduced in “MQSeries and other products” on page 22.)

A queue is a *local queue* if it is managed by the same queue manager that is connected to the application. If the queue is managed by a different queue manager, it is called a *remote queue*. In Figure 2 on page 9 queues A1 and A2 are local queues for applications 1, 2, and 3. However, for applications 4, 5, and 6, they are remote queues.

Queue manager objects

A queue manager provides queuing services to applications and commands to administrators, but it is also an MQSeries object in its own right. Because it is an object, a queue manager can be used in both MQI calls and MQSeries commands.

MQI calls: A queue manager object can be used in some MQI calls. For example, you can inquire about the attributes of the queue manager object using the MQI call **MQINQ**. However, you cannot put messages on a queue manager object using an **MQPUT** call. Messages are always put to queue objects, not to queue manager objects. See the *MQSeries Application Programming Reference* manual for further details of the MQI calls that can be used with a queue manager object.

MQSeries commands: The queue manager object has attributes, which can be displayed using the MQSeries command **DISPLAY QMGR**. Some of these attributes can be changed using the **ALTER QMGR** command. However, unlike the other MQSeries objects, there are no commands to define or delete the queue manager object. Once a queue manager is installed and customized, it is always in existence.

For more information, see Chapter 3, “Customizing the queue managers” on page 37 and “Managing MQSeries with commands and utilities” on page 18.

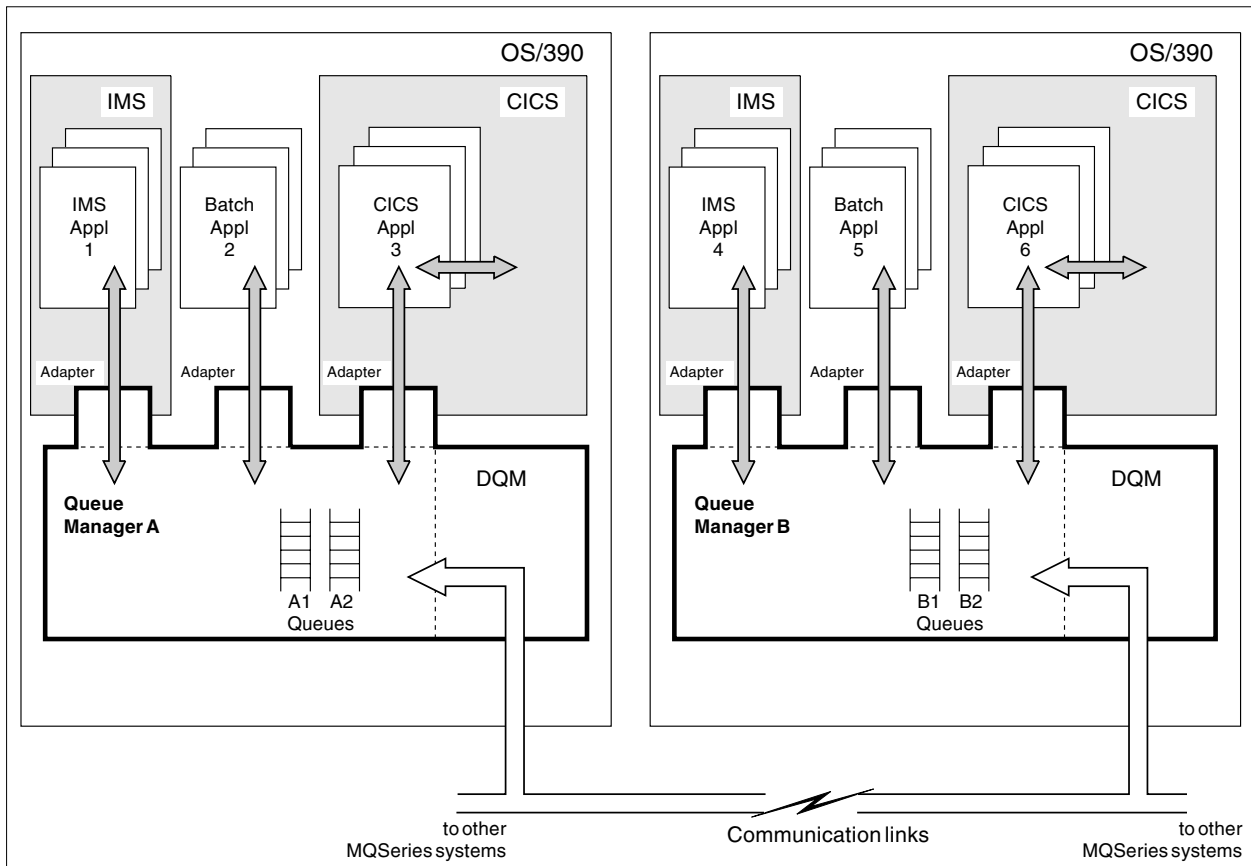


Figure 2. MQSeries for OS/390 overview

MQSeries queues

Queues are defined to MQSeries using the appropriate DEFINE commands for queues. For example, you use the DEFINE QLOCAL command to define a local queue. These commands specify the attributes of the queue being defined. For example, a local queue object has attributes that specify what happens when applications reference that queue in MQI calls. Some of the things that the attributes of a queue determine are:

- Whether applications can retrieve messages from the queue (get enabled).
- Whether applications can put messages on the queue (put enabled).
- Whether access to the queue is exclusive to one application or shared between applications.
- The maximum number of messages that can be stored on the queue at the same time (maximum queue depth).
- The maximum length of messages that can be put on the queue.

This list is only a subset; for full details about queue attributes and the commands for defining and manipulating queue objects, see the *MQSeries Command Reference* manual.

Using queue objects

In MQSeries, there are five types of queue object. Each type of object can be manipulated by MQSeries commands and is associated with real queues in different ways:

1. A *local queue* object identifies a local queue belonging to the queue manager to which the application is connected. Messages always end up on a queue that is a local queue on the destination queue manager.
2. A *remote queue* object identifies a queue belonging to another queue manager. This queue, therefore, must be defined as a local queue to the remote queue manager. The information you specify when you define a remote queue object allows the queue manager to find the remote queue manager, so that any messages destined for the remote queue go to the correct queue manager.
3. An *alias queue* object lets applications access a queue by referring to it indirectly in MQI calls. When an alias queue name is used in an MQI call, the name is resolved to the name of either a local or a remote queue at run time. This lets you to change the queues that applications use without changing the application in any way—you merely change the alias queue definition to reflect the name of the new queue that the alias resolves to.
4. A *model queue* object defines a set of queue attributes that are used as a template for creating a dynamic queue. Dynamic queues are created by the queue manager when an application makes an **MQOPEN** request specifying a queue name that is the name of a model queue. The dynamic queue that is created in this way is a local queue whose attributes are taken from the model queue definition. The dynamic queue name can be specified by the application or, if the application does not specify one, the queue manager generates a unique name, starting “CSQ.” and returns it to the application.

Dynamic queues defined in this way can be *temporary queues*, which do not survive MQSeries restarts, or *permanent queues*, which do.
5. A *cluster queue* object represents a queue, defined on this queue manager or elsewhere in the network, that is in a cluster. MQSeries clusters are described in the *MQSeries Queue Manager Clusters* manual.

Use the appropriate DEFINE command to create one of these types of queue objects. These commands are described in the *MQSeries Command Reference* manual. You can also use the default queue objects supplied with MQSeries as the basis of your definitions. These defaults are described in “CSQ4INSG system object sample” on page 55.

Specific local queues used by MQSeries

MQSeries uses some local queues for specific purposes related to its operation. These queues must be defined before MQSeries can use them.

The MQSeries administrator is responsible for defining and maintaining all queues using the information in this book and the *MQSeries Command Reference* manual.

Initiation queues

An *initiation queue* receives *trigger messages*; these show that trigger events have occurred. Typically, a trigger event occurs when a message is put on a queue so that the trigger conditions for that queue are met. For example, the trigger conditions could be that a trigger event occurs when the number of messages on the queue reaches a predefined depth. The trigger event causes the queue manager to put a trigger message onto the initiation queue for that queue. This trigger message is retrieved from the initiation queue by a trigger monitor application. The trigger monitor then starts up the application program that was specified in the trigger message.

If a queue manager is to use triggering, at least one initiation queue must be defined for that queue manager.

Transmission queues

A *transmission queue* temporarily stores messages that are destined for a remote queue manager. You must define at least one transmission queue for each remote queue manager to which the local queue manager is to send messages directly, unless you are using queue manager clusters. In this case, you only need to define one transmission queue to handle messages for all destinations. For details about the use of transmission queues in distributed queuing, see the *MQSeries Intercommunication* manual.

Channel queues

Channels are objects used in distributed queuing to define how communication between queue managers takes place. The following local queues are required to support channels for distributed queuing:

- A *channel initiation queue* is needed to send commands to channels. This queue must be called `SYSTEM.CHANNEL.INITQ`.
- A *channel synchronization queue* is used for sequential message numbering for handling unit of work information when MQSeries subsystems are communicating. Each queue manager needs the channel synchronization queue if it is to receive or send messages to another queue manager (regardless of the number of other queue managers involved). This queue must be called `SYSTEM.CHANNEL.SYNCQ`.
- A *channel reply information queue* is needed to handle replies from channel commands. This queue must be called `SYSTEM.CHANNEL.REPLY.INFO`.

Different local queues are required if you are using CICS ISC for distributed queuing. These are described in Appendix D, “Enabling distributed queuing using CICS ISC” on page 479.

Cluster support queues

The following local queues are required to support clustering:

- A local queue is needed to hold a persistent copy of the repository. (The repository is a collection of information about the queue managers that are members of the cluster, held on some of the queue managers in the cluster and replicated on the other queue managers.) This queue is called `SYSTEM.CLUSTER.REPOSITORY.QUEUE`.
- A local queue is needed to communicate repository changes between queue managers. This queue is used for information about updates to the repository

data to be applied to the local repository, or requests for repository data. This queue is called `SYSTEM.CLUSTER.COMMAND.QUEUE`.

- A local queue is needed as the transmission queue for all destinations in the cluster. This queue is called `SYSTEM.CLUSTER.TRANSMIT.QUEUE`.

For information about MQSeries clusters, see the *MQSeries Queue Manager Clusters* manual.

Dead-letter queue

A *dead-letter queue* receives messages that cannot be routed to their correct destinations. This occurs when, for example:

- The destination queue is full
- Message puts have been inhibited on the destination queue
- The sender is not authorized to use the destination queue
- The destination queue does not exist

For distributed queuing, it is strongly recommended that you define a dead-letter queue for each queue manager involved. If you do not do this, a channel will close if MQSeries is unable to deliver a message, and distributed queuing will stop.

System-command input queue

The *system-command input queue* (`SYSTEM.COMMAND.INPUT`) is a local queue to which suitably authorized applications can send MQSeries commands. These commands are then retrieved by an MQSeries component called the command server. The command server validates the commands and passes the valid ones on for processing by the MQSeries command processor. See also “Using the command server” on page 213.

A system-command input queue must be defined for each queue manager.

The supplied default model queue `SYSTEM.COMMAND.REPLY.MODEL` is used by the CSQUTIL utility, the distributed queuing features, and the MQSeries operations and control panels. The operations and control panels use dynamic queues generated from this model queue to receive replies to MQSeries commands. These replies can then be viewed by users from the operations and control panels. The dynamic queue names are of the form `SYSTEM.CSQUTIL.★`, `SYSTEM.CSQXCMD.★`, and `SYSTEM.CSQREXX.★` respectively.

Event queues

An *event queue* is a local or remote queue used to hold event messages. Event messages are generated by the queue manager to help you monitor your system.

The following event queues are required:

- `SYSTEM.ADMIN.QMGR` to hold messages about queue manager events
- `SYSTEM.ADMIN.PERFM.EVENT` to hold messages about performance events
- `SYSTEM.ADMIN.CHANNEL.EVENT` to hold messages about channel events

For information about MQSeries events, see “Events” on page 18.

System default queues

The *system default queues* are a set of queue definitions supplied with MQSeries. You can copy and rename any of these queue definitions for use in applications at your installation.

For example, to define a local queue, you can copy the supplied default SYSTEM.DEFAULT.LOCAL.QUEUE, change its name, and then alter any of its other attributes, as required.

Namelists

A *namelist* is an MQSeries object that contains a list of other MQSeries objects. Namelists are maintained independently of applications, that is, they can be updated without stopping any of the applications that use them.

Use the DEFINE NAMELIST command to create a new namelist. You can also use the default namelist object, SYSTEM.DEFAULT.NAMELIST (supplied with MQSeries) as the basis of your own definitions. System default objects are described in “CSQ4INSG system object sample” on page 55.

Channels

A channel provides a communication path. There are two types of channel, message channels and MQI channels.

A *message channel* provides a communication path between two queue managers. The message channel is used for the transmission of messages from one queue manager to another, and shields the application programs from the complexities of the underlying networking protocols.

A message channel can transmit messages in one direction only. If two-way communication is required between two queue managers, two message channels are required.

In order to set up a channel, you usually define one channel definition for each end of the channel. There are six types of message channel: Sender, Server, Receiver, Requester, Cluster-sender and Cluster-receiver.

An *MQI channel* connects an MQSeries client to a queue manager on a server machine. It is for the transfer of MQI calls and responses only and is bidirectional. A channel definition exists for each end of the link. There are two types of MQI channel: Server-connection and Client-connection.

For distributed queuing (without CICS ISC), channels are MQSeries objects. Use the DEFINE CHANNEL command to create a new channel. You can also use the default channel objects (supplied with MQSeries for OS/390) as the basis of your own definitions. This is described in “CSQ4INSX system object sample” on page 56. If you are using clustering, most channels are created automatically.

For distributed queuing using CICS ISC, channels are maintained by a CICS transaction called CKMC.

Process definitions

A *process definition object* defines an application that is to be started in response to a trigger event on an MQSeries queue manager. In MQSeries, an application retrieves messages from one or more specified queues and processes them. An application can be any of the following types:

- CICS transactions
- IMS transactions
- TSO or batch programs

This definition includes the application ID, the application type, and data specific to the application.

Use the DEFINE PROCESS command to create a process definition. You can also use the default process definition object, SYSTEM.DEFAULT.PROCESS (supplied with MQSeries) as the basis of your own definitions. System default objects are described in “CSQ4INSG system object sample” on page 55.

Trigger monitors and task initiators

A *trigger monitor* is an application that monitors an initiation queue associated with a queue manager (see “Initiation queues” on page 11). When a trigger message arrives on the initiation queue, it is retrieved by the trigger monitor. Typically, the trigger monitor then starts an application that is specified in the message on the initiation queue.

MQSeries supplies trigger monitors for use in the following environments:

- CICS This is referred to in this book as the CICS *task initiator* transaction, CKTI. See “Task initiator” on page 113 for a description of this transaction and Chapter 7, “Operating the CICS adapter” on page 123 for how to use it.
- IMS This is referred to in this book as the IMS transaction CSQQTRMN. See “The IMS trigger monitor” on page 160 for a description of this transaction and Chapter 10, “Operating the IMS adapter” on page 169 for how to use it.

Storage classes

See “Storage classes - mapping queues to page sets” on page 15 for information about storage classes.

An overview of storage management

In MQSeries for OS/390, storage management involves using these entities:

- Page sets
- Storage classes
- Buffer pools

Page sets

A *page set* is a linear VSAM data set that has been specially formatted to be used in MQSeries. Page sets are used to store messages and object definitions. Each page set is identified by a page set identifier (PSID), an integer in the range 00 through 99. MQSeries uses page set zero (PSID=00) to store object definitions and other important information relevant to the queue manager subsystem.

Each queue manager must have its own page sets including a page set zero.

Page sets must be formatted and so MQSeries provides a FORMAT utility for this, see “Formatting page sets (FORMAT)” on page 231. Page sets must also be defined to the MQSeries subsystem, see “Task 12: Define your page sets” on page 62.

If you define secondary extents for your page sets, MQSeries for OS/390 expands a page set dynamically if it becomes full. MQSeries continues to expand the page set if required until 123 logical extents exist, provided that there is sufficient disk storage space available.

Note: You cannot use page sets from one MQSeries subsystem on a different MQSeries subsystem, or change the subsystem name. If you want to transfer the data from one subsystem to another, you must unload all the objects and messages from the first subsystem and reload them onto another.

Storage classes - mapping queues to page sets

Storage classes allow you to control where message data is stored for administrative, data set space and load management, or application isolation purposes.

A *storage class* maps one or more queues to a page set. This means that messages for that queue are stored (subject to buffering) on that page set. This is how it works:

- You define a storage class, using the DEFINE STGCLASS command, specifying a page set identifier (PSID).
- When you define a queue, you can specify a storage class in the STGCLASS attribute.

In the following example, the local queue QE5 is mapped to page set 21 through storage class ARC2.

```
DEFINE STGCLASS(ARC2) PSID(21)
DEFINE QLOCAL(QE5) STGCLASS(ARC2)
```

Storage management overview

This means that messages that are put on the queue QE5 are stored on page set 21 (if they stay on the queue long enough to be written to DASD).

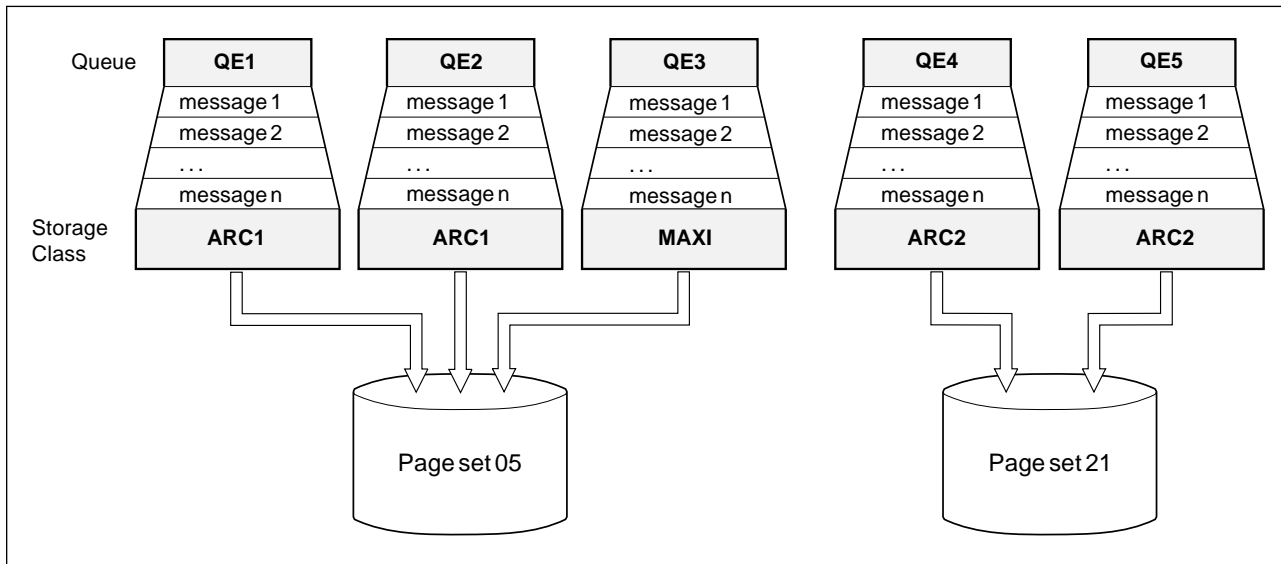


Figure 3. Mapping queues to page sets through storage classes

More than one queue can use the same storage class and you can define as many storage classes as you like. For example, you can extend the previous example to include more storage class and queue definitions, as follows:

```
DEFINE STGCLASS(ARC1) PSID(05)
DEFINE STGCLASS(ARC2) PSID(21)
DEFINE STGCLASS(MAXI) PSID(05)
DEFINE QLOCAL(QE1) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE2) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE3) STGCLASS(MAXI) ...
DEFINE QLOCAL(QE4) STGCLASS(ARC2) ...
DEFINE QLOCAL(QE5) STGCLASS(ARC2) ...
```

In Figure 3, both storage classes ARC1 and MAXI are associated with page set 05. Therefore, the queues QE1, QE2, and QE3 are mapped to page set 05. Similarly, storage class ARC2 associates queues QE4 and QE5 with page set 21.

If you define a queue without specifying a storage class, MQSeries uses the default storage class.

Page set zero

Page set zero is used to store all the object definitions required by the queue manager. For normal operation of MQSeries, it is essential that page set zero does not become full. For performance reasons, it is best not to have object definitions and messages on the same page set. Therefore, we recommend that you do not define storage classes that map to page set zero. If you do, you run the risk of filling page set zero if too many messages accumulate there. This would mean that any MQSeries facility that uses the system-command input queue, for example, the operations and control panels, would fail. Also, applications would be unable to create permanent dynamic queues.

Buffers and buffer pools

For efficiency, MQSeries uses a form of caching whereby messages (and object definitions) are stored temporarily in buffers before being stored in page sets on DASD. Short-lived messages, that is, messages that are retrieved from a queue shortly after they are received, might only ever be stored in the buffers. However, this is all transparent to the user because the buffers are controlled by a buffer manager, which is a component of MQSeries.

The buffers are organized into *buffer pools*. You can define up to four buffer pools (0 through 3) for each MQSeries subsystem; you are recommended to use four buffer pools. Each buffer is 4 KB long. The maximum number of buffers is determined by the amount of storage available in the MQSeries address space, although you should not use more than about 70% of the space for buffers. Usually, the more buffers you have, the more efficient the buffering and the better the performance of the MQSeries subsystem.

Figure 4 on page 18 shows the relationship between messages, buffers, buffer pools, and page sets. A buffer pool is associated with one or more page sets; each page set is associated with a single buffer pool.

You specify the number of buffers in a pool with the DEFINE BUFFPOOL command. For details of this command, see the *MQSeries Command Reference* manual.

For performance reasons, messages and object definitions should not be in the same buffer pool. You are recommended therefore to use one buffer pool (say number zero) exclusively for page set zero, where the object definitions are kept. Similarly, short-lived messages and long-lived messages should be kept in different buffer pools and therefore on different page sets, and in different queues.

Logs and recovery

MQSeries records all persistent messages, object definitions, and significant events as they occur in one of a cycle of log data sets. When this *active log* is full, MQSeries switches the active log to the next data set, and copies the contents of the full log to an *archive log*, which can be a data set on a direct access storage device (DASD) or magnetic tape. If there is a subsystem problem, MQSeries uses these log entries to restore itself to a consistent state. In particular, persistent messages can be recovered over MQSeries subsystem restarts. For greater assurance against, for example, DASD failure, MQSeries supports dual logging for both the active and the archive logs.

For a complete description of logs, their contents, and archiving, see Chapter 17, “Understanding the log and the bootstrap data set” on page 299.

Bootstrap data set (BSDS)

The *bootstrap data set* (BSDS) is a VSAM key-sequenced data set (KSDS) that holds information required by MQSeries for recovery. It contains an inventory of all active and archived log data sets known to MQSeries and a *wrap-around inventory* of all recent MQSeries activity, which is needed if MQSeries has to be restarted. MQSeries also supports dual BSDSs.

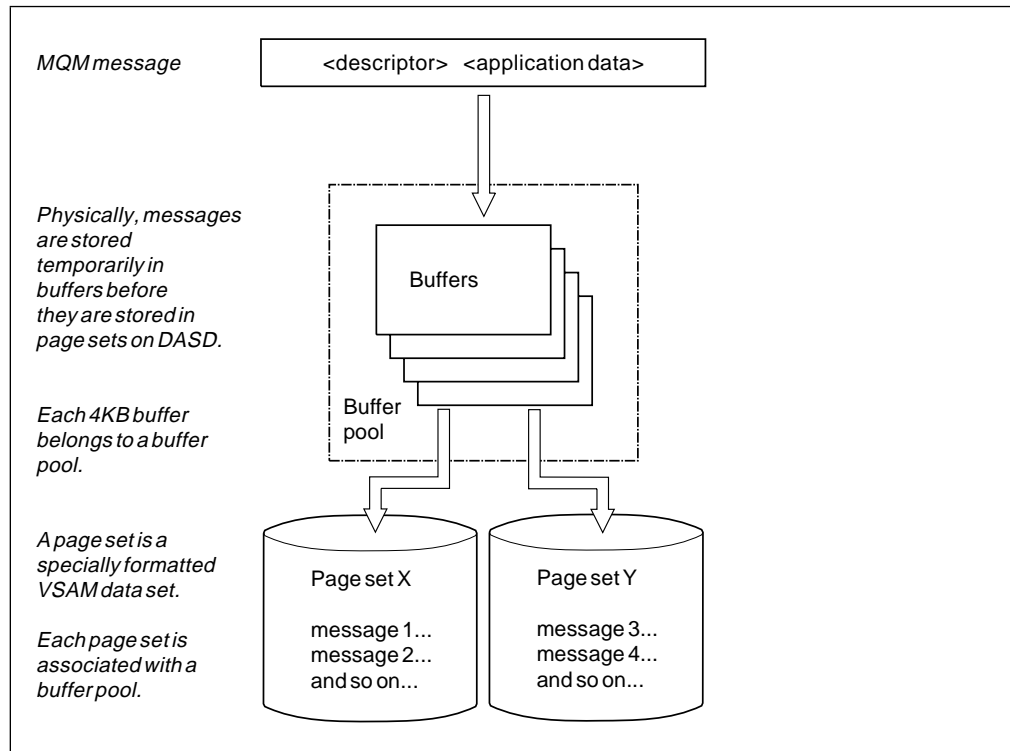


Figure 4. Buffers, buffer pools, and page sets

For a complete description of the functions and uses of the BSDS, see "What the bootstrap data set is for" on page 305.

Events

MQSeries events provide information about errors, warnings, and other significant occurrences in a queue manager. By incorporating these events into your own system management application, you can monitor the activities across many queue managers, for multiple MQSeries applications. In particular, you can monitor all the queue managers in your system from a single queue manager.

Events can be reported through a user-written reporting mechanism to an administration application that supports the presentation of the events to an operator. Events also enable applications acting as agents for other administration networks, for example NetView®, to monitor reports and create the appropriate alerts.

For more information about using events, see the *MQSeries Programmable System Management* manual.

Managing MQSeries with commands and utilities

MQSeries provides a set of commands and a set of utilities to help you to manage MQSeries at your installation.

This section gives an overview of the following:

- Issuing commands
- Administrator commands

- System control commands
- Utilities
- Operations and control panels

For more information on all MQSeries commands, including the command syntax, see the *MQSeries Command Reference* manual.

Issuing commands

The commands can be issued from:

- The initialization input data sets (see page 52)
- The OS/390 console (see page 190)
- The system-command input queue (see page 211)
- The COMMAND function of the CSQUTIL utility (see page 229)
- The operations and control panels (see page 197)

For more information, see “Issuing commands” on page 189.

Administrator commands

You use the administrator commands to manage these MQSeries objects:

- Channels (for distributed queuing without CICS ISC)
- Namelists
- Processes
- Queues
- Queue managers
- Storage classes

For each of these types of objects you can use these types of commands:

Command type	Purpose
ALTER	Changes an existing object definition
DEFINE	Defines the attributes of an object and creates the object (not the queue manager)
DISPLAY	Displays the attributes of an object
DELETE	Deletes an object definition (not the queue manager)

Table 1 shows the administrator commands that are available.

	ALTER	DEFINE	DISPLAY	DELETE
CHANNEL	√	√	√	√
NAMELIST	√	√	√	√
PROCESS	√	√	√	√
QALIAS	√	√	√	√
QCLUSTER			√	
QLOCAL	√	√	√	√
QMGR	√		√	
QMODEL	√	√	√	√
QREMOTE	√	√	√	√

<i>Table 1 (Page 2 of 2). Summary of MQSeries administrator commands</i>				
	ALTER	DEFINE	DISPLAY	DELETE
QUEUE			√	
STGCLASS	√	√	√	√

System control commands

You can use the system control commands to manage entities specific to MQSeries for OS/390, such as page sets and buffer pools. Table 2 summarizes the MQSeries system control commands.

<i>Table 2 (Page 1 of 2). System control commands</i>		
This command...	Deals with...	For these tasks...
ARCHIVE LOG	Logs	Copying the current active log to an archive log.
DEFINE BUFFPOOL	Buffer pools	Defining a buffer pool and the number of 4 KB buffers it contains.
DEFINE MAXSMSGS DISPLAY MAXSMSGS	Messages	Defining the maximum number of messages that a task can get or put within a single unit of recovery.
DEFINE PSID	Page sets	Defining a page set and an associated buffer pool.
DISPLAY THREAD	Threads	Displaying information about a thread.
DISPLAY USAGE	Page sets	Displaying the current state of a page set.
RECOVER BSDS	BSDS	Re-establishing a dual bootstrap data set that had a data set error.
REFRESH SECURITY RVERIFY SECURITY ALTER SECURITY DISPLAY SECURITY	Security	Performing tasks associated with security, for example, refresh security if you change security profiles; changing and displaying security options.
RESOLVE INDOUBT	Threads	Resolving in-doubt threads.
START CMDSERV STOP CMDSERV DISPLAY CMDSERV	Command server	Starting or stopping the command server, displaying command server attributes.
START TRACE STOP TRACE ALTER TRACE DISPLAY TRACE	Traces	Starting or stopping MQSeries traces, changing and displaying trace parameters.
START QMGR STOP QMGR SUSPEND QMGR RESUME QMGR	Queue managers	Starting and stopping queue managers, joining and leaving clusters.
START CHANNEL STOP CHANNEL PING CHANNEL RESET CHANNEL RESOLVE CHANNEL DISPLAY CHSTATUS	Channels	Starting and stopping channels, testing a channel, resetting channel sequence numbers, resolving in-doubt messages, and displaying channel status.

<i>Table 2 (Page 2 of 2). System control commands</i>		
This command...	Deals with...	For these tasks...
START CHINIT STOP CHINIT DISPLAY DQM	Channel initiators	Starting and stopping channel initiators, display information about channel initiators.
START LISTENER STOP LISTENER	Channel listeners	Starting and stopping channel listeners.
RESET TPIPE	IMS transaction pipes (Tpipe)	Resetting sequence numbers for an IMS Tpipe.
DISPLAY CLUSQMGR	Queue managers	Displaying cluster information about queue managers in a cluster.
REFRESH CLUSTER RESET CLUSTER	Clusters	Refreshing locally-held cluster information, performing special cluster operations.

Utilities

MQSeries also provides the following utilities to help you perform other administrative and management tasks:

- MQSeries utility program, CSQUTIL; you can use this utility for:
 - Managing page sets
 - Issuing system administrator and system control commands from an OS/390 batch program
 - Managing queues
- Data conversion exit utility program, CSQUCVX
- Change log inventory program, CSQJU003, for modifying the bootstrap data set
- Print log map utility, CSQJU004, for listing information about the log
- Log print utility, CSQ1LOGP, for printing the log

The MQSeries utilities are described in Chapter 14, “Using the MQSeries utilities” on page 227.

Operations and control panels

MQSeries provides a set of operations and control panels that enable you to manage MQSeries objects interactively. The panels use the Interactive System Productivity Facility (ISPF).

The operations and control panels are available in the following national languages:

- US English (mixed case)
- US English (uppercase)
- Japanese
- Simplified Chinese

depending on which language feature you have installed.

Both the Time Sharing Option/Extensions (TSO/E) and the Interactive Systems Productivity Facility (ISPF) are required if you want to use the operations and control panels supplied with MQSeries.

MQSeries and other products

For more details about the operations and control panels and how they are used, see “Using the operations and control panels” on page 197.

MQSeries and other products

This section describes some of the other products you can use with MQSeries and how these products are used.

MQSeries operates as a formal subsystem of OS/390. MQSeries connects to the application environments through *adapters*. Adapters are provided for each of these environments:

- Batch and TSO
- CICS
- IMS

Address spaces

There is one MQSeries address space and there are *allied address spaces* for each environment in which the applications run.

Each TSO user, each batch program, and each IMS program has its own allied address space. For the CICS adapter, there is one CICS address space and that is for the CICS region (not for individual CICS programs).

MQSeries and OS/390 Batch and TSO

The Batch/TSO adapters are the interface between OS/390 application programs running under JES, TSO, or OS/390 OpenEdition and an MQSeries subsystem. They enable OS/390 application programs to use the MQI.

The adapters provide access to MQSeries resources for programs running in:

- Task (TCB) mode
- Problem or supervisor state
- Non-cross-memory mode
- Non-access register mode

Connections between application programs and MQSeries are at the task level. The adapters provide a connection thread from an application task control block (TCB) to MQSeries.

The Batch/TSO adapter supports a single-phase commit protocol for changes made to resources owned by MQSeries. It does not support multi-phase commit protocols. The RRS adapter enables MQSeries applications to participate in two-phase commit protocols with other RRS-enabled products, coordinated by OS/390 Resource Recovery Services (RRS).

The adapters use the OS/390 STIMERM service to schedule an asynchronous event every second. This event runs an interrupt request block (IRB) that does not involve any waiting by the batch application's task. This IRB checks to see if the MQSeries termination ECB has been posted. If the termination ECB has been posted, the IRB posts any application ECBs that are waiting on an event in the MQSeries subsystem (for example, a signal or a wait).

The MQSeries-Batch/TSO adapter

The MQSeries Batch/TSO adapter provides MQSeries support for OS/390 Batch and TSO applications. All application programs that run under OS/390 Batch or TSO must have the stub CSQBSTUB link-edited with them. The stub provides the application with access to all MQI calls. You use single-phase commit and backout for applications by issuing the MQI calls **MQCMIT** and **MQBACK**.

For information about building application programs that use the Batch/TSO adapter, see the *MQSeries Application Programming Reference* manual.

The MQSeries-Batch/TSO RRS adapter

Resource Recovery Services (RRS) is a subcomponent of OS/390 that provides a system-wide service for coordinating two-phase commit across OS/390 products. The MQSeries Batch/TSO RRS adapter (the RRS adapter) provides MQSeries support for OS/390 Batch and TSO applications that want to use these services. The RRS adapter enables MQSeries to become a full participant in RRS coordination. Applications can participate in two-phase commit processing with other products that support RRS (for example, DB2®).

The RRS adapter provides two stubs; application programs that want to use RRS must be link-edited with one of these stubs.

- CSQBRSTB allows you to use two-phase commit and backout for applications by using the RRS callable resource recovery services instead of the MQI calls **MQCMIT** and **MQBACK**. The callable resource recovery services of RRS are described in the *MVS Programming: Callable Services for High Level Languages* manual. You must also link-edit module ATRSCSS from library SYS1.CSSLIB with your application. If you use the MQI calls **MQCMIT** and **MQBACK**, you will receive return code MQRC_ENVIRONMENT_ERROR.
- CSQBRSSI allows you to use MQI calls **MQCMIT** and **MQBACK**; MQSeries actually implements these calls as the **SRRCMIT** and **SRRBACK** RRS calls.

For information about building application programs that use the RRS adapter, see the *MQSeries Application Programming Reference* manual.

MQSeries and CICS

The MQSeries-supplied CICS adapter gives you access to MQSeries from CICS. You can start and stop CICS and MQSeries independently, and you can establish or terminate a connection between them at any time. You can also allow CICS to connect to MQSeries automatically.

The CICS adapters use the CICS task-related user exit (TRUE) facility. This facility is described in the *CICS Customization Guide*.

In a CICS multiregion operation or intersystem communication (ISC) environment, each CICS address space can have its own attachment to the queue manager subsystem. A single CICS address space can be connected to only one queue manager at a time. However, multiple CICS address spaces can connect to the same MQSeries subsystem.

The CICS adapter also lets you use distributed queuing to send messages to and receive messages from other queue managers using CICS communication facilities. For more information, see the *MQSeries Intercommunication* manual.

The CICS adapter provides CICS applications with access to MQSeries data while operating in the CICS environment. Therefore, CICS applications connected to MQSeries can access both MQSeries data and CICS data.

Usually, if CICS or MQSeries terminate or if the application terminates, CICS coordinates the recovery of both MQSeries data and CICS data. However, if messages are being sent from one queue manager to another using distributed queuing, recovery is coordinated by the sender Message Channel Agent (MCA). For more information about distributed queuing, see the *MQSeries Intercommunication* manual.

You can use MQSeries with the CICS Extended Recovery Facility (XRF) to aid recovery from a CICS error. For more information about XRF, see “Using Extended Recovery Facility” on page 312 and the *CICS/ESA Version 3.3 XRF Guide*.

The CICS adapter uses standard CICS command-level services where required, for example, EXEC CICS WAIT and EXEC CICS ABEND. A portion of the CICS adapter runs under the control of the transaction issuing the messaging requests. Therefore, these calls for CICS services appear to be issued by the transaction. For more information, see Chapter 6, “The MQSeries-CICS adapter” on page 109.

Application programming with CICS

The CICS adapter provides MQSeries support for CICS applications. All application programs that run under CICS must have the stub CSQCSTUB link-edited with them if they are to access MQSeries, unless the program is using dynamic calls. (For information about calling the CICS stub dynamically, see the *MQSeries Application Programming Guide*.) This stub provides the application with access to all MQI calls. For two-phase commit and backout, applications must use the appropriate EXEC CICS commands.

For information about building application programs that use the CICS adapter, see the *MQSeries Application Programming Guide*.

System administration and operation with CICS

An authorized CICS terminal operator can issue CICS commands to control and monitor the CICS adapter. However, the CICS terminal operator has no control over the MQSeries address space. For example, the operator cannot shut down MQSeries from the CICS address space.

The MQSeries-CICS bridge

The MQSeries-CICS bridge enables an application, not running in a CICS environment, to run a *program* or *transaction* on CICS and get a response back. This non-CICS application can be run from any environment that has access to an MQSeries network that encompasses MQSeries for OS/390.

A *program* is a CICS program that can be invoked using the EXEC CICS LINK command. It must conform to the DPL subset of the CICS API, that is, it must not use CICS terminal or syncpoint facilities. A *transaction* is a CICS transaction designed to run on a 3270 terminal. This transaction can use BMS or TC commands. It can be conversational or part of a pseudoconversation. It is permitted to issue syncpoints.

For more information, see “Introduction to the CICS bridge” on page 147.

MQSeries and IMS

The IMS adapter provided with MQSeries gives access to MQSeries from IMS. The IMS adapter receives and interprets requests for access to MQSeries using the External Subsystem Attach Facility (ESAF) provided by IMS. This facility is described in the *IMS Customization Guide*. Usually, IMS connects to MQSeries automatically without operator intervention. (For more information see Chapter 9, “The MQSeries-IMS adapter” on page 159.)

You can use MQSeries with the IMS Extended Recovery Facility (XRF) to aid recovery from a IMS error. For more information about XRF, see “Using Extended Recovery Facility” on page 312, and the *IMS Administration Guide: System manual*.

Application programming with IMS

With the IMS adapter, MQSeries provides message queuing services for IMS-dependent regions. All application programs that run under IMS must have the stub CSQQSTUB and the IMS language interface module link-edited with them if they need to access MQSeries. This stub provides the application with access to all MQI calls. To use two-phase commit and backout, your application should use the appropriate IMS calls. Batch DL/I programs must use the batch adapter stub, CSQBSTUB (see “The MQSeries-Batch/TSO adapter” on page 23).

For information about building application programs that use the IMS adapter, see the *MQSeries Application Programming Guide*.

The MQSeries-IMS bridge

The MQSeries-IMS bridge is the component of MQSeries for OS/390 that allows direct access from MQSeries applications to applications on your IMS system. The bridge enables implicit MQSeries API support. This means that you can re-engineer legacy applications that were controlled by 3270-connected terminals to be controlled by MQSeries messages, without having to rewrite, recompile, or re-link them.

The bridge is an IMS *Open Transaction Manager Access* (OTMA) client. For more information, see “Introduction to the IMS bridge” on page 179.

System administration and operation with IMS

An authorized IMS terminal operator can issue IMS commands to control and monitor the connection to MQSeries. However, the IMS terminal operator has no control over the MQSeries address space. For example, the operator cannot shut down MQSeries from an IMS address space.

MQSeries and security

MQSeries uses the OS/390 System Authorization Facility (SAF) to route authorization requests to an external security manager, for example, the Resource Access Control Facility (RACF). MQSeries does no security verification of its own.

For details of how security is carried out in MQSeries, see Part 8, “Security” on page 391.

MQSeries and SMS

MQSeries parameters enable you to specify Storage Management Subsystem (MVS/DFP™ SMS) storage classes when dynamically allocating MQSeries archive data sets. MQSeries initiates the archiving of log data sets, but SMS can be used to perform allocation of the archive data set.

MQSeries and ARM

The OS/390 Automatic Restart Manager (ARM) is an OS/390 recovery function. You can use it to improve the availability of your MQSeries subsystems.

When a job or task fails, or the system on which it is running fails, ARM can restart the job or task without operator intervention. This means that, for MQSeries, you do not have to wait for operations staff to notice that a queue manager has failed, and to take corrective action. Instead, OS/390 notices if either a queue manager or a channel initiator has failed, or if an OS/390 system has failed. This results in a faster resumption of productive work, and hence in improved system availability.

Using ARM with MQSeries is described in “Using the OS/390 Automatic Restart Manager (ARM)” on page 294.

MQSeries, the Workload Manager, and Workflow

The OS/390 *Workload Manager* (WLM) provides a solution for managing workload distribution, workload balancing, and distributing resources to competing workloads. You can set service classes for each different type of work performed by your system, and then set service goals for each class of service. The WLM allocates system resources to achieve these goals.

To use the queuing services of the workload manager with MQSeries, you need to use *Workflow*. You need to set up OS/390 WLM class qualifiers for MQSeries, and use the WLM to define different classes of service for your MQSeries messages. When you have done this, if you put messages on a special queue called a *WLM-managed queue*, Workflow passes information taken from the message descriptor and a work information header in the message to the WLM, and so determine which class of service to use for the message.

For a general explanation of the basic concepts of MQSeries Workflow, see the *MQSeries Workflow: Concepts and Architecture* manual, and for information about planning to use it on OS/390, see the *MQSeries Workflow for OS/390: Customization and Administration* manual. For information about the Workload Manager, see the *MVS Planning: Workload Management* manual.

Making MQSeries available

Installing MQSeries consists of three stages: receiving, applying, and accepting. See the *MQSeries for OS/390 Program Directory* for instructions on how to do this.

After installation, there are five main tasks to perform to make MQSeries available to application programmers developing MQSeries applications and to applications in production:

- Customizing MQSeries and its adapters
- Operating MQSeries
- Administering MQSeries

- Tuning MQSeries
- Preparing for recovery

Customizing MQSeries and its adapters

When you have installed MQSeries, you must customize it to suit the requirements of your installation. To find out more about customization, read the following chapters:

- Chapter 2, “Preparing for customization” on page 33
- Chapter 3, “Customizing the queue managers” on page 37
- Chapter 5, “Testing your queue manager” on page 99

If you are migrating from a previous version of MQSeries, read Chapter 4, “Migrating from previous versions of MQSeries for MVS/ESA” on page 93. This chapter discusses which customization tasks you need to perform again.

Operating MQSeries

Operating MQSeries involves starting and stopping MQSeries, connecting and disconnecting CICS and IMS regions, using MQSeries commands to manage message queues, and using the MQSeries utilities.

You can find details of these and related tasks in Part 5, “Operating and administering MQSeries” on page 187. Some of these tasks overlap with those of the administrator, so you might also find this Chapter 13, “Writing programs to administer MQSeries” on page 211 useful.

The commands are described in the *MQSeries Command Reference* manual. For details of the MQSeries utilities, see Chapter 14, “Using the MQSeries utilities” on page 227.

Administering MQSeries

MQSeries administration can be performed by a system administrator, system programmer, or computer operator. You can find details of these tasks in Chapter 13, “Writing programs to administer MQSeries” on page 211. The person who undertakes this task has to define and manage resources, and be responsible for managing security and monitoring performance.

Tuning MQSeries

The tuning of MQSeries can be performed by a system programmer or capacity planner. The person who undertakes this task must be able to monitor the operation of MQSeries, and then use the statistics produced to improve its performance.

You can find details of these tasks in Chapter 22, “Monitoring performance and resource usage” on page 363. For information about dealing with performance problems, see the *MQSeries for OS/390 Problem Determination Guide*.

Preparing for recovery

Preparing for recovery underlies all other tasks, and is performed by system programmers. To understand the background to MQSeries recovery, read these chapters:

- Chapter 16, “Understanding termination and restart” on page 279
- Chapter 17, “Understanding the log and the bootstrap data set” on page 299

If you plan the data that must be recovered after an error and how the recovery can be done, you can handle any type of error efficiently. For details on preparing recovery plans and help on recovering from specific failures, see these chapters:

- Chapter 18, “Planning for backup and recovery” on page 307
- Chapter 21, “Example recovery scenarios” on page 337

CSA storage requirement

Each MQSeries for OS/390 subsystem (including the channel initiator) has the following storage requirements:

CSA 4 KB
ECSA 5 MB

In addition, each concurrent MQSeries task requires about 1500 bytes of ECSA. When a task ends, this storage can be reused by other MQSeries tasks. MQSeries does not release the storage until the queue manager is shut down, so the maximum amount of ECSA required can be calculated by multiplying the maximum number of concurrent tasks by 1500 bytes.

Concurrent tasks consist of the following:

- The number of Batch, TSO, or IMS regions that have connected to MQSeries, but not disconnected
- The number of CICS transactions that have issued an MQSeries request, but have not terminated

The trace table also resides in the ECSA; you should use the TRACTBL parameter of the CSQ6SYSP macro to determine the size of the resident trace table. This macro is described in “Using CSQ6SYSP” on page 68.

Private region storage usage

Every channel uses the following private region virtual storage below the 16 MB line in the channel initiator address space:

- 1200 bytes if LE/370 APAR PQ03507 has been applied
- None if LE/370 APAR PQ06157 has also been applied
- 8 KB otherwise

Every channel uses approximately 160 KB of extended private region in the channel initiator address space. Storage is increased if messages larger than 32 KB are transmitted. This increased storage is freed when:

- A sending or client channel requires less than half the current buffer size for 10 consecutive sends
- A heartbeat is sent or received

The storage is freed for re-use within the LE environment but is not seen as free by the OS/390 virtual storage manager.

This means that the upper limit for the number of channels is dependent on message size and arrival patterns as well as individual user system limitations on extended private region size. The limit is likely to be approximately 9000 channels on many systems.

Part 2. Customizing MQSeries after installation

Chapter 2. Preparing for customization	33
Installable features	33
Libraries that exist after installation	34
Chapter 3. Customizing the queue managers	37
Introducing the customization tasks	37
Task 1: Choose which language you want to use	39
Task 2: Choose the distributed queuing facility	40
Task 3: Define the MQSeries subsystem to OS/390	41
Updating the subsystem name table	41
Using command prefix strings	42
Running in a sysplex environment	44
Reviewing the number of system LXs	45
Task 4: Update the OS/390 link list	46
Task 5: APF authorize the MQSeries load libraries	47
Task 6: Update the OS/390 program properties table	48
Task 7: Create procedures for the MQSeries subsystem	49
Task 8: Create procedures for the channel initiator	50
Task 9: Implement your ESM security controls	51
Task 10: Customize the initialization input data sets	52
Initialization commands	52
Initialization data set formats	53
Initialization data set samples	54
Using the CSQINP1 sample	54
Using the CSQINP2 samples	55
Using the other samples	59
Task 11: Create the bootstrap and log data sets	61
Task 12: Define your page sets	62
Calculating the storage requirement for messages	63
Enabling dynamic page set expansion	65
Task 13: Tailor your system parameter module	67
Creating your own system parameter module	67
Fine tuning a system parameter module	68
Using CSQ6SYSP	68
Using CSQ6LOGP	74
Using CSQ6ARVP	76
Task 14: Tailor the channel initiator parameter module	82
Creating your own channel initiator parameter module	82
Using CSQ6CHIP	83
Task 15: Set up Batch, TSO, and RRS adapters	87
Task 16: Set up the operations and control panels	88
Setting up the libraries	88
Updating the ISPF menu	89
Updating the function keys and command settings	89
Task 17: Include the MQSeries dump formatting member	90
Task 18: Suppress information messages	91
Chapter 4. Migrating from previous versions of MQSeries for MVS/ESA	93
Migrating from Version 1.2 to Version 2.1	93
Migrating from Version 1.1.4 to Version 2.1	96

Migrating from Version 1.1.3 to Version 2.1	96
Migrating from Version 1.1.2 or earlier to Version 2.1	97
Coexistence with earlier versions of MQSeries for MVS/ESA	98
Multiple queue manager versions	98
Operations and control panels	98
Application stubs	98
Chapter 5. Testing your queue manager	99
Basic function testing	99
Running the installation verification program CSQ4IVP1	99
Checking the results of CSQ4IVP1	101
Testing for C, C++, COBOL, PL/I, and CICS	102
Testing for distributed queuing	102
Overview of CSQ4IVPX job	102
Preparing to run CSQ4IVPX	102
Running CSQ4IVPX	104
Checking the results of CSQ4IVPX	104

Chapter 2. Preparing for customization

The *MQSeries for OS/390 Program Directory* lists the contents of the MQSeries installation tape, the program and service level information for MQSeries, and describes how to install MQSeries under OS/390 using the System Modification Program Extended (SMP/E).

When you have installed MQSeries, you must carry out a number of tasks before you can make it available to users. Refer to the following chapters for a description of these tasks:

- Chapter 3, “Customizing the queue managers” on page 37
- Chapter 5, “Testing your queue manager” on page 99
- Part 8, “Security” on page 391

If you have migrated from a previous version of MQSeries for OS/390, you don't need to perform most of the customization tasks. Refer to Chapter 4, “Migrating from previous versions of MQSeries for MVS/ESA” on page 93 for information about the tasks you have to perform.

Installable features

MQSeries for OS/390 comprises the following features:

Base

This is required; it comprises all the main functions, and includes the MQSeries-CICS bridge and the distributed queuing facility (supporting both TCP/IP and APPC communications).

Note: This distributed queuing facility is known as the “non-CICS mover” because you do not need to have CICS installed to use it.

National language features

These contain error messages and panels in all the supported national languages. Each language has a language letter associated with it. The languages and letters are:

- C** Simplified Chinese
- E** U.S. English (mixed case)
- K** Japanese
- U** U.S. English (uppercase)

You must install at least one of these (you can install more than one).

Distributed queuing with CICS feature

This is optional; it is required only if you are using CICS ISC for distributed queuing.

Note: This feature is known as the “CICS mover.”

Client attachment feature

This is optional; it is only required if you are going to attach clients to your MQSeries for OS/390 subsystem. When you have installed this feature, there are no configuration parameters to set before you can attach clients to MQSeries for OS/390. Administration for clients is available even if you don't install this feature.

Internet Gateway feature

This is optional; it is only required if you want to use the MQSeries Internet Gateway. This is described in the *MQSeries Internet Gateway User's Guide*. (This online book is supplied with the Internet Gateway.)

Libraries that exist after installation

MQSeries is supplied with a number of separate load libraries. Table 3 shows the libraries that might exist after you have installed MQSeries.

thlqual

Throughout this book, the default value thlqual is used to indicate the target library high-level qualifier for MQSeries data sets in your installation.

For more information, see the *MQSeries for OS/390 Program Directory*.

<i>Table 3 (Page 1 of 3). MQSeries libraries that exist after installation</i>	
Name	Description
thlqual.SCSQANLC	Contains the load modules for the Simplified Chinese version of MQSeries.
thlqual.SCSQANLE	Contains the load modules for the U.S. English (mixed case) version of MQSeries.
thlqual.SCSQANLK	Contains the load modules for the Japanese version of MQSeries.
thlqual.SCSQANLU	Contains the load modules for the U.S. English (uppercase) version of MQSeries.
thlqual.SCSQASMS	Contains source for assembler sample programs.
thlqual.SCSQAUTH	The main repository for all MQSeries product load modules; it also contains the default parameter modules, CSQZPARM and CSQXPARM. This library must be APF-authorized.
thlqual.SCSQCICS	Contains the load modules that must be included in the CICS DFHRPL concatenation. These are separated from the main MQSeries load library so that the number of modules in the concatenation search is kept to a minimum to improve performance and to avoid the need for APF authorization.
thlqual.SCSQCLST	Contains CLISTs used by the mail manager sample program.
thlqual.SCSQCOBC	Contains COBOL copybooks, including copybooks required for the sample programs.
thlqual.SCSQCOBS	Contains source for COBOL sample programs.
thlqual.SCSQCPPS	Contains C source for C++.
thlqual.SCSQC37S	Contains source for C/370™ sample programs.
thlqual.SCSQC370	Contains C/370 headers, including headers required for the sample programs.
thlqual.SCSQDEFS	Contains side definitions for C++.
thlqual.SCSQEXEC	Contains REXX execs to be included in the SYSEXEC or SYSPROC concatenation if you are using the MQSeries operations and control panels.

Table 3 (Page 2 of 3). MQSeries libraries that exist after installation

Name	Description
thlqual.SCSQHPPS	Contains header files for C++.
thlqual.SCSQINST	Contains JCL for installation jobs.
thlqual.SCSQLINK	Early code library. Contains the load modules that must be in the link list because they are loaded at system initial program load (IPL). The library must be APF-authorized and must be in the link list.
thlqual.SCSQLOAD	Load library. Contains load modules for non-APF code, user exits, utilities, samples, installation verification programs, and adapter stubs. The library does not need to be APF-authorized and does not need to be in the link list.
thlqual.SCSQMACS	Contains Assembler macros including: sample macros, product macros, and system parameter macros.
thlqual.SCSQMAPS	Contains CICS mapsets used by sample programs.
thlqual.SCSQMSGC	Contains ISPF messages to be included in the ISPMLIB concatenation if you are using the Simplified Chinese language feature for the MQSeries operations and control panels.
thlqual.SCSQMSGE	Contains ISPF messages to be included in the ISPMLIB concatenation if you are using the U.S. English (mixed case) language feature for the MQSeries operations and control panels.
thlqual.SCSQMSGK	Contains ISPF messages to be included in the ISPMLIB concatenation if you are using the Japanese language feature for the MQSeries operations and control panels.
thlqual.SCSQMSGU	Contains ISPF messages to be included in the ISPMLIB concatenation if you are using the U.S. English (uppercase) language feature for the MQSeries operations and control panels.
thlqual.SCSQMVR1	Contains the load modules for distributed queuing when using TCP/IP with the OpenEdition sockets or IUCV interface, or LU 6.2. This library must be APF-authorized.
thlqual.SCSQMVR2	Contains the load modules for distributed queuing when using TCP/IP with the TCPAccess interface, or LU 6.2. This library must be APF-authorized.
thlqual.SCSQPLIC	Contains PL/I headers.
thlqual.SCSQPLIS	Contains source for PL/I sample programs.
thlqual.SCSQPnLA	Contains IPCS panels, for the dump formatter, to be included in the ISPPLIB concatenation. Also contains panels for MQSeries sample programs.
thlqual.SCSQPnLC	Contains ISPF panels to be included in the ISPPLIB concatenation if you are using the Simplified Chinese language feature for the MQSeries operations and control panels.
thlqual.SCSQPnLE	Contains ISPF panels to be included in the ISPPLIB concatenation if you are using the U.S. English (mixed case) language feature for the MQSeries operations and control panels.
thlqual.SCSQPnLK	Contains ISPF panels to be included in the ISPPLIB concatenation if you are using the Japanese language feature for the MQSeries operations and control panels.

<i>Table 3 (Page 3 of 3). MQSeries libraries that exist after installation</i>	
Name	Description
thlqual.SCSQPNU	Contains ISPF panels to be included in the ISPLIB concatenation if you are using the U.S. English (uppercase) language feature for the MQSeries operations and control panels.
thlqual.SCSQPROC	Contains sample JCL and default system initialization data sets.
thlqual.SCSQSKL	Contains ISPF skeletons to be included in the ISPLIB concatenation if you are using the MQSeries operations and control panels.
thlqual.SCSQSNLC	Contains the load modules for the Simplified Chinese versions of the MQSeries modules that are required for special purpose function (for example the early code).
thlqual.SCSQSNLE	Contains the load modules for the U.S. English (mixed case) versions of the modules that are required for special purpose function (for example the early code).
thlqual.SCSQSNLK	Contains the load modules for the Japanese versions of the MQSeries modules that are required for special purpose function (for example the early code).
thlqual.SCSQSNLU	Contains the load modules for the U.S. English (uppercase) versions of the MQSeries modules that are required for special purpose function (for example the early code).
thlqual.SCSQTBLC	Contains ISPF tables to be included in the ISPTLIB concatenation if you are using the Simplified Chinese language feature for the MQSeries operations and control panels.
thlqual.SCSQTBLE	Contains ISPF tables to be included in the ISPTLIB concatenation if you are using the U.S. English (mixed case) language feature for the MQSeries operations and control panels.
thlqual.SCSQTBLK	Contains ISPF tables to be included in the ISPTLIB concatenation if you are using the Japanese language feature for the MQSeries operations and control panels.
thlqual.SCSQTBLU	Contains ISPF tables to be included in the ISPTLIB concatenation if you are using the U.S. English (uppercase) language feature for the MQSeries operations and control panels.
Attention: Do <i>not</i> modify or customize any of these libraries. If you want to make changes, copy the libraries, and make your changes to the copies.	

Chapter 3. Customizing the queue managers

This chapter leads you through the various stages of customizing MQSeries after you have successfully installed it. The installation process is described in the *MQSeries for OS/390 Program Directory*.

If you are migrating from a previous version of MQSeries for OS/390, you might not need to perform these tasks. See Chapter 4, "Migrating from previous versions of MQSeries for MVS/ESA" on page 93 for information about what you need to do.

Note: The sample data set members supplied with MQSeries have names beginning with the four characters CSQ4 and are in the library thlqual.SCSQPROC.

Introducing the customization tasks

To customize MQSeries for your installation, you must perform the tasks listed here. However, before you begin customization, read this notice:

Read this notice before you do anything

For each task you must consider:

1. **Whether the task must be repeated for each MQSeries subsystem.**

Some of these tasks you need only do once, regardless of the number of MQSeries subsystems, while others must be repeated for each MQSeries subsystem. Each task description tells you which category that task belongs to.

2. **Whether the task requires an IPL.**

Some tasks might only take effect following an OS/390 system initial program load (IPL). For example, an IPL might be required by:

- Any task that changes certain OS/390 system parameters
- Task 7, when you change certain tables used by an external security manager, such as RACF

Therefore, make sure you have completed **all** the necessary tasks before you IPL the system. Each task description tells you whether an IPL is required. In general, an IPL is needed when you install and customize MQSeries, but not when you add a new MQSeries subsystem.

If you already have a previous version of MQSeries for OS/390, read Chapter 4, "Migrating from previous versions of MQSeries for MVS/ESA" on page 93 first. Except for the changes noted in Chapter 4, the installation and customization tasks described in Chapter 3 are unnecessary.

Customizing queue managers

Work through the following tasks, checking each one off as you complete it:

- Task 1: Choose which language you want to use
- Task 2: Choose the distributed queuing facility
- Task 3: Define the MQSeries subsystem to OS/390
- Task 4: Update the OS/390 link list
- Task 5: APF authorize the MQSeries load libraries
- Task 6: Update the OS/390 program properties table
- Task 7: Create procedures for the MQSeries subsystem
- Task 8: Create procedures for the channel initiator
- Task 9: Implement your ESM security controls
- Task 10: Customize the initialization input data sets
- Task 11: Create the bootstrap and log data sets
- Task 12: Define your page sets
- Task 13: Tailor your system parameter module
- Task 14: Tailor the channel initiator parameter module
- Task 15: Set up Batch, TSO, and RRS adapters
- Task 16: Set up the operations and control panels
- Task 17: Include the MQSeries dump formatting member
- Task 18: Suppress information messages

Tasks 3 through 6 involve updating the OS/390 system parameters. You need to know which ones were specified when the system IPL was performed. SYS1.PARMLIB(IEASYSpp) contains a list of parameters that point to other members of SYS1.PARMLIB (where pp represents the OS/390 system parameter list that was used to IPL the system).

The entries you need to find are:

For Task 3:

SSN=ss Points to the defined subsystem list (member IEFSSNss)

NSYSLX=nn

The number of linkage indexes reserved for system LXs (member IEASYSxx)

For Task 4:

LNK=kk Points to the link list (member LNKLSTkk)

For Task 5:

PROG=xx or APF=aa

Points to the Authorized Program Facility (APF) authorized library list (member PROGxx or IEFAPFaa)

For Task 6:

SCH=xx Points to the Program Properties Table (PPT) (member SCHEDxx)

When you have completed all the customization tasks required, refer to Part 3, "MQSeries and CICS" on page 107 if you are using CICS, and Part 4, "MQSeries and IMS" on page 157 if you are using IMS.

Task 1: Choose which language you want to use

- Repeat this task for each MQSeries subsystem.

You can choose one of the following national languages for the MQSeries operator messages and the MQSeries operations and control panels (including the character sets used). Each language is identified by a language letter:

- C** Simplified Chinese
- E** U.S. English (mixed case)
- K** Japanese
- U** U.S. English (uppercase)

The samples, MQSeries commands, and utility control statements are available only in mixed case U.S. English.

You must specify the appropriate libraries in the JCL that you will use for running MQSeries (as described in the following sections). Table 4 shows the names of the libraries for the language features; the language letter is the last letter of the library names.

Description	Japanese	Simplified Chinese	U.S. English (mixed case)	U.S. English (uppercase)
Load modules	thlqual.SCSQANLK	thlqual.SCSQANLC	thlqual.SCSQANLE	thlqual.SCSQANLU
ISPF messages	thlqual.SCSQMSGK	thlqual.SCSQMSGC	thlqual.SCSQMSGE	thlqual.SCSQMSGU
ISPF panels	thlqual.SCSQPNLK	thlqual.SCSQPNLC	thlqual.SCSQPNLE	thlqual.SCSQPNLU
Special purpose function (for example, early code)	thlqual.SCSQSNLK	thlqual.SCSQSNLC	thlqual.SCSQSNLE	thlqual.SCSQSNLU
ISPF tables	thlqual.SCSQTBLK	thlqual.SCSQTBLC	thlqual.SCSQTBLE	thlqual.SCSQTBLU

More details are given in the following sections.

Task 2: Choose the distributed queuing facility

- Repeat this task for each MQSeries subsystem.

The distributed queuing facility provided with the base product feature of MQSeries uses native OS/390 communications (APPC or TCP/IP). It can either use APPC (LU 6.2), TCP/IP from IBM, or TCPAccess from Interlink Computer Sciences, Inc. This facility is also known as the *non-CICS mover* because it does not use CICS intersystem communication (ISC) for communications, and so you do not need to have CICS installed to use it. (However, even if you do use CICS, you can still use this mover.) You are recommended to use the non-CICS mover.

Alternatively, you can use CICS ISC for distributed queuing; this facility is also known as the *CICS mover*. You must have installed the CICS mover feature to use this.

You can enable both facilities and use them simultaneously on the same MQSeries instance. However, the two types will have no knowledge of each other or each other's channels, and you must ensure that the channel names they use are distinct.

If you want to use clustering, you must use the non-CICS mover. See the *MQSeries Queue Manager Clusters* manual for information about using clusters. If you want to use clients, you must also use the non-CICS mover; the client attachment feature is needed as well (but you can administer clients without it).

Whichever mover you choose, you must perform the following three tasks to enable distributed queuing:

- Customize the distributed queuing facility and define the MQSeries objects required.
- Define access security; this is described in "Security considerations for distributed queuing" on page 453.
- Set up your communications; this is described in the *MQSeries Intercommunication* manual. Amongst other things, you must set up your TCP/IP.DATA data set if you are using TCP/IP, LU names and side information if you are using APPC, and CICS definitions if you are using CICS.

If you choose the non-CICS mover, customization instructions are described in this chapter. You also need to choose which communications interface to use. This can be:

- APPC (LU 6.2)
- IBM TCP/IP IUCV (OS/390 Version 2.4 only)
- IBM TCP/IP OpenEdition sockets
- TCPAccess (native)
- TCPAccess IUCV
- TCPAccess OpenEdition sockets

Note: OpenEdition is also known as UNIX System Services.

If you choose the CICS mover, customization instructions are described in Appendix D, "Enabling distributed queuing using CICS ISC" on page 479.

Task 3: Define the MQSeries subsystem to OS/390

- Repeat this task for each MQSeries subsystem.
- You might need to IPL the system before these changes take effect.

Updating the subsystem name table

The subsystem name table of OS/390, which is taken initially from the SYS1.PARMLIB member IEFSSNss, contains the definitions of formally defined OS/390 subsystems. To define each MQSeries subsystem, you must add an entry to this table, either by changing the IEFSSNss member of SYS1.PARMLIB, or by using the SETSSI OS/390 command if it is available.

If you use the SETSSI command, the change takes effect immediately, otherwise you must IPL your system. Even if you use the SETSSI OS/390 command so that changes take effect immediately, you should add the entries to the IEFSSNss member of SYS1.PARMLIB as well, so that they will remain in effect after subsequent IPLs.

The format of a definition for an MQSeries subsystem in IEFSSNss is:

```
ssid,CSQ3INI,'CSQ3EPX,cpf,scope'
```

and the corresponding SETSSI command is:

```
SETSSI ADD,S=ssid,I=CSQ3INI,P='CSQ3EPX,cpf,scope'
```

where:

ssid The subsystem identifier.

cpf The command prefix string (see “Using command prefix strings” on page 42 for information about the rules for defining CPFs).

scope The system scope, used if you are running in an OS/390 sysplex (see “Defining the scope for sysplex operation” on page 44). If you are not running in a sysplex, use M for this value.

Figure 5 shows several examples.

```
CSQ1,CSQ3INI,'CSQ3EPX,+mqs1cpf,M'
CSQ2,CSQ3INI,'CSQ3EPX,+mqs2cpf,M'
CSQ3,CSQ3INI,'CSQ3EPX,+,M'
```

Figure 5. Sample IEFSSNss statements for defining subsystems

Note: Once you have created objects in a subsystem, you cannot change the subsystem name or use the page sets from one subsystem in another subsystem. To do either of these, you must unload all the objects and messages from one subsystem and reload them into another.

Table 5 gives a number of examples showing the associations of subsystem names and CPFs, as defined by the statements in Figure 5.

Define the subsystem

MQSeries subsystem name	CPF
CSQ1	+mqs1cpf
CSQ2	+mqs2cpf
CSQ3	++

Note: The ACTIVATE and DEACTIVATE functions of the SETSSI OS/390 command are not supported by MQSeries.

Using command prefix strings

Each instance of MQSeries that you install must have its own *command prefix* string (CPF). You use the CPF to identify the OS/390 subsystem that commands are intended for. It also identifies the OS/390 subsystem from which messages sent to the console originate.

You can issue all MQSeries commands from an authorized console by inserting the CPF before the command. If you enter commands via the system command input queue (for example, using CSQUTIL), or use the MQSeries operations and control panels, you do not use the CPF.

To start a subsystem called CSQ1 whose CPF is '+cpf', issue the command +cpf START QMGR from the operator console (the space between the CPF and the command is optional).

The CPF also identifies the subsystem that is returning operator messages. The following example shows +cpf as the CPF between the message number and the message text.

```
CSQ9022I +cpf CSQNCDSR ' DISPLAY CMDSERV' NORMAL COMPLETION
```

Defining command prefix strings

You should adopt a system-wide convention for your CPFs for all subsystems to avoid conflicts. You should adhere to the following guidelines:

- Define a CPF as a one- to eight-character string.
- Do not use a CPF that is already in use by any other subsystem, and avoid using the JES backspace character defined on your system as the first character of your string.
- Define your CPF using characters from the set of valid characters listed in Table 7 on page 43.
- Do not use a CPF that is an abbreviation for an already defined process or that might be confused with command syntax. For example, a CPF such as 'D' conflicts with OS/390 commands such as DISPLAY. To avoid this happening, you should use one of the special characters (shown in Table 7 on page 43) as the first or only character in your CPF string.

- Do not define a CPF that is either a subset or a superset of an existing CPF.
For an example, see Table 6:

Table 6. Example of CPF subset and superset rules

Subsystem name	CPF defined	Commands routed to...
MQA	!A	MQA
MQB	!B	MQB
MQC1	!C1	MQC1
MQC2	!C2	MQC2
MQB1	!B1	MQB

Commands intended for subsystem MQB1 (using CPF !B1) are routed to subsystem MQB because the CPF for this subsystem is !B, a subset of !B1. For example, if you entered the command !B1 START QMGR, subsystem MQB will receive the command 1 START QMGR (which, in this case, it will be unable to deal with).

You can see which prefixes already exist by issuing the OS/390 command DISPLAY OPDATA.

If you are running in a sysplex, OS/390 will diagnose any conflicts of this type at the time of CPF registration (see "Running in a sysplex environment" on page 44 for information about CPF registration).

Table 7 shows the characters that you can use when defining your command prefix (CPF) strings:

Table 7 (Page 1 of 2). Valid character set for CPF strings

Character set	Contents	
Alphanumeric	Alphabetic Numeric	Uppercase A through Z, lowercase a through z, 0 through 9
National (see note)	"At" sign Dollar sign Pound sign	@ \$ # (Characters that can be represented as hexadecimal values X'7C', X'5B', and X'7B')

Define the subsystem

Character set	Contents	
Special	period	.
	slash	/
	left parenthesis	(
	right parenthesis)
	asterisk	*
	ampersand	&
	plus sign	+
	hyphen	-
	equal sign	=
	cent sign	¢
	less than sign	<
	vertical bar	
	exclamation point	!
	semi-colon	;
	percent sign	%
	underscore	_
	question mark	?
	colon	:

Note: The system recognizes the following hexadecimal representations of the national characters: @ as X'7C', \$ as X'5B', and # as X'7B'. In countries other than the U.S., the U.S. national characters represented on terminal keyboards might generate a different hexadecimal representation and cause an error. For example, in some countries the \$ character might generate an X'4A'.

Running in a sysplex environment

If you are in a sysplex environment, MQSeries registers your CPFs to enable you to enter a command from any console in the sysplex and route that command to the appropriate system for execution. The command responses are returned to the originating console.

Defining the scope for sysplex operation

Scope is used to determine the type of CPF registration performed by the MQSeries subsystem when you are running MQSeries in a sysplex environment.

Possible values for scope are as follows:

M System scope.

The CPF is registered with OS/390 at system IPL time by MQSeries and remains registered for the entire time that the OS/390 system is active.

MQSeries operator commands must be entered at a console connected to the OS/390 image running the target subsystem, or you must use ROUTE commands to direct the command to that image.

You should use this option if you are not running in a sysplex.

X Sysplex IPL scope.

The CPF is registered with OS/390 at system IPL time by MQSeries and remains registered for the entire time that the OS/390 system is active.

MQSeries operator commands can be entered at any console connected to the sysplex, and are routed to the image that is executing the target system automatically.

S Sysplex started scope.

The CPF is registered with OS/390 at the time the MQSeries subsystem is started and remains active until the MQSeries subsystem terminates.

You must use ROUTE commands to direct the original START MQSeries command to the target system, but all further MQSeries operator commands can be entered at any console connected to the sysplex, and are routed to the target system automatically.

After MQSeries termination, you must use the ROUTE commands to direct subsequent START commands to the target MQSeries subsystem.

An MQSeries subsystem with a CPF with scope of X can only be defined on one OS/390 image within a sysplex. If you use this option, you must define a unique subsystem name table for each OS/390 image requiring MQSeries subsystems with CPFs of scope X.

An MQSeries subsystem with a CPF with scope of S can be defined on one or more OS/390 images within a sysplex, so these images can share a single subsystem name table. However you must ensure that the initial START command is issued on (or routed to) the OS/390 image on which you want the MQSeries subsystem to run. If you use this option, you can stop the MQSeries subsystem and restart it on a different OS/390 image within the sysplex without having to change the subsystem name table or re-IPL an OS/390 system.

If you want to use ARM to restart queue managers in different OS/390 images automatically, every queue manager must be defined in each OS/390 image on which that queue manager might be restarted, with a sysplex-wide, unique 4-character subsystem name with a CPF scope of S. See “Using the OS/390 Automatic Restart Manager (ARM)” on page 294 for more information about automatic restart.

Reviewing the number of system LXs

Each MQSeries subsystem defined in the subsystem name table reserves one system linkage index at IPL time. This system linkage index is reused if the MQSeries subsystem is stopped and restarted. The NSYSLX parameter in IEASYSxx defines the number of linkage indexes (in addition to those in the system function table) to be reserved as system linkages. The default number is 55.

If your environment has a number of subsystems defined that use system linkage indexes (for example, DB2, IRLM, and IMS V5), you might need to increase the value of NSYSLX when you define MQSeries subsystems. Each MQSeries subsystem reserves one system linkage index, and each instance of the distributed queuing feature reserves one non-system linkage index.

You must IPL your system before changes to NSYSLX take effect.

Task 4: Update the OS/390 link list

- *You need only perform this task once.*
- *You must IPL the system before these changes take effect.*

You must add the MQSeries early code library, thlqual.SCSQLINK, to the link list, SYS1.PARMLIB(LNKLSTkk), and put thlqual.SCSQLINK in the master catalog.

If you want to minimize the number of libraries in the link list, copy the load modules from thlqual.SCSQLINK into an existing library that is in the link list and in the master catalog. The library you copy the members into must also be APF-authorized. However, if you do this, the installation program (SMP/E) cannot apply service to these modules, so you must recopy the load modules if service is to be applied to them.

You also need the associated early error message module, CSQ3ECMX. Either add the library containing the language you want to the link list, or copy this module from that library to an existing library in the link list. The libraries are called thlqual.SCSQSNLx, where x is the language letter.

The distributed queuing facility, CICS bridge, and Internet Gateway need access to the LE run-time library SCEERUN. You might want to include it in the link list. (This is not required for the CICS mover.)

Task 5: APF authorize the MQSeries load libraries

- *You need only perform this task once.*
- *You might need to IPL the system before these changes take effect.*

The MQSeries load libraries thlqual.SCSQAUTH and thlqual.SCSQLINK must be APF-authorized. You must also APF-authorize the libraries for your national language feature (thlqual.SCSQANLx and thlqual.SCSQSNLx) and for the non-CICS mover (thlqual.SCSQMVR1 or thlqual.SCSQMVR2).

All members of the link list are APF-authorized if the SYS1.PARMLIB member IEASYSpp contains the statement:

```
LNKAUTH=LNKLST
```

LNKAUTH=LNKLST is the default if LNKAUTH is not specified.

Because thlqual.SCSQLINK must be included in the link list, if IEASYSpp contains this LNKAUTH statement or if you allow it to default, you do not need to put thlqual.SCSQLINK in the APF list as well.

Note: You must APF-authorize all the libraries that you include in the MQSeries STEPLIB. If you put a library that is not APF-authorized in the STEPLIB, the whole library concatenation loses its APF authorization.

The APF lists are in the SYS1.PARMLIB member PROGxx or IEAAPFaa. The lists contain the names of APF authorized OS/390 libraries. The order of the entries in the lists is not significant. See the *MVS Initialization and Tuning Reference* manual for information about APF lists.

If you use PROGxx members with dynamic format, you need only issue the SET PROG= OS/390 command for the changes to take effect. Otherwise, if you use static format or IEAAPFaa members, you must IPL your system.

Task 6: Update the OS/390 program properties table

- *You need only perform this task once.*
- *You must IPL the system before these changes take effect.*

You must add the following entry to the program properties table (PPT) which you can find in SYS1.PARMLIB(SCHEDxx).

```
PPT      PGMNAME(CSQYASCP) /* CSQ - THIS IS REQUIRED FOR MQSERIES */
          CANCEL           /* CAN BE CANCELED */
          KEY(7)           /* STORAGE PROTECTION KEY */
          SWAP             /* PROGRAM IS SWAPPABLE */
          NOPRIV           /* NOT PRIVILEGED */
          DSI              /* REQUIRES DATA SET INTEGRITY */
          PASS             /* NOT ALLOWED TO BYPASS PASS PROT */
          SYST             /* SYSTEM TASK SO NOT TIMED */
          AFF(NONE)        /* NO PROCESSOR AFFINITY */
          NOPREF           /* NO PREFERRED STORAGE FRAMES */
```

Figure 6. PPT additional entries needed for MQSeries

Task 7: Create procedures for the MQSeries subsystem

- *Repeat this task for each MQSeries subsystem.*

For each MQSeries subsystem defined in the subsystem name table, create a cataloged procedure in a procedure library. The IBM-supplied procedure library is called SYS1.PROCLIB, but your installation might use its own naming convention.

The name of the MQSeries started task procedure is formed by concatenating the subsystem name with the characters MSTR. For example, subsystem CSQ1 has the procedure name CSQ1MSTR. You need one procedure for each of the subsystems you define.

We recommend that a subsystem called CSQ1MSTR is created initially for installation verification and testing purposes.

Copy the sample started task procedure thlqual.SCSQPROC(CSQ4MSTR) to member CSQ1MSTR (or a name of your choice) of your SYS1.PROCLIB or, if you are not using SYS1.PROCLIB, your procedure library. Copy CSQ4MSTR to a member in your procedure library for each of the MQSeries subsystems that you define.

When you have copied the members, you can tailor them to the requirements of each subsystem, using the instructions in member CSQ4MSTR. You can also use symbolic parameters in the JCL to allow the procedure to be modified when it is started. See “Start options” on page 194 for an example of this.

You must concatenate thlqual.SCSQANLx (where x is the language letter for your national language) before thlqual.SCSQAUTH in the STEPLIB DD statement.

Before you start MQSeries, you should set up MQSeries data set and system security by:

- Authorizing the queue manager started task procedure to run under your external security manager.
- Authorizing access to the queue manager data sets.

For details about how to do this, see “Security installation tasks” on page 456.

The exit library (CSQXLIB) can be added to this procedure later if you want to use queue manager exits. In this case, you also need access to the LE run-time library SCEERUN; if it is not in your link list (SYS1.PARMLIB(LNKLSTkk)), concatenate it in the STEPLIB DD statement. You need to stop and restart your queue manager to do this.

Task 8: Create procedures for the channel initiator

- Repeat this task for each MQSeries subsystem.
- Omit this task if you are using the CICS mover.

You need to create a channel-initiator started task procedure for each MQSeries subsystem that is going to use distributed queuing. To do this, you need to:

1. Copy the sample started task procedure thlqual.SCSQPROC(CSQ4CHIN) to your procedure library. Name the procedure xxxxCHIN, where xxxx is the name of your MQSeries subsystem (for example, CSQ1CHIN would be the channel initiator started task procedure for queue manager CSQ1).
2. Make a copy for each MQSeries subsystem that you are going to use.
3. Tailor the procedures to your requirements using the instructions in the sample procedure CSQ4CHIN. You can also use symbolic parameters in the JCL to allow the procedure to be modified when it is started. See “Start options” on page 194 for an example of this.

Concatenate the library containing your national language feature (thlqual.SCSQANLx where x is the letter for your language) before thlqual.SCSQAUTH in the STEPLIB DD statement.

Choose the appropriate distributed queuing library: thlqual.SCSQMVR1 if you are using the OpenEdition sockets or IUCV TCP/IP interface, or thlqual.SCSQMVR2 if you are using the TCPaccess interface. For LU 6.2 you can use either library.

Access to the LE run-time library SCEERUN is required; if it is not in your link list (SYS1.PARMLIB(LNKLSTkk)), concatenate it in the STEPLIB DD statement.

4. Authorize the procedures to run under your external security manager.

The exit library (CSQXLIB) can be added to this procedure later if you want to use channel exits. You will need to stop and restart your channel initiator to do this.

If you are using TCP/IP, the channel initiator address space must be able to access the TCPIP.DATA data set that contains TCP/IP system parameters. There are various ways that this has to be set up, depending on which TCP/IP product and interface you are using. They include:

- Environment variable, RESOLVER_CONFIG
- HFS file, /etc/resolv.conf
- //SYSTCPD DD statement
- //SYSTCPDD DD statement
- jobname/userid.TCPIP.DATA
- SYS1.TCPPARMS(TCPDATA)
- zapname.TCPIP.DATA

Some of these will affect your started-task procedure JCL. For more information, see the following:

- *TCP/IP OpenEdition: Planning and Release Guide*
- *OS/390 OpenEdition: Planning*
- Your TCPaccess documentation

Task 9: Implement your ESM security controls

- *Repeat this task for each MQSeries subsystem.*
- *You might have to IPL the system before these changes take effect.*

You must now consider how you are going to implement any security controls for MQSeries.

If you use RACF as your external security manager, see Part 8, “Security” on page 391, which describes how to implement these security controls. If you are using RACF, you might need to IPL the system if you change the started-task procedures table, depending on how you choose to associate user IDs with the started tasks.

If you are using the non-CICS mover, you must also do the following tasks:

1. If your subsystem has connection security active, define a connection security profile ssid.CHIN to your external security manager (see “Connection security profiles for distributed queuing” on page 405 for information about this).
2. If you are using a sockets interface, ensure that the user ID under whose authority the channel initiator is running is configured to use OpenEdition, as described in the *OS/390 OpenEdition Planning* manual.

Task 10: Customize the initialization input data sets

- *Repeat this task for each MQSeries subsystem.*

Each MQSeries instance gets its initial definitions from a series of commands contained in the MQSeries *initialization input data sets*. These data sets are referenced by the DDnames CSQINP1 and CSQINP2 defined in the MQSeries subsystem started task procedure.

Responses to these commands are written to the initialization output data sets referenced by the DDnames CSQOUT1 and CSQOUT2.

Initialization commands

Commands in the initialization input data sets are processed when MQSeries is initialized on MQSeries startup. Three types of command can be issued from the initialization input data sets:

- Commands to define MQSeries entities that cannot be recovered. For example:

```
DEFINE BUFFPOOL  
DEFINE PSID
```

Note: This set of DEFINE commands specifically excludes MQSeries objects.

These commands must reside in the data set identified by the DDname CSQINP1. They are processed before the restart phase of initialization. They cannot be issued through the console, operations and control panels, or an application program. The responses to these commands are written to the sequential data set that you refer to in the CSQOUT1 statement of the started task procedure.

- Commands to define MQSeries objects that are recoverable after restart. These objects are listed in “Objects and commands” on page 7. These definitions must be specified in the data set identified by the DDname CSQINP2. They are stored in page set zero. CSQINP2 is processed after the restart phase of initialization. The responses to these commands are written to the sequential data set that you refer to in the CSQOUT2 statement of the started task procedure.
- Commands to manipulate MQSeries objects. These commands must also be specified in the data set identified by the DDname CSQINP2. For example, the MQSeries-supplied data set CSQ4INP2 contains an ALTER QMGR command to specify a dead-letter queue for the subsystem. The response to these commands is written to the CSQOUT2 output data set.

Commands are restricted to a maximum of 32762 characters.

If MQSeries objects are defined in CSQINP2, MQSeries attempts to redefine them each time the MQSeries subsystem is started. If the queues already exist, the attempt to define them fails. You can avoid this problem by using the REPLACE parameter of the DEFINE commands as described in the *MQSeries Command Reference* manual.

Initialization commands for distributed queuing

You can also use the CSQINP2 initialization data set for the START CHINIT command, and follow it with a series of other commands to define your distributed queuing environment (for example, defining your channels). If you stop and restart the channel initiator however, CSQINP2 is not reprocessed, so MQSeries provides a third initialization input data set, called CSQINPX, that you can choose to process as part of the channel initiator started task procedure.

The MQSC commands contained in the data set are executed at the end of channel initiator initialization, and output is written to the data set specified by the CSQOUTX DD statement. The output is similar to that produced by the COMMAND function of the MQSeries utility program (CSQUTIL). See “MQSeries utility program (CSQUTIL)” on page 229 for information about the MQSeries utility program.

You can specify any of the MQSC commands that can be issued from CSQUTIL, not only the channel commands. You can enter commands from other sources while CSQINPX is being processed. All commands are issued in sequence, regardless of the success of the previous command.

To specify a command response time, you can use the pseudo-command COMMAND as the first command in the data set. This takes a single optional keyword RESPTIME(*nnn*), where *nnn* is the time, in seconds, to wait for a response to each of the commands. This is in the range 5 through 999; the default is 30. If MQSeries detects that the responses to four commands have taken too long, processing of CSQINPX is stopped and no further commands are issued. The channel initiator is not stopped, but message CSQU052E is written to the CSQOUTX data set, and message CSQU013E is sent to the console. When MQSeries has completed processing of CSQINPX successfully, message CSQU012I is sent to the console.

Initialization data set formats

The initialization input data sets can be partitioned data set (PDS) members or sequential data sets. They can be a concatenated series of data sets. Define them with a record length of 80 bytes, where:

- Only columns 1 through 72 are significant. Columns 73 through 80 are ignored.
- Records with an asterisk (*) in column 1 are interpreted as comments and are ignored.
- Blank records are ignored.
- Each command must start on a new record.
- A trailing – means continue from column 1 of the next record.
- A trailing + means continue from the first non-blank column of the next record.
- The maximum number of characters permitted in a command is 32762.

If you use a sequential data set for CSQINP1 or CSQINP2, the data set remains allocated to the queue manager started task while the queue manager is active. During this time, it is not available for editing; if you want to change the data set, you must first stop the queue manager. The same applies to CSQINPX for the duration of the channel initiator started task.

Customize initialization data sets

The initialization output data sets are sequential data sets, with a record length of 125, a record format of VBA, and a block size of 629.

Initialization data set samples

The following sample initialization data set members are supplied with MQSeries:

For CSQINP1

```
thlqual.SCSQPROC(CSQ4INP1)
```

For CSQINP2

```
thlqual.SCSQPROC(CSQ4INSG)
thlqual.SCSQPROC(CSQ4INSX)
thlqual.SCSQPROC(CSQ4INYG)
thlqual.SCSQPROC(CSQ4INYD)
thlqual.SCSQPROC(CSQ4INYC)
```

Other

```
thlqual.SCSQPROC(CSQ4DISP)
thlqual.SCSQPROC(CSQ4DISQ)
thlqual.SCSQPROC(CSQ4IVP)
thlqual.SCSQPROC(CSQ4INPX)
```

To preserve the originals, you should make working copies of each sample. Then you can tailor the commands in these working copies to suit your system requirements.

If you intend to define more than one MQSeries subsystem, you are recommended to include the subsystem name in the high-level qualifier of the initialization input data set name. This allows you to identify more easily the MQSeries subsystem associated with each data set.

Using the CSQINP1 sample

The sample CSQINP1 data set thlqual.SCSQPROC(CSQ4INP1) contains definitions of buffer pools, page set to buffer pool associations, MAXSMSGS, and an ALTER SECURITY command. The sample should be included in the CSQINP1 concatenation of your MQSeries started task procedure.

Notes:

1. MQSeries supports up to four buffer pools (0 through 3). The DEFINE BUFFPOOL command can only be issued from a CSQINP1 initialization data set. The definitions in the sample specify four buffer pools.
2. Each page set used by the subsystem must be defined in the CSQINP1 initialization data set by using the DEFINE PSID command. The page set definition associates a buffer pool ID with a page set. If no buffer pool is specified, buffer pool 0 is used by default.

Page set zero (00) must be defined. It contains all the object definitions. You can define up to 100 page sets for each MQSeries subsystem.
3. The DEFINE MAXSMSGS command defines the maximum number of **MQGET** and **MQPUT** calls that can be made within an MQSeries unit of recovery. In CSQ4INP1, the default value for MAXSMSGS is defined as 10 000.

4. The ALTER SECURITY command can be used to alter the security attributes TIMEOUT and INTERVAL. In CSQ4INP1, the default values are defined as 54 and 12 respectively.

See “An overview of storage management” on page 15 for information about organizing buffer pools and page sets.

Using the CSQINP2 samples

CSQ4INSG and CSQ4INSX contain system object definitions that can be included in the CSQINP2 concatenation of your MQSeries started task procedure. CSQ4INYG, CSQ4INYD, and CSQ4INYC contain some definitions that you can customize for your own objects.

CSQ4INSG system object sample

The sample CSQINP2 data set thlqual.SCSQPROC(CSQ4INSG) contains definitions for system objects for general use, which comprise:

- System default objects
- System command objects
- Event reporting objects

You must define the objects in this sample, but you need to do it only once when the subsystem is first started. Including the definitions in the CSQINP2 data set is the best way to do this. They are maintained across MQSeries subsystem shutdown and restart. You must not change the object names, but you can change their attributes if required.

If you attempt to define objects that already exist, you get messages similar to the following:

```
CSQM095I +cpf CSQMMSGP QLOCAL(SYSTEM.DEFAULT.LOCAL.QUEUE) ALREADY EXISTS
CSQM090E +cpf CSQMMSGP FAILURE REASON CODE X'00D44003'
CSQ9023E +cpf CSQMMSGP ' DEFINE QLOCAL' ABNORMAL COMPLETION
```

The objects are not damaged by this failure. If you want to leave the SYSTEM definitions data set in the CSQINP2 concatenation, you can avoid the failure messages by specifying the REPLACE attribute against each object.

System default objects: The names of the default system object definitions begin with the characters “SYSTEM.DEFAULT” or “SYSTEM.DEF”. For example, the system default local queue is named:

SYSTEM.DEFAULT.LOCAL.QUEUE

The sample describes how you can tailor these objects. These objects define the system defaults for the attributes of these MQSeries objects:

- Local queues
- Model queues
- Alias queues
- Remote queues
- Processes
- Namelists
- Channels (for distributed queuing without CICS)
- Storage classes

Customize initialization data sets

These default objects are used by DEFINE commands if no LIKE attribute is specified.

System command objects: The names of the system command objects begin with the characters SYSTEM.COMMAND.

There are two system-command objects:

1. The system-command input queue is a local queue on which commands are put before they are processed by the MQSeries command processor. It must be called SYSTEM.COMMAND.INPUT, and for normal operation it must have these attributes:

- MAXSMSGL(32 762)
- USAGE(NORMAL)
- DEFSOPT(EXCL)
- NOTRIGGER

You can specify any of the other local queue attributes as required.

2. SYSTEM.COMMAND.REPLY.MODEL is a model queue that defines the system-command reply-to queue.

You must define these objects before the MQSeries operations and control panels can be used to issue commands to an MQSeries subsystem.

Commands are normally sent using nonpersistent messages so both the system-command objects should have the DEFPSIST(NO) attribute. However, in case you wish an application to use persistent messages for commands, you should set the DEFTYPE(PERMDYN) attribute for the reply-to queue.

Event reporting objects: The names of the event reporting objects begin with the characters SYSTEM.ADMIN.

There are three system-administration objects:

- The SYSTEM.ADMIN.QMGR.EVENT queue
- The SYSTEM.ADMIN.PERFM.EVENT queue
- The SYSTEM.ADMIN.CHANNEL.EVENT queue

These queues are used for event messages; for more information about using MQSeries events, see the *MQSeries Programmable System Management* manual.

CSQ4INSX system object sample

Additional system queues must be defined in order to use non-CICS distributed queuing and clustering. The member CSQ4INSX in the thlqual.SCSQPROC library contains the queue definitions required. You can include this member in the CSQINP2 DD concatenation of the MQSeries startup procedure, or you can use the COMMAND function in CSQUTIL utility to issue the required DEFINE commands.

There are two types of object definitions:

- SYSTEM.CHANNEL.xx, needed for any distributed queuing
- SYSTEM.CLUSTER.xx, needed for clustering

Channel queues: Distributed queuing requires queues for use with sequence numbers and logical units of work identifiers (LUWID). You must ensure that a queue is available with the name SYSTEM.CHANNEL.SYNCQ. To improve

channel performance, you should define this queue with an index type of MSGID (as shown in the supplied sample queue definitions). Queue indexes are described in the *MQSeries Command Reference*.

You need to ensure that channel command queues exist for your system with the names SYSTEM.CHANNEL.INITQ and SYSTEM.CHANNEL.REPLY.INFO. You also need to ensure that the channel initiator has access to the SYSTEM.COMMAND.INPUT queue (both to get and put messages).

Cluster queues: To use MQSeries clusters, you need to define the following objects:

- A local queue called the SYSTEM.CLUSTER.COMMAND.QUEUE, which is used to communicate repository changes between queue managers. Messages written to this queue contain updates to the repository data to be applied to the local copy of the repository, or requests for repository data.
- A local queue called SYSTEM.CLUSTER.REPOSITORY.QUEUE, which is used to hold a persistent copy of the repository.
- A local queue called SYSTEM.CLUSTER.TRANSMIT.QUEUE, which is the transmission queue for all destinations in the cluster. For performance reasons you should define this queue with an index type of CORRELID (as shown in the sample queue definitions).

These queues typically contain large numbers of messages.

Cluster queue managers and repositories are described in the *MQSeries Queue Manager Clusters* manual.

CSQ4INYG object sample

The member CSQ4INYG in the thlqual.SCSQPROC library contains sample definitions that you can use for customizing your own objects for general use. It comprises:

- Storage classes
- Recommended queues
- CICS adapter objects

You cannot use this sample as is, you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the MQSeries startup procedure, or you can use the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands.

In addition to the sample definitions here, you can use the system object definitions as the basis for your own resource definitions. For example, you could make a working copy of SYSTEM.DEFAULT.LOCAL.QUEUE and name it MY.DEFAULT.LOCAL.QUEUE. You can then change any of the parameters in this copy as required. You could then issue a DEFINE command by whichever method you choose, provided you have the authority to create resources of that type.

Storage classes: Four storage classes (SYSTEM, REMOTE, DEFAULT, and NODEFINE) are required because they are used by other object definitions. Modify their attributes and add other storage classes as required.

Notes:

1. A storage class can only be changed when:
 - All queues that use this storage class are empty, and have no uncommitted activity.
 - All queues that use this storage class are closed.
2. If a message is put on a queue that names a non-existent storage class, the calling program will receive error MQRC_STORAGE_CLASS_ERROR. Alter the queue definition to give it an existing storage class name, or create the storage class named by the queue.
3. If a queue is defined and no storage class is explicitly assigned, the DEFAULT storage class is assigned automatically by MQSeries.

Recommended queues: You are recommended to have two special purpose queues of your own; a default transmission queue and a dead-letter queue. The default transmission queue is used when sending messages to another queue manager and no other suitable transmission queue is available.

The dead-letter queue is used if the message destination is not valid. The queue manager puts such messages on a local queue called the dead-letter queue. Although having a dead-letter queue is not mandatory, it should be regarded as essential, especially if you are using either distributed queuing or one of the MQSeries bridges.

In addition to defining these queues you also need to tell the queue manager their names. To do this use the ALTER QMGR command, as shown in the sample.

CICS adapter objects: The sample defines an initiation queue named CICS01.INITQ. This queue is used by the MQSeries-supplied CKTI transaction. You can change the name of this queue; however it must match the name specified in the CICS system initialization table (SIT) or SYSIN override in the INITPARM statement:

```
INITPARM=(CSQCPARM='IQ=CICS01.INITQ,...')
```

This example shows you how to define the initiation queue name in INITPARM. For more details, see the sample INITPARM in Figure 14 on page 120 and the accompanying notes.

CSQ4INXD object sample

If you are using distributed queuing (non-CICS) and not clustering, you need to set up your own queues, processes, and channels. The member CSQ4INXD in the thlqual.SCSQPROC library contains sample definitions that you can use for customizing your distributed queuing objects. It comprises:

- A set of definitions for the sending end
- A set of definitions for the receiving end
- A set of definitions for using clients

You cannot use this sample as is; you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the MQSeries startup procedure, or you can use the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands.

CSQ4INYC object sample

If you are using clustering, definitions equivalent to the channel definitions and remote queue definitions of distributed queuing are created automatically, when needed. However, some manual channel definitions are needed – a cluster-receiver channel for the cluster, and a cluster-sender definition to at least one cluster repository queue manager.

The member CSQ4INYC in the thlqual.SCSQPROC library contains sample definitions that you can use for customizing your clustering objects. It comprises:

- Definitions for the queue manager
- Definitions for the receiving channel
- Definitions for the sending channel
- Definitions for cluster queues
- Definitions for lists of clusters

You cannot use this sample as is; you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the MQSeries startup procedure, or you can use the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands.

Using the other samples

This section describes the other sample definitions that are supplied.

CSQ4DISP display sample

The sample thlqual.SCSQPROC(CSQ4DISP) contains a set of generic DISPLAY commands that display all the defined resources on your MQSeries subsystem. This includes the definitions for all MQSeries objects and definitions such as storage classes and trace. These commands can generate a large amount of output. This sample can be used in the CSQINP2 data set or as input to the COMMAND function of the CSQUTIL utility.

CSQ4DISQ distributed queuing using CICS sample

The sample thlqual.SCSQPROC(CSQ4DISQ) contains a set of commands that are required to implement distributed queuing using CICS ISC. For more information see Appendix D, “Enabling distributed queuing using CICS ISC” on page 479.

You can include this member in the CSQINP2 DD concatenation of the MQSeries startup procedure, or you can use the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands.

CSQ4IVP sample

The sample thlqual.SCSQPROC(CSQ4IVP) contains a set of DEFINE commands that are required to run the installation verification program (IVP). For more information, see “Running the installation verification program CSQ4IVP1” on page 99.

You can include this sample in the CSQINP2 data set. Once you have successfully run the IVP, you do not need to run it each time MQSeries is restarted. Therefore, you do not need to keep CSQ4IVP permanently in the CSQINP2 concatenation.

Customize initialization data sets

CSQ4INPX sample

The sample thlqual.SCSQPROC(CSQ4INPX) contains a set of commands that you might want to execute each time the channel initiator starts. You must customize this sample before use; you can then include it in the CSQINPX data set for the channel initiator.

Task 11: Create the bootstrap and log data sets

- Repeat this task for each MQSeries subsystem.

Use the supplied program CSQJU003 to prepare the bootstrap data sets (BSDSs) and log data sets. You must run this job once for each subsystem you want to define. The sample JCL and Access Method Services (AMS) control statements to run CSQJU003 to create a single or dual logging environment are held in thlqual.SCSQPROC(CSQ4BSDS). Customize and run this job to create your BSDSs and logs.

The startup procedure, CSQ4MSTR, described in “Task 7: Create procedures for the MQSeries subsystem” on page 49, refers to BSDSs in statements of the form:

```
//BSDS1 DD DSN=++HLQ++.BSDS01,DISP=SHR
//BSDS2 DD DSN=++HLQ++.BSDS02,DISP=SHR
```

The log data sets are referred to by the BSDSs.

Notes:

1. The BLKSIZE must be specified on the SYSPRINT DD statement in the CSQTLOG step. The BLKSIZE must be 629.
2. To help identify bootstrap data sets and log data sets from different MQSeries subsystems, include the subsystem name in the high level qualifier of these data sets.
3. Each BSDS requires 500 KB of storage.
4. The number of records to specify in the cluster for each log data set is:

$$\text{Number of records} = (\text{a} * \text{log switch interval required in seconds}) / 4096$$

where

$$\begin{aligned} \text{a} = & (\text{Number of MQPUTs/sec} * (\text{Average message size} + 440)) \\ & + (\text{Number of MQGETs/sec} * 72) \\ & + (\text{Number of units of recovery started} * 100) \\ & + (\text{Number of syncpoints per second} * 196) \end{aligned}$$

and

$$\text{log switch interval} = \frac{\text{time period between successive log switches}}{\text{log switches}}$$

Each log data set should have the same number of records specified and should not have secondary extents. Other than for a very small number of records, AMS rounds up the number of records so that a whole number of cylinders is allocated. The number of records actually allocated is:

$$c = (\text{INT}(\text{number of log records} / b) + 1) * b$$

Where b is the number of 4096-byte blocks per cylinder (180 for a 3390 device) and INT means round down to an integer

5. You are recommended to have at least three log data sets, and to use dual logging and log archiving. See Chapter 17, “Understanding the log and the bootstrap data set” on page 299 for more information.

Task 12: Define your page sets

- *Repeat this task for each MQSeries subsystem.*

You must define separate page sets for each MQSeries subsystem. `thlqual.SCSQPROC(CSQ4PAGE)` contains JCL and AMS control statements to define and format four page sets. The JCL runs the supplied utility program `CSQUTIL`.

The startup procedure `CSQ4MSTR` described in “Task 7: Create procedures for the MQSeries subsystem” on page 49 refers to the page sets, in a statement of the form:

```
//CSQP00nn DD DISP=OLD,DSN=xxxxxxxx
```

where *nn* is the page set number between 00 and 99, and *xxxxxxxx* is the data set that you define.

Notes:

1. Each MQSeries subsystem must have a page set 00. This is where all the object definitions used by the queue manager are kept. You should avoid putting messages on page set 00 for the reasons discussed in “Page set zero” on page 16.
2. Each MQSeries subsystem can have a maximum of 100 page sets.
3. If you intend to allow the `FORCE` option to be used with the `FORMAT` function of the utility program `CSQUTIL`, you must add the `REUSE` attribute on the `AMS DEFINE CLUSTER` statement. See page 231 for details.
4. If you intend to use the dynamic page set expansion feature, ensure that secondary extents are defined for each page set. `thlqual.SCSQPROC(CSQ4PAGE)` shows how to do this.
5. To help identify page sets from different MQSeries subsystems, include the subsystem name in the high level qualifier of the data set associated with each page set.
6. MQSeries attempts to keep data in virtual storage buffer pools for as long as it can (subject to demands for buffer pool storage). This means that the page set storage required is that to hold the maximum amount of data held in the system. If the system is closed down (using the `STOP` command), all the data held in the buffer pools is flushed out to DASD.

For queue manager object definitions (for example, queues and processes), it is simple to calculate the storage requirement because these objects are of fixed size and are permanent. For messages however, the calculation is more complex for the following reasons:

- Messages vary in size.
- Messages are transitory.
- Space occupied by messages that have been retrieved is reclaimed periodically by an asynchronous process.

“Calculating the storage requirement for messages” on page 63 describes how to calculate the space requirement for messages.

7. For page set zero, the storage required is:

- (maximum number of local queue definitions x 1010)
- + (maximum number of model queue definitions x 746)
- + (maximum number of alias queue definitions x 338)
- + (maximum number of remote queue definitions x 434)
- + (maximum number of permanent dynamic queue definitions x 1010)
- + (maximum number of process definitions x 674)
- + (maximum number of namelist definitions x 12320)
- + (maximum number of message channel definitions x 1010)
- + (maximum number of client-connection channel definitions x 1714)
- + (maximum number of server-connection channel definitions x 1010)
- + (maximum number of cluster-receiver channel definitions x 1010)
- + (maximum number of cluster-sender channel definitions x 1010)
- + (maximum number of storage class definitions x 266)

Divide this value by 4096 to determine the number of records to specify in the cluster for the page set data set.

8. The total number of objects that can be created is limited by the capacity of page set zero. There is an implementation limit on the number of local queues that can be defined, which is 524 287.

9. For page sets 01 to 99, the storage required for each page set is determined by the number and size of the messages stored on that page set.

“Calculating the storage requirement for messages” describes how to calculate the space requirement for messages. Divide this value by 4096 to determine the number of records to specify in the cluster for the page set data set.

Calculating the storage requirement for messages

This section describes how messages are stored on pages. Understanding this will help you calculate how much page set storage you need to define for your messages. To calculate the approximate space required for all messages on a page set you must consider maximum queue depth of all the queues that map to the page set and the average size of messages on those queues.

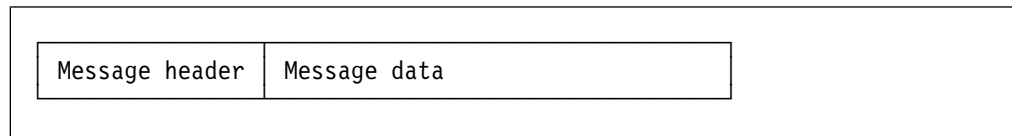
You must allow for the possibility that message “gets” might be delayed for reasons outside the control of MQSeries (for example, because of a problem with your communications protocol). In this case, the “put” rate of messages might far exceed the “get” rate. This could lead to a large increase in the number of messages stored in the page sets and a consequent increase in the storage size demanded.

Each page in the page set is 4096 bytes long. Allowing for fixed header information, each page has 4057 bytes of space available for storing messages.

When calculating the space required for each message, the first thing you need to consider is whether the message will fit on one page (a *short message*) or whether it needs to be split over two or more pages (a *long message*). When messages are split in this way, you need to allow for additional control information in your space calculations.

Define page sets

For the purposes of space calculation, a message can be represented like this:

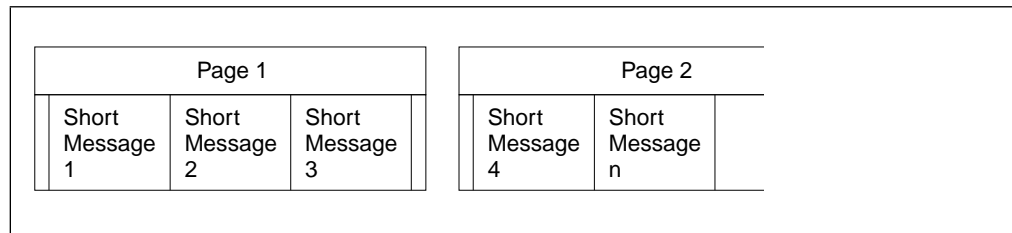


The message header section contains the message descriptor (352 bytes) and other control information, the size of which varies depending on the size of the message. The message data section contains all the actual message data, and any other headers (for example, the transmission header or the IMS bridge header).

Short messages

A short message is defined as a message that will fit on one page.

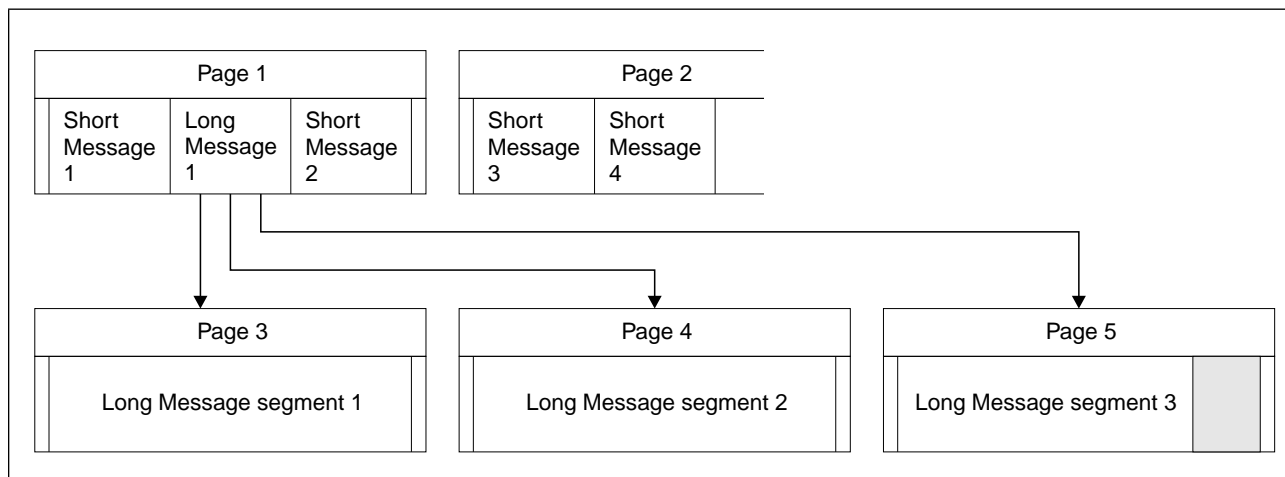
For a short message the control information is 20 bytes long. When this is added to the length of the message header, the usable space remaining on the page is 3685 bytes. If the size of the message data is 3685 bytes or less, MQSeries stores the messages in the next available space on the page, or if there is not enough space available, on the next page, as shown below:



If there is enough space remaining on the page, the next message is also stored on this page, if not, the remaining space on the page is left unused.

Long messages

If the size of the message data is greater than 3685 bytes, the message is classed as a long message. When presented with a long message, MQSeries stores the message on a series of pages, and stores control information that points to these pages in the same way that it would store a short message, as shown below:



Each segment of the long message is preceded by 8 bytes of control information, and the first segment also includes the message header portion of 352 bytes. This means that the first page contains 3697 bytes of the message data. The remaining message data is placed on subsequent pages, in 4049-byte segments. If this does not fill an exact number of pages, the remaining space in the last page is left unused.

The number of pages (n) used for a long message is calculated as follows:

$$n = \frac{\text{message data length} + 352}{4049}$$

rounded up to the nearest page

In addition to this, you need to allow space for the control information that points to the pages. The length of this (c) depends on the length of the message, and is calculated as follows:

$$c = 20 + (3n) \text{ bytes}$$

(where n is the number of pages calculated above)

This means that the total page set space required for a long message is:

$$(n * 4096) + c \text{ bytes}$$

Enabling dynamic page set expansion

Page sets can be dynamically extended while MQSeries is running. A page set can have up to 123 secondary extents, which can exist on multiple disk volumes.

Note: The maximum number of extents for a page set cataloged in an ICF catalog is between 119 and 123, depending upon the number of extents (1-5) allocated by direct access storage data management (DADSM) per allocate/extend request.

In order to use this facility, your page sets must be allocated with secondary extent values defined. If you have existing page set definitions, they cannot be altered to add secondary extent definitions. You will have to re-allocate each of your page sets with secondary extents, and then use the COPYPAGE function of CSQUTIL to copy the old versions of the page sets to the new ones.

thlqual.SCSQPROC(CSQ4PAGE) shows how to define the secondary extents, and Chapter 14, "Using the MQSeries utilities" on page 227 gives information about using CSQUTIL.

How to determine an appropriate secondary extent value

You might decide the secondary extent value by considering how many times the page set should exceed its original value. For instance, if your page set primary allocation is 1000 units (records/pages, tracks, cylinders, kilobytes, megabytes), and you want it to grow to be at most four times that size, then determine the secondary extent size from:

$$\frac{(\text{maximum size} - \text{original size})}{119}$$

$$\text{In this case, } \frac{4000 - 1000}{119} = \frac{3000}{119} = 25 \text{ or } 26$$

However, see “Number of extents available” for a description of why you might not be able to use this much disk space.

Note: If you define the size of your extent in records, IDCAMS rounds this up to map onto a physical boundary. The queue manager uses all this space for the secondary extent.

Multivolume data sets

If the definition of a page set allows it to utilize multiple volumes, the primary space is wholly contained on the first volume, and secondary extents are first allocated on the same volume, while space is available, and thereafter the next volume is used, while space is available, and so on. The process stops when you have used all the secondary extents, or no more disk space is available.

Note: This behavior differs if the page set is a data set managed by Storage Management Subsystem (SMS), and you use a storage class that uses the **GUARANTEED SPACE** attribute. Then the multivolume processing differs in that a primary extent is allocated on each volume when the page set is defined. Thereafter, secondary extents are allocated, as before, except that when the services of a new volume are required, the pre-allocated secondary extent is used.

Number of extents available

Note that the Data Facility Product (DFP) uses up to five non-contiguous areas of disk to satisfy the total space requirements of a primary or secondary extent. This means, in the worst case of badly fragmented disk space, that you might only get around 22 times the secondary space allocated before you reach the maximum extent limit.

Task 13: Tailor your system parameter module

- Repeat this task for each MQSeries subsystem, as required.

The MQSeries system parameter module controls the logging, archiving, tracing, and connection environments that MQSeries uses in its operation. The system parameter module has three macros as follows:

Macro name	Purpose
CSQ6SYSP	Specifies the connection and tracing parameters, see page 68
CSQ6LOGP	Controls log initialization, see page 74
CSQ6ARVP	Controls archive initialization, see page 76

MQSeries supplies a default system parameter module, CSQZPARM, which is invoked automatically if you issue the START QMGR command (without a PARM parameter) to start an instance of MQSeries. CSQZPARM is in the APF-authorized library thlqual.SCSQAUTH also supplied with MQSeries. The values of these parameters are displayed as a series of messages when you start MQSeries.

See the *MQSeries Command Reference* manual for more information about the START command and “Starting MQSeries” on page 192 for more information about how this command is used.

Creating your own system parameter module

If CSQZPARM does not contain the system parameters you want, you can create your own system parameter module using the sample JCL provided in thlqual.SCSQPROC(CSQ4ZPRM).

To create your own system parameter module:

1. Make a working copy of the JCL sample.
2. Edit the parameters for each macro in the copy as required. If you remove any parameters from the macro calls, the default values are automatically picked up at run time.
3. Replace the placeholder ++NAME++ with the name that the load module is to take (this can be CSQZPARM).
4. If your assembler is not high level assembler, change the JCL as required by your assembler.
5. Run the JCL to assemble and link-edit the tailored versions of the system parameter macros to produce a load module. This is the new system parameter module with the name that you have specified.
6. Put the load module produced in an APF-authorized user library.
7. Include this library in the MQSeries started task procedure STEPLIB. This library name must come before the library thlqual.SCSQAUTH in STEPLIB.

8. Invoke the new system parameter module when you start MQSeries. For example, if the new module is named NEWMODS, issue the command:

```
START QMGR PARM(NEWMODS)
```

Note: If you choose to name your module CSQZPARM, you do not need to specify the PARM parameter on the START QMGR command.

Fine tuning a system parameter module

MQSeries also supplies a set of three assembler source modules, which can be used to fine tune an existing system parameter module. These modules are in library thlqual.SCSQASMS. Typically, you use these modules in a test environment to change the default parameters in the system parameter macros. Each source module calls a different system parameter macro:

This assembler source module...	Calls this macro...
CSQFSYSP	CSQ6SYSP (connection and tracing parameters)
CSQJLOGP	CSQ6LOGP (log initialization)
CSQJARVP	CSQ6ARVP (archive initialization)

This is how you use these modules:

1. Make working copies of each assembler source module in a user assembler library.
2. Edit your copies by adding or altering the values of any parameters as required.
3. Assemble your copies of any edited modules to create object modules in a user object library.
4. Link-edit these object code modules with an existing system parameter module to produce a load module that is the new system parameter module.
5. Ensure that new system parameter module is a member of a user authorized library.
6. Include this library in the MQSeries started task procedure STEPLIB. This library must come before the library thlqual.SCSQAUTH in STEPLIB.
7. Invoke the new system parameter module by issuing a START QMGR command, specifying the new module name in the PARM parameter, as before.

Using CSQ6SYSP

Use CSQ6SYSP to set system parameters.

The default parameters for CSQ6SYSP are shown in Table 8. If you want to change any of these values, refer to the detailed descriptions of the parameters.

Parameter	Description	Default value
CMDUSER	The default user ID for command security checks.	CSQOPR

Table 8 (Page 2 of 2). Default values of CSQ6SYSP parameters

Parameter	Description	Default value
CTHREAD	Maximum number of connections from batch, CICS, IMS, and TSO tasks to a single instance of MQSeries.	300
EXITLIM	Time (in seconds) for which queue-manager exits can execute during each invocation.	30
EXITTCB	How many started server tasks to use to run queue manager exits.	8
IDBACK	Maximum number of connections to a single instance of MQSeries from batch or TSO background tasks.	20
IDFORE	Maximum number of connections to a single instance of MQSeries from TSO foreground tasks.	100
LOGLOAD	Number of log records written by MQSeries between the start of one checkpoint and the next.	10 000
OTMACON	OTMA connection parameters.	See below
QMCCSID	Coded character set identifier for the queue manager.	0
ROUTCDE	Message routing code assigned to messages not solicited from a specific console.	1
SMFACCT	Specifies whether SMF accounting data is to be collected when MQSeries is started.	NO
SMFSTAT	Specifies whether SMF statistics are to be collected when MQSeries is started.	NO
STATIME	Default time (in minutes) between each gathering of statistics.	30
TRACSTR	Specifies whether tracing is to be started automatically.	NO
TRACTBL	Size of trace table, in 4 KB blocks, to be used by the global trace facility.	99 (396 KB)
WLMTIME	Time (in minutes) between scanning the queue index for WLM-managed queues.	30

CMDUSER

Specifies the default user ID used for command security checks. This user ID must be defined to the ESM (for example, RACF). Specify a name of 1 through 8 alphanumeric characters. The first character must be alphabetic.

The default is CSQOPR.

CTHREAD

Specifies the maximum number of connections from batch, CICS, IMS, TSO, and the channel initiator to a single instance of MQSeries.

CICS connections are not limited in the same way as TSO and batch connections. During the connection of the main CICS TCB to MQSeries, the adapter attempts to attach up to eight OS/390 subtasks (TCBs) to be used by this CICS system. This means each CICS system connected takes up nine of the connections specified on CTHREAD, so you must increase CTHREAD by

nine for each CICS system connected. You must also increase the value of CTHREAD by one for each instance of the task initiator CKTI.

For IMS connections, the number of connections required is one for the control region, and one for each dependant region connected to MQSeries. For each IMS MPP or IFP region that is defined to permit MQSeries connections through either a specific SSM= EXEC parameter or through the control region default, a thread is created when the first application is scheduled in that region, regardless of whether that application invokes any MQSeries calls. The value you set for CTHREAD should take account of this.

For distributed queuing (without CICS), the number of connections required by the channel initiator address space depends on the number of adapter subtasks and dispatchers there will be; see "Using CSQ6CHIP" on page 83.

When the number of connections reaches the limit set by CTHREAD, any further requests for a connection are suspended until a spare slot becomes available. For example, an **MQDISC** call releases that connection. For planning purposes, the value of CTHREAD must be greater than the maximum of IDBACK, IDFORE, and the number of potential connections from the CICS, IMS, and channel initiator address spaces.

Specify a number in the range 1 through 32767.

The default is 300.

Note that this controls the number of *connections*; a connection might involve more than one thread. See "Connections and threads" on page 280 for more information.

EXITLIM

Specifies the time, in seconds, allowed for each invocation of the queue manager exits. (This parameter has no effect on channel exits.)

Specify a value in the range 5 through 9999.

The default is 30. The queue manager polls exits that are running every 30 seconds. On each poll, any that have been running for more than the time specified by EXITLIM are forcibly terminated.

EXITTCB

Specifies the number of started server tasks to use to run exits in the queue manager. (This parameter has no effect on channel exits.)

Specify a value in the range 0 through 99. A value of 0 means that no exits can be run.

The default is 8.

IDBACK

Specifies the maximum number of background batch and TSO connections to a single instance of MQSeries. The value of IDBACK is related to those of IDFORE and CTHREAD. See the description of the CTHREAD parameter for more information.

Specify a number in the range 1 through 32767.

The default is 20.

IDFORE

Specifies the maximum number of TSO foreground connections to MQSeries.

The value of IDFORE is related to those of IDBACK and CTHREAD. See the description of the CTHREAD parameter for more information.

The number of TSO connections might be greater than the number of concurrent TSO users if, for example, users split their ISPF screens.

Specify a number in the range 0 through 32767.

The default is 100.

LOGLOAD

Specifies the number of log records that MQSeries writes between the start of one checkpoint and the next. MQSeries starts a new checkpoint after the number of records that you specify has been written.

Specify a value in the range 200 through 16 000 000.

The larger the number, the better the performance of MQSeries. However, abnormal restart takes longer if the parameter is set to a large number.

The default is 10 000.

OTMACON

OTMA parameters. This keyword takes five positional parameters, as shown below:

OTMACON = (Group,Member,Druexit,Age,TpipePrefix)

Group

This is the name of the XCF group to which this particular instance of MQSeries belongs.

It can be 1 through 8 characters long and must be entered in uppercase characters.

The default is blanks, which indicates that MQSeries should not attempt to join an XCF group.

Member

This is the member name of this particular instance of MQSeries within the XCF group.

It can be 1 through 16 characters long and must be entered in uppercase characters.

The default is the 4-character queue manager name.

Druexit

This specifies the name of the OTMA destination resolution user exit to be run by IMS.

It can be 1 through 8 characters long.

The default is DFSYDRU0.

This parameter is optional; it is required if MQSeries is to receive messages from an IMS application that was not started by MQSeries. The name should correspond to the destination resolution user exit coded in the IMS system. For more information see Appendix B, "Using OTMA exits in IMS" on page 473.

Age

This represents the length of time, in seconds, that a user ID from MQSeries is considered previously verified by IMS.

It can be in the range 0 through 2 147 483 647.

The default is 2 147 483 647.

TpipePrefix

This represents the prefix to be used for Tpipe names.

It comprises three characters; the first character is in the range A through Z, subsequent characters are A through Z or 0 through 9. The default is CSQ.

This is used each time MQSeries creates a Tpipe; the rest of the name is assigned by MQSeries. You cannot set the full Tpipe name for any Tpipe created by MQSeries.

QMCCSID

Specifies the default coded character set identifier that the queue manager (and therefore distributed queuing) is to use.

Specify a value in the range 0 through 65 535. 0 means use the CCSID currently set or, if none is set, use CCSID 500.

The default is 0.

ROUTCDE

Specifies the default OS/390 message routing code assigned to messages that are not sent in direct response to an MQSeries command.

Specify one of:

1. A value in the range 1 through 16, inclusive.
2. A list of values, separated by a comma and enclosed in parentheses. Each value must be in the range 1 through 16, inclusive.

The default is 1.

For more information about OS/390 routing codes, see the *MVS Routing and Descriptor Codes* manual.

SMFACCT

Specifies whether MQSeries sends accounting data to SMF automatically when MQSeries starts.

Specify one of:

- NO** Do not start gathering accounting data automatically.
YES Start gathering accounting data automatically for the default class 1.

The default is NO.

SMFSTAT

Specifies whether to gather SMF statistics automatically when MQSeries starts.

Specify one of:

- NO** Do not start gathering statistics automatically.
YES Start gathering statistics automatically for the default class 1.

The default is NO.

STATIME

Specifies the default time, in minutes, between each gathering of statistics.

Specify a number in the range 1 through 1440.

The default is 30.

TRACSTR

Specifies whether global tracing is to start automatically.

Specify one of:

NO Do not start global tracing automatically.

YES Start global tracing automatically for the default class, class 1.

integers A list of classes for which global tracing is to be started automatically in the range 1 through 4.

***** Start global trace automatically for all classes.

The default is NO if you do not specify the keyword in the macro.

Note: The supplied default system parameter load module (CSQZPARM) has TRACSTR=YES (set in the assembler module CSQFSYSP). If you do not want to start tracing automatically, you can change this after you have successfully started your MQSeries subsystem. Refer to “Fine tuning a system parameter module” on page 68 for information about how to do this.

For details about the START TRACE command, see the *MQSeries Command Reference* manual.

TRACTBL

Specifies the default size, in 4 KB blocks, of trace table where the global trace facility stores MQSeries trace records.

Specify a value in the range 1 through 999.

Note: Storage for the trace table is allocated in the ECSA above the line. Therefore, you must select this value with care.

The default is 99. This is equivalent to 396 KB.

WLMTIME

Specifies the time (in minutes) between each scan of the indexes for WLM-managed queues.

Specify a value in the range 1 through 9999.

The default is 30.

Using CSQ6LOGP

Use CSQ6LOGP to establish your logging options.

The default parameters for CSQ6LOGP are shown in Table 9. If you need to change any of these values, refer to the detailed descriptions of the parameters.

Parameter	Description	Default value
INBUFF	Active and archive logs input buffer size.	28 KB
MAXALLC	Maximum number of archive log volumes that can be allocated.	3
MAXARCH	Maximum number of archive log volumes that can be recorded.	500
OFFLOAD	Archiving on or off.	YES (ON)
OUTBUFF	Size of output buffer storage for active and archive log data sets.	400 KB
TWOACTV	Single or dual active logging.	YES (dual)
TWOARCH	Single or dual archive logging.	YES (dual)
TWOBSDS	Single or dual BSDS.	YES (dual BSDS)
WRTHRSH	Number of output buffers to be filled before they are written to the active log data sets.	20

INBUFF

Specifies the size, in kilobytes, of the input buffer for reading the active and archive logs during recovery. Use a decimal number in the range 28 through 60. The value specified is rounded up to a multiple of 4.

The default is 28 KB.

MAXALLC

Specifies the maximum number of archive log volumes that can be allocated concurrently for input mode. Archive log data sets are read when required for recovery, system restart, or dynamic backout. Use a decimal number in the range 1 through 99.

For information about the logs, see Chapter 17, "Understanding the log and the bootstrap data set" on page 299.

The default is 3.

MAXARCH

Specifies the maximum number of archive log volumes that can be recorded in the BSDS. When this number is exceeded, recording begins again at the start of the BSDS.

Use a decimal number in the range 10 through 1000.

For information about the logs and BSDS, see Chapter 17, "Understanding the log and the bootstrap data set" on page 299.

The default is 500.

OFFLOAD

Specifies whether archiving is on or off.

Specify either:

YES Archiving is on
NO Archiving is off

The parameter cannot be blank.

Attention: Do **not** switch archiving off unless you are working in a test environment. If you do switch it off, you cannot guarantee that data will be recovered in the event of a system or transaction failure. See the note about archiving in “What logs are” on page 299.

The default is YES.

OUTBUFF

Specifies the total size, in kilobytes, of the storage to be used by MQSeries for output buffers for writing the active and archive log data sets. Each output buffer is 4 KB.

The parameter cannot be blank, and must be in the range 40 through 4000. The value specified is rounded up to a multiple of 4.

The default is 400 KB.

TWOACTV

Specifies single or dual active logging.

Specify either:

NO Single active logs
YES Dual active logs

The parameter cannot be blank.

The default is YES.

TWOARCH

Specifies the number of archive logs that MQSeries produces when the active log is off-loaded.

Specify either:

NO Single archive logs
YES Dual archive logs

This parameter cannot be blank even if the OFFLOAD parameter is specified as NO.

The default is YES.

TWOBSDS

Specifies the number of bootstrap data sets.

Specify either:

NO Single BSIDS
YES Dual BSIDS

This parameter cannot be left blank.

The default is YES.

WRTHRS

Specifies the number of 4 KB output buffers to be filled before they are written to the active log data sets.

The larger the number of buffers, the less often the write takes place, and this improves the performance of MQSeries. The buffers might be written before this number is reached if significant events, such as a commit point, occur.

Specify the number of buffers in the range 1 through 256.

The default is 20.

Using CSQ6ARVP

Use CSQ6ARVP to establish your archiving environment.

The default parameters for CSQ6ARVP are shown in Table 10. If you need to change any of these values, refer to the detailed descriptions of the parameters. See "Planning your archive storage" on page 310 for more information.

Table 10. Default values of CSQ6ARVP parameters

Parameter	Description	Default value
ALCUNIT	Units in which primary and secondary space allocations are made.	BLK (blocks)
ARCPFX1	Prefix for first archive log data set name.	CSQARC1
ARCPFX2	Prefix for second archive log data set name.	CSQARC2
ARCRETN	The retention period of the archive log data set in days.	9999
ARCWRTC	List of route codes for messages to the operator about archive log data sets.	1,3,4
ARCWTOR	Whether to send message to operator and wait for reply before trying to mount an archive log data set.	YES
BLKSIZE	Block size of archive log data set.	20 480
CATALOG	Whether archive log data sets are cataloged in the ICF.	NO
COMPACT	Whether archive log data sets should be compacted.	NO
PRIQTY	Primary space allocation for DASD data sets.	4 320
PROTECT	Whether archive log data sets are protected by ESM profiles when the data sets are created.	NO
QUIESCE	Maximum time, in seconds, allowed for quiesce when ARCHIVE LOG with MODE(QUIESCE) specified.	5
SECQTY	Secondary space allocation for DASD data sets. See the ALCUNIT parameter for the units to be used.	540
TSTAMP	Whether the archive data set name should include a time stamp.	NO
UNIT	Device type or unit name on which archive log data sets are stored.	TAPE

ALCUNIT

Specifies the unit in which primary and secondary space allocations are made.

Specify one of:

CYL	Cylinders
TRK	Tracks
BLK	Blocks

You are recommended to use BLK because it is independent of the device type.

The default is BLK.

ARCPFX1

Specifies the prefix for the first archive log data set name.

See the TSTAMP parameter for a description of how the data sets will be named and for restrictions on the length of ARCPFX1.

This parameter cannot be left blank.

You might need to authorize the MQSeries subsystem to create archive logs with this prefix.

The default is CSQARC1.

ARCPFX2

Specifies the prefix for the second archive log data set name.

See the TSTAMP parameter for a description of how the data sets will be named and for restrictions on the length of ARCPFX2.

This parameter cannot be blank even if the TWOARCH parameter is specified as NO.

You might need to authorize the MQSeries subsystem to create archive logs with this prefix.

The default is CSQARC2.

ARCRETN

Specifies the retention period, in days, to be used when the archive log data set is created.

The parameter must be in the range 0 through 9999.

For more guidance, see "Automatic archive log data set deletion" on page 317.

The default is 9999.

ARCWRTC

Specifies the list of OS/390 routing codes for messages about the archive log data sets to the operator. This field is ignored if ARCWTOR is set to NO.

Specify up to 14 routing codes, each with a value in the range 1 through 16. You must specify at least one code. Separate codes in the list by commas, not by blanks.

The default is the list of values: 1,3,4.

For more information about OS/390 routing codes, see the *MVS Routing and Descriptor Codes* manual.

ARCWTOR

Specifies whether a message is to be sent to the operator and a reply is received before attempting to mount an archive log data set.

Other MQSeries users might be forced to wait until the data set is mounted, but they are not affected while MQSeries is waiting for the reply to the message.

Specify either:

YES The device needs a long time to mount archive log data sets. For example, a tape drive.

NO The device does not have long delays. For example, DASD.

The default is YES.

BLKSIZE

Specifies the block size of the archive log data set. The block size you specify must be compatible with the device type you specify in the UNIT parameter.

The parameter must be in the range 4 097 through 28 672. The value you specify is rounded up to a multiple of 4 096.

This parameter is ignored for data sets that are managed by the storage management subsystem (SMS).

If the archive log data set is written to DASD, you are recommended to choose the maximum block size that will allow 2 blocks per track. For example, for a 3390 device, you should use a block size of 24 576.

If the archive log data set is written to tape, specifying the largest possible block size improves the speed of reading the archive log.

The default is 20 480.

CATALOG

Specifies whether archive log data sets are cataloged in the primary integrated catalog facility (ICF) catalog.

Specify either:

NO Archive log data sets are not cataloged

YES Archive log data sets are cataloged

This parameter cannot be blank.

All archive log data sets allocated on DASD must be cataloged. If you archive to DASD with the CATALOG parameter set to NO, message CSQJ072E is displayed each time an archive log data set is allocated, and MQSeries catalogs the data set.

The default is NO.

COMPACT

Specifies whether data written to archive logs is to be compacted. This option applies only to a 3480 or 3490 device that has the improved data recording capability (IDRC) feature. When this feature is turned on, hardware in the tape control unit writes data at a much higher density than normal, allowing for more data on each volume. Specify NO if you do not use a 3480 device with the IDRC feature or a 3490 base model, with the exception of the 3490E. Specify YES if you want the data to be compacted.

Specifying YES adversely affects performance. Also be aware that data compressed to tape can be read only using a device that supports the IDRC feature. This can be a concern if you have to send archive tapes to another site for remote recovery.

Specify either:

NO Do not compact the data sets
YES Compact the data sets

The default is NO.

PRIQTY

Specifies the primary space allocation for DASD data sets in ALCUNITs.

The value must be greater than zero.

The default is 4 320.

This quantity must be sufficient for a copy of both the log data set and its corresponding BSDS. To determine the necessary value, follow this procedure:

1. Determine the number of active log records actually allocated (c) as explained in "Task 11: Create the bootstrap and log data sets" on page 61.
2. Determine the number of 4096-byte blocks in each archive log block:

$$d = \text{BLKSIZE} / 4096$$

Where BLKSIZE is the rounded up value

3. If ALCUNIT=BLK:

$$\text{PRIQTY} = \text{INT}(c / d)$$

where INT means round down to an integer

If ALCUNIT=TRK:

$$\text{PRIQTY} = \text{INT}(c / (d * \text{INT}(e/\text{BLKSIZE}))) + 1$$

where e is the number of bytes per track (56664 for a 3390 device) and INT means round down to an integer

If ALCUNIT=CYL:

$$\text{PRIQTY} = \text{INT}(c / (d * \text{INT}(e/\text{BLKSIZE}) * f)) + 1$$

where f is the number of tracks per cylinder (5 for a 3390 device) and INT means round down to an integer

For information about how large to make your log and archive data sets, see "Task 11: Create the bootstrap and log data sets" on page 61 and "Task 12: Define your page sets" on page 62.

PROTECT

Specifies whether archive log data sets are to be protected by discrete ESM (external security manager) profiles when the data sets are created.

Specify either:

NO Profiles are not created.

- YES** Discrete data set profiles are created when logs are offloaded. If you specify YES:
- ESM protection must be active for MQSeries.
 - The user ID associated with the MQSeries address space must have authority to create these profiles.
 - The TAPEVOL class must be active if you are archiving to tape.
- Otherwise, offloads will fail.

The default is NO.

QUIESCE

Specifies the maximum time in seconds allowed for the quiesce when an +cpf ARCHIVE LOG command is issued with MODE QUIESCE specified.

The parameter must be in the range 1 through 999.

The default is 5.

SECQTY

Specifies the secondary space allocation for DASD data sets in ALCUNITs.

The parameter must be greater than 0.

The default is 540.

TSTAMP

Specifies whether the archive log data set name has a time stamp in it.

Specify either:

- NO** Names do not include a time stamp. The archive log data sets will be named:

arcpxi.Annnnnn

Where *arcpxi* is the data set name prefix specified by ARCPFX1 or ARCPFX2. *arcpxi* can have up to 35 characters.

- YES** Names include a time stamp. The archive log data sets will be named:

arcpxi.cyyddd.Thhmsst.Annnnnn

where *c* is 'D' for the years up to and including 1999 or 'E' for the year 2000 and later, and *arcpxi* is the data set name prefix specified by ARCPFX1 or ARCPFX2. *arcpxi* can have up to 19 characters.

- EXT** Names include a time stamp. The archive log data sets will be named:

arcpxi.Dyyyyddd.Thhmsst.Annnnnn

Where *arcpxi* is the data set name prefix specified by ARCPFX1 or ARCPFX2. *arcpxi* can have up to 17 characters.

The default is NO.

UNIT

Specifies the device type or unit name of the device that is used to store archive log data sets.

Specify a device type or unit name of 1 through 8 alphanumeric characters. The first character must be alphabetic.

This parameter cannot be blank.

If you archive to DASD, you can specify a generic device type with a limited volume range.

If you archive to DASD:

- Make sure that the primary space allocation is large enough to contain all the data from the active log data sets.
- Make sure that the archive log data set catalog option (CATALOG) is set to YES.
- The archive log data sets cannot extend to another volume.

If you archive to TAPE, MQSeries can extend to a maximum of 20 volumes.

The default is TAPE.

Task 14: Tailor the channel initiator parameter module

- *Repeat this task for each MQSeries subsystem, as required.*
- *Omit this task if you are using the CICS mover.*

This process is analogous to tailoring the system parameter module (see “Task 13: Tailor your system parameter module” on page 67).

The channel initiator parameter module controls how distributed queuing operates. It has the single macro, CSQ6CHIP.

MQSeries supplies a default parameter module, CSQXPARM, which is invoked automatically if you issue the START CHINIT command (without a PARM parameter) to start a channel initiator. “Using CSQ6CHIP” on page 83 lists the default values for the supplied CSQXPARM. CSQXPARM is in the APF-authorized library thlqual.SCSQAUTH, also supplied with MQSeries.

The values of these parameters are displayed as a series of messages each time you start the channel initiator.

Creating your own channel initiator parameter module

In most cases you will need to create your own parameter module. If you are using LU 6.2 communications, you will have to do this because you will at least need to set the outbound LU name to be used. If you are using TCP/IP, you will probably need to set TCPTYPE and TCPNAME.

To create your own parameter module, use the sample JCL provided in thlqual.SCSQPROC(CSQ4XPRM):

1. Make a working copy of the JCL sample.
2. Edit the parameters in the copy as required. See “Using CSQ6CHIP” on page 83 for more information about each parameter. If you remove any parameters from the macro call, the default values are automatically picked up at run time.
3. Replace the placeholder ++NAME++ with the name that the load module is to take. (This can be CSQXPARM.)
4. If your assembler is not high-level assembler, change the JCL as required by your assembler.
5. Run the JCL to assemble and link-edit the tailored versions of the channel initiator parameter macros to produce a load module. This is the new channel initiator parameter module with the name that you have specified.
6. Put the load module produced in an APF-authorized user library.
7. Include this library in the channel initiator started task procedure STEPLIB. This library name must come before the library thlqual.SCSQAUTH in STEPLIB.

- Invoke the new channel initiator parameter module when you start the channel initiator. For example, if the new module is named NEWMODS, issue the command:

```
START CHINIT PARM(NEWMODS)
```

Note: If you choose to name your module CSQXPARM, you do not need to specify the PARM parameter on the START CHINIT command.

Using CSQ6CHIP

Use CSQ6CHIP to set channel initiator parameters.

The default parameters for CSQ6CHIP are shown in Table 11. If you want to change any of these values, refer to the detailed descriptions of the parameters.

Parameter	Description	Default value
ACTCHL	The maximum number of channels that can be active.	CURRCHL
ADAPS	The number of adapter subtasks to use for processing MQI calls.	8
CURRCHL	The maximum number of channels that can be current.	200
DISPS	The number of dispatchers to use.	5
LSTRTMR	The interval, in seconds, between listener restart attempts.	60
LUNAME	The name of the LU to use for outbound transmissions.	Blank
LU62ARM	APPCPMxx SYS1.PARMLIB member name suffix.	Blank
LU62CHL	The maximum number of channels that can be current and use the LU 6.2 transmission protocol.	CURRCHL
TCPCHL	The maximum number of channels that can be current and use the TCP/IP transmission protocol.	CURRCHL
TCPKEEP	Whether the TCP KEEPALIVE facility is to be used or not.	NO
TCPNAME	The name of the TCP/IP address space or system that will be used.	TCPIP
TCPTYPE	TCP/IP interface method.	OESOCKET
TRAXSTR	Whether trace should start automatically or not.	YES
TRAXTBL	The size of the trace data space in MB.	2

ACTCHL

Specifies the maximum number of channels that can be active.

Specify a value in the range 1 through 9999.

The default value is CURRCHL.

Tailor the channel initiator parameter module

ADAPS

Specifies the number of adapter subtasks to use for processing MQI calls. As a guideline, the ratio of adapters to dispatchers (the DISPS parameter) should be about 8 to 5. However, if you have only a small number of channels, you do not have to decrease the value of this parameter from the default value.

Specify a value in the range 0 through 9999.

The default value is 8.

CURRCHL

Specifies the maximum number of channels that can be current (including server-connection channels with connected clients).

Specify a value in the range 1 through 9999.

The default value is 200.

DISPS

Specifies the number of dispatchers to use for the channel initiator. As a guideline, allow one dispatcher for each 50 current channels. However, if you have only a small number of channels, you do not have to decrease the value of this parameter from the default value.

If you are using TCP/IP, the greatest number of dispatchers that will be used for TCP/IP channels is 100, even if you specify a larger value here.

Specify a value in the range 1 through 9999.

The default value is 5.

LSTRTMR

Specifies the time interval (in seconds) between attempts by MQSeries to restart the listener if there has been an APPC or TCP/IP failure.

Specify a value in the range 5 through 9999.

The default value is 60.

Note: This parameter is ignored if you are using the IUCV or TCPAccess interfaces to TCP/IP.

LUNAME

Specifies the name of the LU to use for outbound LU 6.2 transmissions. This must be set to the same LU that will be used for inbound transmissions by the listener.

Specify the LU name.

The default is blank, which means that the APPC/MVS default LU should be used; this is variable, so LUNAME should always be set if you are using LU 6.2.

LU62ARM

Specifies the suffix of the SYS1.PARMLIB member APPCPMxx, that nominates the LUADD for this channel initiator. The channel initiator issues the OS/390 command SET APPC=xx when it starts.

Specify the two-character suffix.

The default is blank which means that no SET APPC=xx is issued.

LU62CHL

Specifies the maximum number of channels that can be current or clients that can be connected, that use the LU 6.2 transmission protocol. If 0, the LU 6.2 transmission protocol is not used.

Specify a value in the range 0 through 9999.

The default value is CURRCHL.

TCPCHL

Specifies the maximum number of channels that can be current or clients that can be connected, that use the TCP/IP transmission protocol. If 0, the TCP/IP transmission protocol is not used.

The maximum number of TCP/IP sockets used is TCPCHL+DISPS. The OpenEdition MAXFILEPROC parameter (specified in the BPXPRMxx member of SYS1.PARMLIB) controls how many sockets each task is allowed, and thus how many channels each dispatcher is allowed. The number of channels using TCP/IP in this case is limited to MAXFILEPROC*DISPS.

Specify a value in the range 0 through 9999.

Note: TCP/IP might not support as many as 9999 channels.

The default value is CURRCHL.

TCPKEEP

Specifies whether the TCP KEEPALIVE facility, as specified by the KEEPALIVEOPTIONS statement in the TCP profile configuration data set, is to be used or not.

Specify YES or NO.

The default is NO.

TCPNAME

Specify the name of the TCP/IP system that you are using. This depends on the type of TCP/IP interface that you are using:

IUCV

The name of the TCP/IP address space

OpenEdition Sockets

The name of the OpenEdition stack for TCP/IP, as specified in the SUBFILESYSTYPE NAME parameter in the BPXPRMxx member of SYS1.PARMLIB.

TCPaccess

The name of the TCPaccess subsystem.

The default is TCPIP.

TCPTYPE

Specifies the type of TCP/IP interface to be used.

Specify one of the following:

IUCV	IUCV interface
OESOCKET	OpenEdition sockets interface
SNSTCPACCESS	TCPaccess native interface

The default is OESOCKET.

Tailor the channel initiator parameter module

See “Task 2: Choose the distributed queuing facility” on page 40 for information about these interfaces, and Table 12 on page 86 for a summary of the settings.

TRAXSTR

Specifies whether trace should start automatically or not.

Specify YES or NO.

The default is YES.

TRAXTBL

Specifies the size of the trace data space (in MB).

Specify a value in the range 0 through 2048.

The default value is 2.

Note: Whenever you use large OS/390 data spaces, you should ensure that sufficient auxiliary storage is available on your system to support any related OS/390 paging activity. You might also need to increase the size of your SYS1.DUMP data sets.

Notes:

1. The channel initiator makes a number of connections to the queue manager that must be allowed for when setting the CTHREAD system parameter (see “Using CSQ6SYSP” on page 68). The number of connections is up to 6 plus the value of ADAPS plus the value of DISPS.
2. Each dispatcher and each adapter subtask uses a separate OS/390 task. As a guideline, keep the total number of dispatchers and adapter subtasks below 20.

Product	Interface	Library	TCPTYPE	TCPNAME
IBM TCP/IP	OpenEdition sockets	SCSQMVR1	OESOCKET	OpenEdition TCP/IP stack name
IBM TCP/IP	IUCV	SCSQMVR1	IUCV	TCP/IP address space name
Interlink TCPaccess	OpenEdition sockets	SCSQMVR1	OESOCKET	OpenEdition TCP/IP stack name
Interlink TCPaccess	IUCV	SCSQMVR1	IUCV	TCP/IP address space name
Interlink TCPaccess	Native TCPaccess	SCSQMVR2	SNSTCPACCESS	TCPaccess subsystem name

Task 15: Set up Batch, TSO, and RRS adapters

- *Repeat this task for each MQSeries subsystem as required.*

The Batch/TSO and RRS adapters are the interface between OS/390 application programs running under JES, TSO, or OS/390 OpenEdition and an MQSeries subsystem. They enable OS/390 application programs to use the MQI. They are generally referred to in this book as the batch adapter and the RRS adapter. “MQSeries and OS/390 Batch and TSO” on page 22 describes these adapters.

To make the adapters available to batch and TSO applications, add the following MQSeries libraries to the STEPLIB concatenation for your batch application or TSO logon procedure:

- thlqual.SCSQANLx
- thlqual.SCSQAUTH

where *x* is the language letter for your national language.

Note: If the adapter detects an unexpected MQSeries subsystem error, it issues an OS/390 SNAP dump to DDname CSQSNAP, and issues reason code MQRC_UNEXPECTED_ERROR to the application.

If this occurs, rerun the job with a CSQSNAP DD statement included in the JCL, and contact your IBM support center.

The supplied program CSQBDEFV improves the portability of your application programs. In CSQBDEFV, you can specify the name of a subsystem to be connected to rather than specifying it in the **MQCONN** call in an application program. You can create a new version of CSQBDEFV for each subsystem. To do this, follow these steps:

1. Copy the MQSeries assembler program CSQBDEFV from thlqual.SCSQASMS to a user library.
2. The supplied program contains the default subsystem name CSQ1. You can retain this name for testing and installation verification. For production subsystems, you can change the NAME=CSQ1 to your one- to four-character subsystem name, or use CSQ1.
3. Assemble and link-edit the program to produce the CSQBDEFV load module. For the assembly, include the library thlqual.SCSQMACS in your SYSLIB concatenation; use the link-edit parameters RENT,AMODE=31,RMODE=ANY. This is shown in the sample JCL in thlqual.SCSQPROC(CSQ4DEFV). Then include the load library in the OS/390 Batch or the TSO JOBLIB or STEPLIB.

Task 16: Set up the operations and control panels

- *You need only perform this task once.*

To set up the operations and control panels you must first set up the libraries that contain the required panels, EXECs, messages, and tables. To do this, you must take into account which national language feature is to be used for the panels. When you have done this, you can optionally:

- Update the main ISPF menu for MQSeries operations and control panels
- Change the function key settings

Setting up the libraries

Follow these steps to install the MQSeries operations and control panels:

1. Include the library thlqual.SCSQEXEC in your SYSEXEC or SYSPROC concatenation. This library, which is allocated with a fixed-block 80 record format during installation, contains the required EXECs.

Notes:

- a. You should put these EXECs into your SYSEXEC concatenation. However, if you want to put them in SYSPROC, it must have a record length of 80 bytes. If you move the EXECs from SYSPROC to SYSEXEC and your system searches SYSPROC before SYSEXEC, you must ensure that the EXECs are deleted from SYSPROC.
 - b. Ensure that all the libraries contained in your concatenations are either in the same format (F, FB, V, VB) and have the same block size, or in the order of decreasing block sizes. Otherwise, you might have problems trying to use these panels.
2. Add SCSQAUTH to the TSO logon procedure STEPLIB if it is not in LINKLIB.
 3. You can either install the MQSeries panel libraries permanently in your ISPF library setup, or allow them to be set up dynamically when the panels are used. For the former choice, you need to do the following:
 - a. Include the name of the library containing the operations and control panel definitions in your ISPPLIB concatenation. The name is thlqual.SCSQPNLx, where x is the language letter for your national language.
 - b. Install the required tables in your ISPTLIB concatenation. The name is thlqual.SCSQTBLx, where x is the language letter for your national language.
 - c. Install the required messages in your ISPMLIB concatenation. The name is thlqual.SCSQMSGx, where x is the language letter for your national language.
 - d. Include the library thlqual.SCSQSKL in your ISPSLIB concatenation. This library contains the required skeletons for data set tailoring.
 - e. Install the required load modules in your ISPLLIB concatenation. These modules are in the thlqual.SCSQAUTH library with names beginning CSQOX.

4. Test that you can access the MQSeries panels from the TSO Command Processor panel. This is usually option 6 on the ISPF/PDF Primary Options Menu. The name of the EXEC that you run is CSQOREXX. There are no parameters to specify if you have installed the MQSeries libraries permanently in your ISPF setup as in step 3 on page 88. If you have not, use the following:

```
CSQOREXX thlqual langletter
```

where langletter is a letter identifying the national language to be used:

- C** Simplified Chinese
- E** U.S. English (mixed case)
- K** Japanese
- U** U.S. English (uppercase)

Updating the ISPF menu

You can update the ISPF main menu to allow access to the MQSeries operations and control panels from ISPF. The required setting for &ZSEL is:

```
CMD(%CSQOREXX thlqual langletter) NEWAPPL(CSQO) PASSLIB
```

For information about thlqual and langletter, see Step 4.

For more details, see the *ISPF Dialog Developer's Guide and Reference* manual.

Updating the function keys and command settings

You can use the normal ISPF procedures for changing the function keys and command settings used by the panels. The application identifier is CSQO.

However, this is *not* recommended because the help information is not updated to reflect any changes that you have made.

Task 17: Include the MQSeries dump formatting member

- *You need only perform this task once.*

To be able to format MQSeries dumps using the Interactive Problem Control System (IPCS), copy the data set thlqual.SCSQPROC(CSQ7IPCS) to SYS1.PARMLIB. You should not need to edit this data set.

Edit SYS1.PARMLIB member BLSCECTX and add this statement at the end of the member:

```
IMBED MEMBER(CSQ7IPCS) ENVIRONMENT(ALL)
```

(This member is described in the *MVS Initialization and Tuning Reference* manual.)

If you have customized the TSO procedure for IPCS, thlqual.SCSQPROC(CSQ7IPCS) can be copied into any library in the IPCSPARM definition. See the *MVS IPCS Customization* manual for details on IPCSPARM.

You must also include the library thlqual.SCSQPNLA in your ISPPLIB concatenation.

To make the dump formatting programs available to your TSO session or IPCS job, you must also include the library thlqual.SCSQAUTH in your STEPLIB concatenation. (Alternatively SCSQAUTH can be included in the OS/390 LINKLIST concatenation.)

Task 18: Suppress information messages

- *You need only perform this task once.*

If your MQSeries system is heavily used, with many channels stopping and starting, a large number of information messages are sent to the OS/390 console. The MQSeries-IMS bridge and buffer manager can also produce a large number of information messages.

If required, you can suppress some of these messages by using the OS/390 message processing facility list, specified by the MPFLSTxx members of SYS1.PARMLIB. The messages you specify still appear on the hard-copy log, but not on the console.

Sample thlqual.SCSQPROC(CSQ4MPFL) shows suggested settings for MPFLSTxx. See the *MVS Initialization and Tuning Reference* manual for more information about MPFLSTxx.

Suppress information messages

Chapter 4. Migrating from previous versions of MQSeries for MVS/ESA

This chapter describes the things that you must consider if you are migrating from a previous version of MQSeries for MVS/ESA. The following topics are discussed:

- “Migrating from Version 1.2 to Version 2.1”
- “Migrating from Version 1.1.4 to Version 2.1” on page 96
- “Migrating from Version 1.1.3 to Version 2.1” on page 96
- “Migrating from Version 1.1.2 or earlier to Version 2.1” on page 97
- “Coexistence with earlier versions of MQSeries for MVS/ESA” on page 98

You can continue to use your existing subsystems along with their page sets, log data sets, object definitions, and initialization input data sets with the new version. After installing the new version, you must IPL the system so that the new MQSeries early code is brought into use. **Once you have used the new version, it will not be possible to revert to a previous version.**

Migrating from Version 1.2 to Version 2.1

- The minimum levels for many of the items of software required to use MQSeries for OS/390 have changed (OS/390, CICS, and IMS in particular). Check that you have the correct levels for your prerequisite and corequisite software from the list in the *MQSeries Planning Guide*.
- Relink your system parameter module if you are not using the supplied default CSQZPARM (see “Task 13: Tailor your system parameter module” on page 67).

There are new system parameters (EXITLIM, EXITTCB, and WLMTIME) and channel initiator parameters (TCPTYPE, LU62ARM, and LSTRTMR). Consider whether you need to use these new parameters, and change your parameter modules again accordingly. See “Task 13: Tailor your system parameter module” on page 67 and “Task 13: Tailor your system parameter module” on page 67 for more information.

- There are several changes to the installation process, and some new libraries. The two distributed queuing features for the non-CICS mover have been incorporated into the base product, and the CICS mover has been made an optional feature. The CICS bridge has also been incorporated into the base product.

These are described in the *MQSeries for OS/390 Program Directory*.

- The OS/390 Automatic Restart Manager (ARM) is now supported. This support coexists on the same OS/390 image with earlier releases that do not support ARM. The queue managers and channel initiators in the earlier releases do not register with ARM and so can not be restarted automatically.

If you do not want to use ARM with your Version 2.1 queue managers and channel initiators, specify RESTART_ATTEMPTS(0) for the MQSeries element in your ARM policy. Note that if you do not specify MQSeries elements in your ARM policy, default ARM policies are used for MQSeries.

OS/390 ARM support is described in “Using the OS/390 Automatic Restart Manager (ARM)” on page 294.

Migrating from previous versions

- MQSeries now supports clustering. Before you use clustering you must review all of your applications to determine whether each one can operate in a clustering environment. You might have to modify your applications to remove or manage inter-message affinity. Applications that attempt to open non-existent queues might experience delays, or might even successfully open a queue somewhere in the cluster.

You also need to create the new system objects required for clustering. This is described in “CSQ4INSX system object sample” on page 56.

There is a cluster workload user exit; if you use this you need to add a CSQXLIB DD statement to your queue manager started task procedure, xxxxMSTR, and ensure that you have access to the LE run-time library SCEERUN.

Cluster support is described in the *MQSeries Queue Manager Clusters* manual.

- The supplied default for storage class SYSTEM (which was used by many of the SYSTEM queues) has been changed to page set 01, so that messages are not put on page set 00.

If you currently use the defaults supplied, this change will probably have no effect, even if you use the DEFINE REPLACE option for your storage class definitions in your initialization input data set. This is because some of the queues using that storage class (like the SYSTEM.CHANNEL.SYNCQ for example) have messages on them permanently. If you want to move the queues to another page set, follow the procedure given in “Load balancing by moving queues” on page 328.

- The sample input initialization data sets supplied with MQSeries have been reorganized and renamed. This is described in “Task 10: Customize the initialization input data sets” on page 52.
- You can migrate your existing batch/TSO MQSeries applications to exploit RRS coordination with little or no application program change. If you link-edit your MQSeries application with the CSQBRSI adapter, **MQCMIT** and **MQBACK** will synchronize your unit of work across MQSeries and all other RRS-enabled resource managers. If you link-edit your MQSeries application with the CSQBRSTB adapter, you must change **MQCMIT** and **MQBACK** to **SRRCMIT** and **SRRBACK**.

Version 2.1 continues to support the non-RRS managed batch adapter in addition to supporting the RRS managed adapter. Thus different versions of MQSeries queue managers can coexist on the same OS/390 image.

- OpenEdition sockets are now available for use as an alternative to IUCV. If you are using OS/390 Version 2.5 or later, and are using IBM TCP/IP for distributed queuing, IUCV is not available. You must set the TCPTYPE channel initiator parameter to OESOCKETS (as described in Table 11 on page 83). Using OpenEdition sockets, you do not need to restart the channel initiator if TCP/IP has to be restarted.
- Channel initiator user ID checking has been changed and some new facilities added. See “User IDs used by the channel initiator” on page 433 for details, and review your channel definitions to ensure that you are getting the security control you want.
- The channel initiator can now record error information in a data set instead of taking a dump. Add the CSQSNAP DD statement to your channel initiator started task procedure to support this.

- The IMS language interface module CSQ2LI00 is no longer supported. All IMS applications should use the IMS supplied DFSLI000 module.
- Support for the new euro currency symbol has been added to MQSeries. If you need to modify your applications to use this symbol, ensure that they use one of the coded character sets that include it. These are described in the *MQSeries Application Programming Reference* manual. If you need to change the coded character set used by your queue manager, use the CCSID parameter of the system parameter module. This is described in “Using CSQ6SYSP” on page 68.
- The size of queue objects has increased for Version 2.1 to allow for the new cluster attributes. MQSeries automatically updates each of these queue objects the first time that it is changed. This happens whether you are using clustering or not. However, due to the nature of space reclamation in MQSeries, the space used on page set zero might increase dramatically until all these queue objects have been updated.

To avoid this, you should run a job similar to that in Figure 7 that changes all of your queue objects, enabling MQSeries to update them all at the same time.

```
//STEP1 EXEC PGM=CSQUTIL,PARM='CSQ1'
//STEPLIB DD DISP=SHR,DSN=thlqua1.SCSQANLE
// DD DISP=SHR,DSN=thlqua1.SCSQAUTH
//OUTPUT1 DD DISP=OLD,DSN=MY.MQSERIES.COMMANDS(DEFS)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
COMMAND DDNAME(CMDINP) MAKEDEF(OUTPUT1)
/*
//CMDINP DD *
DISPLAY QUEUE(*) TYPE(QLOCAL) ALL
DISPLAY QUEUE(*) TYPE(QMODEL) ALL
/*
//* STEP2
//*****
/* PERFORM A GLOBAL CHANGE ON THE OUTPUT DATA SET FROM STEP 1, THAT *
/* IS: MY.MQSERIES.COMMANDS(DEFS). CHANGE 'NOREPLACE' TO 'REPLACE' *
/* THE CHANGED MY.MQSERIES.COMMANDS(DEFS) WILL BE THE INPUT FOR STEP3. *
//*****
/*
//STEP3 EXEC PGM=CSQUTIL,PARM='CSQ1'
//STEPLIB DD DISP=SHR,DSN=thlqua1.SCSQANLE
// DD DISP=SHR,DSN=thlqua1.SCSQAUTH
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
COMMAND DDNAME(DEFINES)
/*
//DEFINES DD DISP=SHR,DSN=MY.MQSERIES.COMMANDS(DEFS)
```

Figure 7. Example job for migrating queue objects

- The operations and control panels have been extended and reorganized to accommodate the new clustering function. The panels can now also be used for the security commands. A number of actions have been consolidated into a new PERFORM action. This is described in “Using the operations and control panels” on page 197.

Migrating from Version 1.1.4 to Version 2.1

If you intend to migrate from Version 1.1.4 to Version 2.1, you need to note the following when you customize your new version in addition to the tasks in the previous section (you don't need to install and customize the intervening versions):

- There is a new queue attribute, `INDXTYPE`, that allows the queue manager to expedite **MQGET** operations on the queue. To improve channel performance, you should set `INDXTYPE` to `MSGID` for the `SYSTEM.CHANNEL.SYNCQ` queue. In general, you will then need to restart the queue manager to have this take effect. See the *MQSeries Command Reference* manual for more information.
- There is a new channel attribute, `NPMSPEED`, that specifies a class of service for nonpersistent messages on the channel. **The class applied to existing channel definitions is FAST, which means that nonpersistent messages might be lost if there is a channel error.** If this is not acceptable, you must alter your channel definitions to have `NPMSPEED(NORMAL)`.

There are also two other new channel attributes, `HBINT` and `BATCHINT`. See the *MQSeries Command Reference* manual for more information.

- There are new libraries for distributed queuing: `thlqua1.SCSQMVR1` if you are using the OpenEdition sockets or IUCV interface and `thlqua1.SCSQMVR2` if you are using the TCPaccess interface. Add the appropriate library to the `STEPLIB` DD statement of the JCL used for your channel initiator started task procedures (`xxxxCHIN`) and data conversion utility (`CSQUCVX`).
- Use of the C/370 library product by MQSeries is no longer supported (although MQSeries applications can still use it). You must now use LE instead. Change your channel initiator started-task procedures (`xxxxCHIN`) and data conversion utility (`CSQUCVX`) JCL accordingly.
- TCP/IP Version 2 is no longer supported.

Migrating from Version 1.1.3 to Version 2.1

If you intend to migrate from Version 1.1.3 to Version 2.1, you need to do the following when you customize Version 2.1, in addition to the tasks in the previous sections (you don't need to install and customize the intervening versions).

Note: This information is included for your convenience, although this version of MQSeries for MVS/ESA is no longer supported.

- The connection names for channels can be up to 48 characters long. The data format in `SYSTEM.CHANNEL.SYNCQ` will be updated to accommodate this change automatically. Create new system channel objects `SYSTEM.DEF.SVRCONN` and `SYSTEM.DEF.CLNTCONN` as described in "Task 10: Customize the initialization input data sets" on page 52.
- Ensure that your `CSQINP2` initialization input data set has `DEFINE` commands for the storage classes `SYSTEMST`, `DEFAULT`, `REMOTE`, `SYSTEM` and `NODEFINE`, as in the supplied samples `thlqua1.SCSQPROC(CSQ4INSG)` and `thlqua1.SCSQPROC(CSQ4INYG)`; if not, you might get MQI calls failing with `MQRC_STORAGE_CLASS_ERROR`. Any storage classes that were being used by your queues will have non-empty storage class objects automatically created for them.

The NODEFINE storage class is no longer allocated automatically. It must be defined explicitly, as in the supplied sample thlqual.SCSQPROC(CSQ4INYG).

- Change the STEPLIB DD statements in all JCL used for started task procedures and utility jobs to reflect the new library organization. The MQSeries libraries required are now as follows:

Authorized	Distributed queuing	Non-authorized	CICS
SCSQANLx	SCSQANLx	SCSQANLx	SCSQANLx
SCSQAUTH	SCSQMVRy SCSQAUTH	SCSQLOAD	SCSQCICS

Where x is the language letter of your national language, and y is 1 if you are using the OpenEdition sockets or IUCV interface or 2 if you are using the TCPAccess interface.

- There are new system parameters in CSQZPARM (QMCCSID, OTMACON) and CSQXPARM (ACTCHL, TCPKEEP). If you do not use the supplied default parameter modules, consider whether you need to use these new parameters, and change and link-edit your parameter modules again accordingly.
- There is a new queue manager attribute, DEFXMITQ, which you might wish to use.
- For the CICS adapter, additional programs need to be defined to CICS/ESA®; update the CSD using the supplied sample data set thlqual.SCSQPROC(CSQ4B100).
- The channel initiator no longer uses the CSQSYNA data set; this can be deleted and removed from the channel initiator started task procedure.
- The channel initiator can now have an initialization input data set, CSQINPX (see “Initialization commands for distributed queuing” on page 53) and the channel initiator started task procedure can now have symbolic parameters.

Migrating from Version 1.1.2 or earlier to Version 2.1

If you intend to migrate from Version 1.1.2 to Version 2.1, you need to do the following when you customize Version 2.1, in addition to the tasks in the previous sections (you don’t need to install and customize the intervening versions).

Note: This information is included for your convenience, although this version of MQSeries for MVS/ESA is no longer supported.

- Create new system objects and queue attributes (see “Task 10: Customize the initialization input data sets” on page 52).
- If you want to use the dynamic page set expansion feature, re-allocate each of your page sets with secondary extents, and then use the COPYPAGE function of CSQUTIL to copy the old versions of the page sets to the new ones. (You can define up to 123 secondary extents for a page set, provided that there is sufficient disk storage space available.)

The supplied sample thlqual.SCSQPROC(CSQ4PAGE) shows how to define the secondary extents, and Chapter 14, “Using the MQSeries utilities” on page 227 gives information about using CSQUTIL.

Migrating from previous versions

- If you want to use both command prefix strings (CPFs) and subsystem recognition characters (SRCs) check the coexistence rules given under “Coexistence with earlier versions of MQSeries for MVS/ESA” on page 98.
- If you have programs that send commands to the SYSTEM.COMMAND.INPUT queue, note that command replies can be sent in several batches. (See “Interpreting the replies” on page 220 for more information.)

Coexistence with earlier versions of MQSeries for MVS/ESA

This topic discusses coexistence issues for the following:

- “Multiple queue manager versions”
- “Operations and control panels”
- “Application stubs”

Multiple queue manager versions

There can be several MQSeries subsystems in an OS/390 image, and they can use different versions of MQSeries, provided the MQSeries early code modules are of the latest version being used. (These modules are loaded at OS/390 IPL time and are shared among all the MQSeries subsystems the OS/390 image.) This means that you can run one queue manager with Version 2.1 and another in the same image with Version 1.2, provided that the early code is that of Version 2.1.

As explained in “Migrating from Version 1.2 to Version 2.1” on page 93, ARM support and RRS support allow such coexistence.

Operations and control panels

When using the operations and control panels, the MQSeries libraries you use in ISPF must match those of the queue manager you are working with. That is, the panels at the Version 2.1 level will work only with a Version 2.1 queue manager, and Version 1.1.4 panels will work only with a Version 1.1.4 queue manager, and so on.

Application stubs

The stub modules that are link-edited with applications and exits (CSQASTUB, CSQBRSSI, CSQBRSTB, CSQBSTUB, CSQCSTUB, CSQQSTUB, and CSQXSTUB) might not work with earlier versions of the queue manager. For example, stubs supplied with Version 1.2 can be used by applications running on a Version 1.2 or 2.1 queue manager; however, if the application is run on a Version 1.1.4 queue manager, it might not work, or might end abnormally.

Chapter 5. Testing your queue manager

When you have customized or migrated your queue manager, you can test it by running some of the sample applications shipped with MQSeries.

You can then compile and link-edit whichever of the other samples are appropriate to your installation using the sample JCL supplied.

This chapter tells you about:

- Basic function testing using an Assembler MQSeries application program in the OS/390 batch environment
- “Testing for C, C++, COBOL, PL/I, and CICS” on page 102
- “Testing for distributed queuing” on page 102

Basic function testing

Use the Batch Assembler installation verification program (IVP) to verify the base MQSeries without using C, COBOL, or CICS samples.

The Batch Assembler IVP is link-edited by SMP/E and the load modules are shipped in library thlqual.SCSQLOAD.

After you have completed both the SMP/E APPLY step and the customization steps, run the Batch Assembler IVP.

Running the installation verification program CSQ4IVP1

After you have installed and customized MQSeries, you can use the supplied installation verification program, CSQ4IVP1, to confirm that MQSeries is operational.

Overview of the CSQ4IVP1 job

CSQ4IVP1 is a batch job that connects to your MQSeries subsystem and performs these basic functions:

- Issues MQI calls
- Communicates with the command server
- Verifies triggering is active
- Generates and deletes a dynamic queue

Preparing to run CSQ4IVP1

Before you run CSQ4IVP1:

1. Check that the IVP entries are in the CSQINP2 data set concatenation in the MQSeries startup program. The IVP entries are supplied in member thlqual.SCSQPROC(CSQ4IVP). If not, add the definitions supplied in thlqual.SCSQPROC(CSQ4IVP) to your CSQINP2 concatenation. If MQSeries is currently running, you will need to restart it so that these definitions can take effect.
2. The sample JCL, CSQ4IVPR, required to run the installation verification program (CSQ4IVP1) is in library thlqual.SCSQPROC.

Testing customization

Customize the CSQ4IVPR JCL with the high-level qualifier for the MQSeries libraries, the national language you want to use, the four-character MQSeries subsystem name, and the destination for the job output.

3. Update RACF to allow CSQ4IVP1 to access its resources if MQSeries security is active.

To run CSQ4IVP1 when MQSeries security is enabled, you need a RACF user ID with authority to access the objects. For details of defining resources to RACF, see Part 8, "Security" on page 391. The user ID that runs the IVP must have:

- READ access to ssid.DISPLAY.PROCESS in class MQCMDS
- UPDATE access to ssid.SYSTEM.COMMAND.INPUT in class MQQUEUE
- UPDATE access to ssid.SYSTEM.COMMAND.REPLY.MODEL in class MQQUEUE
- UPDATE access to ssid.CSQ4IVP1.** in MQQUEUE
- READ access to ssid.BATCH in MQCONN

These requirements assume that all MQSeries security is active. The RACF commands to activate MQSeries security are shown in Figure 8. This example assumes that the MQSeries subsystem name is CSQ1 and that the user ID of the person running sample CSQ4IVP1 is TS101.

```
RDEFINE MQCMDS CSQ1.DISPLAY.PROCESS
PERMIT CSQ1.DISPLAY.PROCESS CLASS(MQCMDS) ID(TS101) ACCESS(READ)

RDEFINE MQQUEUE CSQ1.SYSTEM.COMMAND.INPUT
PERMIT CSQ1.SYSTEM.COMMAND.INPUT CLASS(MQQUEUE) ID(TS101) ACCESS(UPDATE)

RDEFINE MQQUEUE CSQ1.SYSTEM.COMMAND.REPLY.MODEL
PERMIT CSQ1.SYSTEM.COMMAND.REPLY.MODEL CLASS(MQQUEUE) ID(TS101) ACCESS(UPDATE)

RDEFINE MQQUEUE CSQ1.CSQ4IVP1.**
PERMIT CSQ1.CSQ4IVP1.** CLASS(MQQUEUE) ID(TS101) ACCESS(UPDATE)

RDEFINE MQCONN CSQ1.BATCH
PERMIT CSQ1.BATCH CLASS(MQCONN) ID(TS101) ACCESS(READ)
```

Figure 8. RACF commands for CSQ4IVP1

Running CSQ4IVP1

When you have completed these steps, start your MQSeries subsystem. If MQSeries is already running and you have made changes to CSQINP2, you must stop MQSeries and restart it.

The IVP runs as a batch job. Customize the job card to meet the submission requirements of your installation.

Checking the results of CSQ4IVP1

The IVP is split into eight stages; each stage must complete with a zero completion code before the next stage is run. The IVP generates a report, listing:

- The name of queue manager that is being connected to.
- A one-line message showing the completion code and the reason code returned from each stage.

For an explanation of the completion and reason codes, see the *MQSeries for OS/390 Messages and Codes* manual.

Some stages have more than one MQI call and, in the event of failure, a message is issued indicating the specific MQI call that returned the failure. Also, for some stages the IVP puts explanatory and diagnostic information into a comment field.

The IVP job requests exclusive control of certain queue manager objects and therefore should be single threaded through the system. However, there is no limit to the number of times the IVP can be run against your MQSeries subsystem.

The functions performed by each stage are:

Stage 1 Connect to the queue manager by issuing **MQCONN**.

Stage 2 Determine the name of the system-command input queue used by the command server to retrieve request messages. This queue receives display requests from Stage 5.

To do this, the sequence of calls is:

1. Issue an **MQOPEN**, specifying the queue manager name, to open the queue manager object.
2. Issue an **MQINQ** to find out the name of the system-command input queue.
3. Issue an **MQCLOSE** to close the queue manager object.

On successful completion of this stage, the name of the system-command input queue is displayed in the comment field.

Stage 3 Open an initiation queue using **MQOPEN**.

This queue is opened at this stage in anticipation of a trigger message, which arrives as a result of the command server replying to the request from Stage 5. The queue must be opened for input to meet the triggering criteria.

Stage 4 Create a permanent dynamic queue using the CSQ4IVP1.MODEL queue as a model. The dynamic queue has the same attributes as the model from which it was created. This means that when the replies from the command server request in Stage 5 are written to this queue, a trigger message is written to the initiation queue opened in Stage 3.

Upon successful completion of this stage, the name of the permanent dynamic queue is indicated in the comment field.

Stage 5 Issue an **MQPUT1** request to the command server command queue.

A message of MQMT_REQUEST is written to the system-command input queue requesting a display of process CSQ4IVP1. The message descriptor for the message specifies the permanent dynamic queue

Testing distributed queuing

created in Stage 4 as the reply-to queue for the command server's response.

Stage 6 Issue an **MQGET** request from the initiation queue. At this stage, a GET WAIT with an interval of one minute is issued against the initiation queue opened in Stage 3. The message returned is expected to be the trigger message generated by the command server's response messages being written to the reply-to queue.

Stage 7 Delete the permanent dynamic queue created in Stage 4. As the queue still has messages on it, the MQCO_PURGE_DELETE option is used.

Stage 8 Disconnect from the queue manager using **MQDISC**.

After running the IVP, you can delete any objects that you no longer require.

If the IVP does not run successfully, try each step manually to find out which function is failing.

Testing for C, C++, COBOL, PL/I, and CICS

You can test for C, C++, COBOL, PL/I, or CICS, using the sample applications supplied with MQSeries. Although the IVP (CSQ4IVP1) is supplied as a load module, the samples are supplied as source modules.

For more information about sample applications, see the *MQSeries Application Programming Reference* and *MQSeries Using C++* manuals.

Testing for distributed queuing

You can use the supplied installation verification program, CSQ4IVPX, to confirm that distributed queuing (without CICS) is operational.

Overview of CSQ4IVPX job

CSQ4IVPX is a batch job that starts the channel initiator and issues the DISPLAY DQM MQSC command. This verifies that all major aspects of distributed queuing are operational, while avoiding the need to set up channel and network definitions.

Preparing to run CSQ4IVPX

Before you run CSQ4IVPX:

1. The sample JCL, CSQ4IVPX, required to run the installation verification program is in library thlqual.SCSQPROC.

Customize the CSQ4IVPX JCL with the high-level qualifier for the MQSeries libraries, the national language you want to use, the four-character MQSeries subsystem name, and the destination for the job output. If you have a customized channel initiator parameter module, replace CSQXPARM with the name of your module.

2. Update RACF to allow CSQ4IVPX to access its resources if MQSeries security is active. To run CSQ4IVPX when MQSeries security is enabled, you need a RACF user ID with authority to access the objects. For details of defining resources to RACF, see Part 8, “Security” on page 391. The user ID that runs the IVP must have:

- READ access to ssid.DISPLAY.DQM in class MQCMDS
- CONTROL access to ssid.START.CHINIT and STOP.CHINIT in class MQCMDS
- UPDATE access to ssid.SYSTEM.COMMAND.INPUT in class MQQUEUE
- UPDATE access to ssid.SYSTEM.CSQUTIL.* in MQQUEUE
- READ access to ssid.BATCH in MQCONN

These requirements assume that the connection security profile ssid.CHIN has been defined (as shown in “Connection security profiles for distributed queuing” on page 405), and that all MQSeries security is active. The RACF commands to do this are shown in Figure 9 on page 104. This example assumes that:

- The MQSeries subsystem name is CSQ1
- The user ID of the person running sample CSQ4IVPX is TS101
- The channel initiator address space is running under the user ID CSQ1MSTR

3. Update RACF to allow the channel initiator address space the following RACF access:

- READ access to ssid.CHIN in class MQCONN
- UPDATE access to ssid.SYSTEM.COMMAND.INPUT in class MQQUEUE
- UPDATE access to ssid.SYSTEM.CHANNEL.INITQ in class MQQUEUE
- UPDATE access to ssid.SYSTEM.CHANNEL.SYNCQ in class MQQUEUE
- ALTER access to ssid.SYSTEM.CLUSTER.COMMAND.QUEUE in class MQQUEUE
- UPDATE access to ssid.SYSTEM.CLUSTER.TRANSMIT.QUEUE in class MQQUEUE
- ALTER access to ssid.SYSTEM.CLUSTER.REPOSITORY.QUEUE in class MQQUEUE
- CONTROL access to ssid.CONTEXT in class MQADMIN

The RACF commands to do this are also shown in Figure 9 on page 104.

Testing distributed queuing

```
RDEFINE MQCMDS CSQ1.DISPLAY.DQM
PERMIT CSQ1.DISPLAY.DQM CLASS(MQCMDS) ID(TS101) ACCESS(READ)

RDEFINE MQCMDS CSQ1.START.CHINIT
PERMIT CSQ1.START.CHINIT CLASS(MQCMDS) ID(TS101) ACCESS(CONTROL)

RDEFINE MQCMDS CSQ1.STOP.CHINIT
PERMIT CSQ1.STOP.CHINIT CLASS(MQCMDS) ID(TS101) ACCESS(CONTROL)

RDEFINE MQQUEUE CSQ1.SYSTEM.COMMAND.INPUT
PERMIT CSQ1.SYSTEM.COMMAND.INPUT CLASS(MQQUEUE) ID(TS101,CSQ1MSTR) ACCESS(UPDATE)

RDEFINE MQQUEUE CSQ1.SYSTEM.CSQUTIL.*
PERMIT CSQ1.SYSTEM.CSQUTIL.* CLASS(MQQUEUE) ID(TS101) ACCESS(UPDATE)

RDEFINE MQCONN CSQ1.BATCH
PERMIT CSQ1.BATCH CLASS(MQCONN) ID(TS101) ACCESS(READ)

RDEFINE MQCONN CSQ1.CHIN
PERMIT CSQ1.CHIN CLASS(MQCONN) ID(CSQ1MSTR) ACCESS(READ)

RDEFINE MQQUEUE CSQ1.SYSTEM.CHANNEL.SYNCQ
PERMIT CSQ1.SYSTEM.CHANNEL.SYNCQ CLASS(MQQUEUE) ID(CSQ1MSTR) ACCESS(UPDATE)

RDEFINE MQQUEUE CSQ1.SYSTEM.CLUSTER.COMMAND.QUEUE
PERMIT CSQ1.SYSTEM.CLUSTER.COMMAND.QUEUE CLASS(MQQUEUE) ID(CSQ1MSTR) ACCESS(ALTER)

RDEFINE MQQUEUE CSQ1.SYSTEM.CLUSTER.TRANSMIT.QUEUE
PERMIT CSQ1.SYSTEM.CLUSTER.TRANSMIT.QUEUE CLASS(MQQUEUE) ID(CSQ1MSTR) ACCESS(UPDATE)

RDEFINE MQQUEUE CSQ1.SYSTEM.CLUSTER.REPOSITORY.QUEUE
PERMIT CSQ1.SYSTEM.CLUSTER.REPOSITORY.QUEUE CLASS(MQQUEUE) ID(CSQ1MSTR) ACCESS(ALTER)

RDEFINE MQQUEUE CSQ1.SYSTEM.CHANNEL.INITQ
PERMIT CSQ1.SYSTEM.CHANNEL.INITQ CLASS(MQQUEUE) ID(CSQ1MSTR) ACCESS(UPDATE)

RDEFINE MQADMIN CSQ1.CONTEXT
PERMIT CSQ1.CONTEXT CLASS(MQADMIN) ID(CSQ1MSTR) ACCESS(CONTROL)
```

Figure 9. RACF commands for CSQ4IVPX

Running CSQ4IVPX

When you have completed these steps, start your MQSeries subsystem.

The IVP runs as a batch job. Customize the job card to meet the submission requirements of your installation.

Checking the results of CSQ4IVPX

CSQ4IVPX runs the CSQUTIL MQSeries utility to issue three MQSC commands. The SYSPRINT output data set should look like Figure 10 on page 105, although details might differ depending on your channel initiator parameters.

- You should see the commands **(1)** each followed by several messages.

- The last message from each command should be “CSQ9022I ... NORMAL COMPLETION” **(2)**. The job as a whole should complete with return code 0 **(3)**.

```

CSQU000I CSQUTIL IBM MQSeries for OS/390 - V2.1
CSQU001I CSQUTIL Queue Manager Utility - 1999-01-31 09:06:48
COMMAND
CSQU127I CSQUTIL Executing COMMAND using input from CSQUCMD data set
CSQU055I CSQUTIL Target queue manager is CSQ1
CSQU120I CSQUTIL Connecting to queue manager CSQ1
CSQU121I CSQUTIL Connected to queue manager CSQ1
START CHINIT PARM(CSQXPARM)
(1)
CSQN205I COUNT= 2, RETURN=00000000, REASON=00000004
CSQM138I +cpf CSQMSCHI CHANNEL INITIATOR STARTING
CSQN205I COUNT= 2, RETURN=00000000, REASON=00000000
CSQ9022I +cpf CSQXCRPS ' START CHINIT' NORMAL COMPLETION
(2)
DISPLAY DQM
(1)
CSQN205I COUNT= 2, RETURN=00000000, REASON=00000004
CSQM137I +cpf CSQMDDQM DISPLAY DQM COMMAND ACCEPTED
CSQN205I COUNT= 12, RETURN=00000000, REASON=00000000
CSQX830I +cpf CSQXRDM Channel initiator active
CSQX845I +cpf CSQXRDM TCP/IP address space name is TCPIP
CSQX848I +cpf CSQXRDM TCP/IP listener not started
CSQX849I +cpf CSQXRDM LU6.2 listener not started
CSQX832I +cpf CSQXRDM 5 dispatchers started, 5 requested
CSQX831I +cpf CSQXRDM 8 adapter subtasks started, 8 requested
CSQX840I +cpf CSQXRDM 0 channel connections current, maximum 200
CSQX841I +cpf CSQXRDM 0 channel connections active, maximum 200
CSQX842I +cpf CSQXRDM 0 channel connections starting,
0 stopped, 0 retrying
CSQ9022I +cpf CSQXCRPS ' DISPLAY DQM' NORMAL COMPLETION
(2)
STOP CHINIT
(1)
CSQN205I COUNT= 2, RETURN=00000000, REASON=00000004
CSQM137I +cpf CSQMTCHI STOP CHINIT COMMAND ACCEPTED
CSQN205I COUNT= 2, RETURN=00000000, REASON=00000000
CSQ9022I +cpf CSQXCRPS ' STOP CHINIT' NORMAL COMPLETION
(2)
CSQU057I CSQUCMDS 3 commands read
CSQU058I CSQUCMDS 3 commands issued and responses received
CSQU143I CSQUTIL 1 COMMAND statements attempted
CSQU144I CSQUTIL 1 COMMAND statements executed successfully
CSQU148I CSQUTIL Utility completed, return code=0
(3)

```

Figure 10. Example output from CSQ4IVPX

Testing distributed queuing

Part 3. MQSeries and CICS

Chapter 6. The MQSeries-CICS adapter	109
CICS adapter overview	109
Control functions	109
MQI support	110
Adapter components	111
Other adapter features	112
Alert monitor	112
Auto-reconnect	113
Task initiator	113
Multi-tasking	114
The API-crossing exit	114
CICS adapter performance considerations	115
CICS adapter conventions	117
Temporary storage queue names	117
MQGET	117
ENQUEUE names	117
Setting up the CICS adapter	118
Resource definition	118
System definition	120
Completing the connection from CICS	121
Controlling CICS application connections	122
Customizing the CICS adapter	122
Writing a PLTPI program to start the connection	122
Chapter 7. Operating the CICS adapter	123
Invoking the adapter's control functions	123
From the CICS adapter control panels	123
From the CICS command line	123
From CICS application programs	124
Preparing to use the CICS adapter	125
Accessing the CICS adapter control panels	126
Starting a connection	127
Starting a connection from the CICS adapter control panels	127
Starting a connection from the CICS command line	128
Starting a connection from a CICS application program	129
Stopping a connection	130
Stopping a connection from the CICS adapter control panels	130
Stopping a connection from the CICS command line	130
Stopping a connection from a CICS application program	131
Modifying a connection	132
Modifying a connection from the CICS adapter control panels	132
Modifying a connection from the CICS command line	133
Modifying a connection from a CICS application program	133
Displaying details of connections and CICS tasks	135
Displaying details of a connection from the CICS adapter control panels	135
Starting an instance of the task initiator CKTI	136
Starting CKTI from the CICS adapter control panels	136
Starting CKTI from the CICS command line	137
Starting CKTI from a CICS application program	137
Stopping an instance of CKTI	138

Stopping an instance of CKTI from the CICS adapter control panels	138
Stopping an instance of CKTI from the command line	138
Stopping an instance of CKTI from an application program	139
Displaying the current instances of CKTI	140
Displaying the current instances of CKTI from the CICS adapter control panels	140
Displaying CICS task information	141
Displaying CICS tasks from the CICS adapter control panels	141
Displaying connection status and in-flight tasks	142
Purging tasks that are using the CICS adapter	143
Shutting down a connection between MQSeries and the CICS adapter	144
Orderly shutdown	144
Forced shutdown	145
Chapter 8. The MQSeries-CICS bridge	147
Introduction to the CICS bridge	147
When to use the CICS bridge	147
Running CICS DPL programs	148
Running CICS 3270 transactions	149
Customizing the CICS bridge	152
Starting the CICS bridge	154
Tuning considerations	154
Shutting down the CICS bridge	155

Chapter 6. The MQSeries-CICS adapter

This chapter describes the MQSeries-CICS adapter (generally referred to as the CICS adapter or, where the context permits, simply as the adapter). It contains these sections:

- “CICS adapter overview”
- “Adapter components” on page 111
- “Other adapter features” on page 112
- “CICS adapter performance considerations” on page 115
- “CICS adapter conventions” on page 117
- “Setting up the CICS adapter” on page 118
- “Customizing the CICS adapter” on page 122

CICS adapter overview

The CICS adapter connects a CICS subsystem to an MQSeries subsystem, enabling CICS application programs to use the MQI.

The CICS adapter provides two main facilities:

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

The CICS adapter is supplied with MQSeries as the CICS transaction CKQC. Through this transaction, you can control the MQSeries-supplied task initiator transaction CKTI², which is described on page 113.

Control functions

The CICS adapter’s control functions let you manage the connections between CICS and MQSeries dynamically. These functions can be invoked using the CICS adapter panels, from the command line, or from a CICS application. You can use the adapter’s control function to:

- Start a connection to a queue manager.
- Stop the connection.
- Modify the current connection. For example, you can reset the connection statistics, change the adapter’s trace ID number, and enable or disable the API-crossing exit.
- Display the current status of a connection and the statistics associated with that connection.
- Start an instance of the task initiator transaction, CKTI.
- Stop an instance of CKTI.
- Display details of the current instances of CKTI.

² This is CICS terminology. In MQSeries terminology, this is a trigger monitor process. It must have its own process definition—generated by issuing the MQSeries command, DEFINE PROCESS.

CICS adapter overview

- Display details of the CICS tasks currently using the adapter.

These functions and the different methods of invoking them are described in Chapter 7, “Operating the CICS adapter” on page 123.

MQI support

The CICS adapter implements the MQI for use by CICS application programs. The MQI calls, and how they are used, are described in the *MQSeries Application Programming Guide*. The adapter also supports an *API-crossing exit*, see “The API-crossing exit” on page 114, and a trace facility.

For performance, the CICS adapter can handle up to eight MQI calls concurrently. For transaction integrity, the adapter fully supports syncpointing under the control of the CICS syncpoint manager, so that units of work can be committed or backed out as required. The adapter also supports security checking of MQSeries resources when used with an appropriate security management product, such as RACF. The adapter provides high availability with automatic reconnection after an MQSeries termination, and automatic resource resynchronization after a restart. It also features an alert monitor that responds to unscheduled events such as a shut down of the MQSeries subsystem.

Note: If the CICS adapter detects an unexpected MQSeries subsystem error, it issues an OS/390 SNAP dump to DDname CSQSNAP, and issues reason code MQRC_UNEXPECTED_ERROR to the application.

If this occurs, rerun the application with a CSQSNAP DD statement included in the CICS JCL, and contact your IBM support center.

Adapter components

Figure 11 shows the relationship between CICS, the CICS adapter, and an MQSeries subsystem. CICS and the adapter share the same address space; the MQSeries for OS/390 is a separate OS/390 subsystem, executing in its own address space.

Part of the adapter is a CICS task-related user exit that communicates with the MQSeries message manager. CICS management modules call the exit directly; application programs call it through a supplied *API stub program* called CSQCSTUB. Task-related user exits and stub programs are described in the *CICS Customization Guide*.

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

The adapter management interface provides the operation and control functions described in Chapter 7, “Operating the CICS adapter” on page 123.

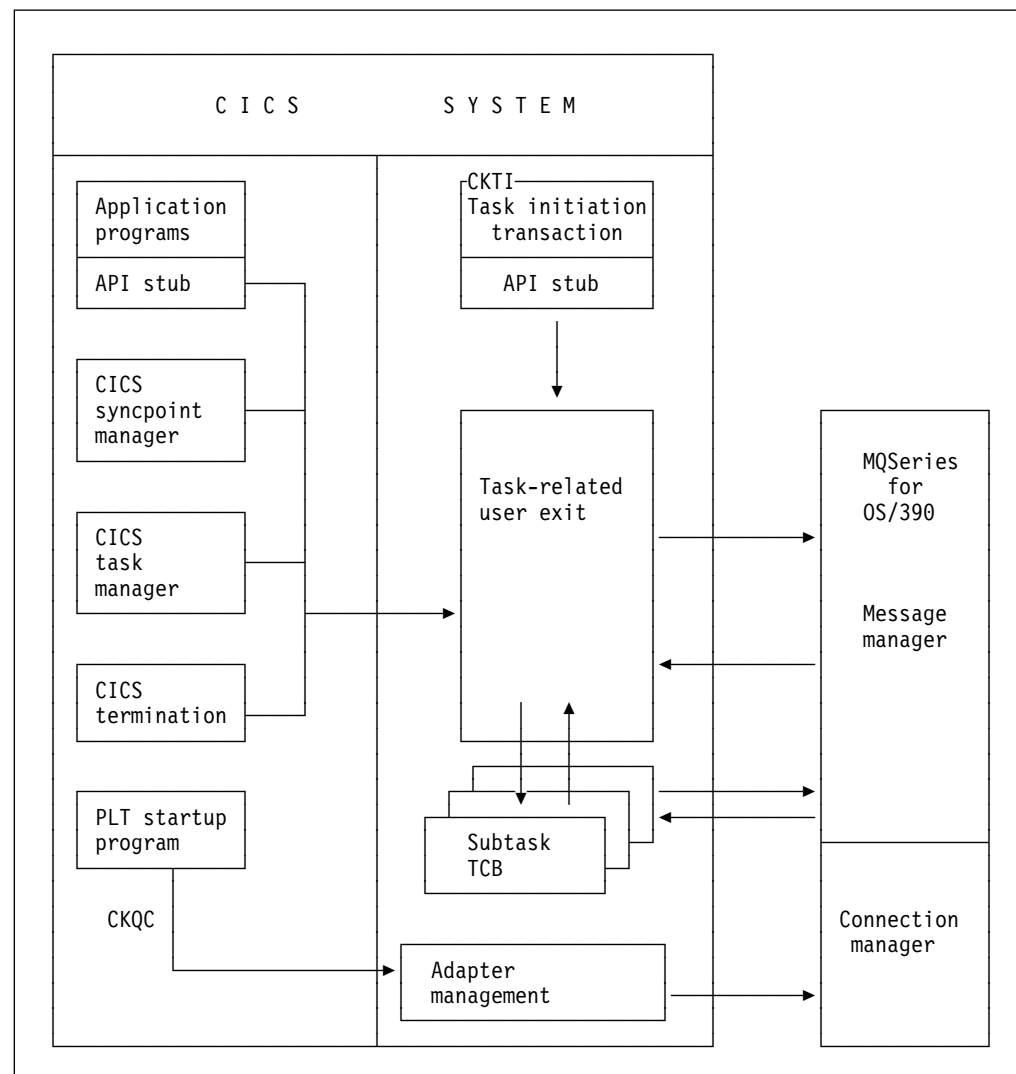


Figure 11. How CICS, the CICS adapter, and an MQSeries subsystem are related

Other adapter features

The CICS adapter incorporates several other features that you should be aware of:

- Alert monitor
- Auto-reconnect
- Task initiator
- Multi-tasking
- API-crossing exit

Alert monitor

The *alert monitor* transaction, CKAM, handles unscheduled events—known as *pending events*—that occur as a result of connect requests to instances of MQSeries. The alert monitor generates messages that are sent to the system console.

There are two kinds of pending events:

1. Deferred connection

If CICS tries to connect to MQSeries before MQSeries is started, a *pending event* called a *deferred connection* is activated. When MQSeries is started, a connection request is issued by the CICS adapter, a connection is made, and the pending event is canceled.

There can be multiple deferred connections, one of which will be connected when MQSeries is started. If there is more than one instance of MQSeries, which deferred connection is made to which instance of MQSeries is unpredictable.

2. Termination notification

When a connection is successfully made to MQSeries, a pending event called *termination notification* is created. This pending event expires when:

- MQSeries shuts down normally with MODE(QUIESCE). The alert monitor issues a quiesce request on the connection.
- MQSeries shuts down with MODE(FORCE) or terminates abnormally. After an abnormal termination, the CICS adapter waits for ten seconds and then tries a connect call. This enables the CICS system to be automatically reconnected to the queue manager when the latter is restarted.
- The connection is shut down from the CKQC transaction.

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

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

If the alert monitor has been inadvertently force-purged (this is not recommended) you must first disable the task-related user exit before attempting to start a new connection. The command to do this is:

```
CECI DISABLE PROGRAM(CSQCTRUE) ENTRYNAME(MQM) EXITALL STOP
```

Auto-reconnect

When CICS is connected to MQSeries and MQSeries terminates, the CICS adapter tries to issue a connect request ten seconds after the stoppage has been detected. This request uses the same connect parameters that were used in the previous connect request. If MQSeries has not been restarted within the ten seconds, the connect request is deferred until MQSeries is restarted later.

Task initiator

CKTI is an MQSeries-supplied CICS transaction that starts a CICS transaction when an MQSeries event occurs, for example when a message is put onto a specific queue.

How it works:

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

Each copy of CKTI services a single initiation queue. To start or stop a copy of CKTI, you must supply the name of the queue that this CKTI is to serve, or is serving. You cannot start more than one instance of CKTI against the same initiation queue from a single CICS subsystem.

At CICS system initialization or at connect time, you can define a default initiation queue. This is described in “System definition” on page 120 and “Starting a connection” on page 127. If you issue a STARTCKTI or a STOPCKTI without specifying an initiation queue, these commands are automatically interpreted as referring to the default initiation queue.

Notes:

1. If you are using version 4.1 of CICS, any transaction entries processed by CKTI, for example EXEC CICS START, are locked by the CKTI task until it terminates. Any attempt to CEDA INSTALL such entries after altering them will fail: CEDA rejects the install request because the transaction entry is being used by another task.

In this situation, you must stop the CKTI task using the CICS adapter, see “Stopping an instance of CKTI” on page 138, and restart it after the CEDA install, see “Starting an instance of the task initiator CKTI” on page 136.

2. This restriction also applies to intersystem connection (ISC) and multi-region operation (MRO) links. For example, if CKTI has started a remote transaction, a connection cannot be reinstalled until CKTI has been stopped.

Multi-tasking

The CICS adapter optimizes the performance of a CICS to MQSeries connection by exploiting multi-processors and by removing work from the main CICS task control block (TCB), allowing multiple MQI calls to be handled concurrently.

The adapter enables some MQI calls to be executed under subtasks, rather than under the main CICS TCB that runs the application code. All the CICS adapter administration code, including connection and disconnection from MQSeries, runs under the main CICS TCB.

The adapter tries to attach up to eight OS/390 subtasks (TCBs) to be used by this CICS system. *You cannot modify this number.* Each subtask makes a connect call to MQSeries. Each CICS system connected takes up nine of the connections specified on CTHREAD. This means that you must increase the value specified for CTHREAD in CSQ6SYSP by nine for each CICS system connected. For more details, see “Using CSQ6SYSP” on page 68. MQI calls can flow over those connections. When the main connection is terminated, the subtasks are disconnected and terminated automatically.

The API-crossing exit

MQSeries provides an API-crossing exit for use with the CICS adapter; it runs in the CICS address space. You can use this exit to intercept MQI calls as they are being run, for monitoring, testing, maintenance, or security purposes.

For more information about writing API-crossing exit programs, see the *MQSeries Application Programming Guide*.

Note: Using the API-crossing exit degrades MQSeries performance. You should plan your use of it carefully.

Defining the exit program

Before the API-crossing exit can be used, an exit program load module must be available when the CICS adapter connects to MQSeries. The exit program is a CICS program that must be named CSQCAPX and reside in a library in the DFHRPL concatenation. CSQCAPX must be defined in the CICS system definition file (CSD) and must be enabled.

When CSQCAPX is loaded a confirmation message is written to the CICS adapter control panel, CKQC, or the console. If it cannot be loaded, a diagnostic message is displayed, but otherwise the application program runs normally.

CICS adapter performance considerations

This section describes how the CICS adapter optimizes the performance of a CICS to MQSeries connection, and how you can monitor the connection.

There are a number of factors to be taken into consideration when performance is critical:

First MQI call

In general, the first MQI call of a task takes longer to perform than subsequent calls. This is because the environment must be set up. For example, the adapter must acquire storage and security information, and control blocks must be allocated and formatted.

MQGET and the SIGNAL option

Using the SIGNAL option with an **MQGET** call imposes an additional overhead. This is because the SIGNAL option can produce a CICS GETMAIN in the adapter, which is used to record the address of the ECB so that it can be posted if the queue manager abends.

API-crossing exit

Using the API-crossing exit also imposes a host processor overhead on each MQI call. The overhead in handling the exit parameter block and the invocations are minimal, but the exit can be invoked twice for each MQI call through EXEC CICS LINK.

CICS tracing

CICS tracing in the adapter also increases the pathlength of an MQI call. A large number of trace entries can be generated depending on how busy the system is. There is no control over the granularity of the trace entries produced in the adapter. Therefore, tracing should only be switched on if necessary.

MQGET and the WAIT option

Using **MQGET** with the WAIT option is less efficient if the task has been put into a wait until a message arrives. The adapter implements the wait as a form of CICS wait. When a message arrives, the adapter effectively re-issues the **MQGET** call for the application.

Therefore, use the WAIT option with care and only when the queue is known to be not very busy.

MQCLOSE

Issuing an **MQCLOSE** call is not always necessary because MQSeries automatically closes any unclosed handles when the task ends.

MQPUT1

If there is only one **MQPUT**, it is more efficient than an **MQOPEN-MQPUT-MQCLOSE** sequence because only one flow is generated between the MQSeries and the adapter, instead of three.

If there are multiple messages to be put, **MQOPEN-MQPUT...MQPUT-MQCLOSE** should be used.

EXEC CICS RETURN

Implicit syncpointing generated by EXEC CICS RETURN is more efficient than issuing the explicit syncpoint call EXEC CICS SYNCPOINT followed by EXEC CICS RETURN.

The EXEC CICS RETURN call accommodates all the work needed for syncpointing and task termination into one flow to MQSeries instead of the two separate flows used when explicit syncpointing is used.

Two-phase commit

A two-phase commit consumes more resources than a single-phase commit, both in host processor cost and response time. This is because a two-phase commit involves one more flow to MQSeries and more physical logging. If an application is restricted to recoverable updates in MQSeries and no other resource managers, CICS invokes the adapter for a single-phase commit.

Syncpoint bypassing

The adapter does not use the read-only commit feature in CICS. When a transaction is restricted to non-recoverable or non-destructive work in MQSeries, syncpointing is bypassed because it is not necessary. The clean-up process is performed when the task ends.

Statistics collection

Statistics collection by connection and by task is done on a per MQI call basis and cannot be switched off. This overhead is negligible.

You can use the CKQC transaction to display statistics for the current connection. See “Displaying details of connections and CICS tasks” on page 135.

CICS adapter conventions

There are a number of conventions that must be observed in applications using the adapter.

Temporary storage queue names

The CICS adapter display function uses two temporary storage queues (MAIN) per invoking task to store the output data for browsing. The names of the queues are **ttttCKRT** and **ttttCKDP**, where **tttt** is the terminal identifier of the terminal from which the display function is requested.

Do not try to access these queues.

MQGET

When the CICS adapter puts a task on a CICS wait because the WAIT option was used with the **MQGET** call and there was no message available, the RESOURCE NAME used is GETWAIT and the RESOURCE_TYPE is MQSeries.

When the CICS adapter puts a task on a CICS wait because of a need to perform task switching the RESOURCE NAME used is TASKSWCH and the RESOURCE_TYPE is MQSeries.

ENQUEUE names

The CICS adapter uses the name:

`CSQ.genericapplid(8).QMGR`

to issue CICS ENQ and CICS DEQ calls during processing, for example, starting and stopping the connection.

Attempts to use similar names for CICS ENQ or DEQ purposes should be avoided.

Setting up the CICS adapter

This section tells you how to make the MQSeries-CICS adapter (generally referred to in this book as the CICS adapter) available to your CICS subsystem. If you are not familiar with defining resources to CICS, refer to:

- The *CICS System Definition Guide* for general information on setting up a CICS subsystem.
- The *CICS Resource Definition Guide*, for background information on defining resources to CICS, details of and the command syntax of the CEDA transaction, and the MIGRATE command.
- The *CICS Operations and Utilities Guide* and the *CICS Resource Definition Guide* for details of the CSD utility program (DFHCSDUP).

Resource definition

This section takes you through the steps you must perform to define the resources for the CICS adapter.

Updating the CSD

This section describes the updates required for the CICS system definition (CSD) data set for the CICS adapter. It also describes the CSD updates required for the distributed queuing facility (if you want to use the “CICS mover”) and the CICS sample application programs. However, it does not contain all the information required to complete these tasks. If you are implementing distributed queuing, see Appendix D, “Enabling distributed queuing using CICS ISC” on page 479. If you intend to use the CICS sample application programs, see the *MQSeries Application Programming Guide*.

You must use resource definition online (RDO) to add new groups to the CSD data set. The new groups must contain definitions of:

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

To update the CSD, run the CICS offline utility program, DFHCSDUP, with the supplied sample input data sets:

- thlqual.SCSQPROC(CSQ4B100)
- thlqual.SCSQPROC(CSQ4D100)
- thlqual.SCSQPROC(CSQ4S100)

Where:

This data set...	Provides the definitions required for...
CSQ4B100	CICS adapter
CSQ4D100	Distributed queuing using CICS ISC (this is optional)
CSQ4S100	Supplied samples

Each of these data sets contains sample CICS definitions that must be tailored. To preserve the originals, copy these data sets into a user JCL library whose name

contains the MQSeries subsystem name, for example, MQS.CSQ1.USERJCL, and tailor them there.

Note: With some versions of CICS, you might receive warning messages about obsolete keywords; you can ignore these.

Ensure that any user-written CICS applications that issue MQI calls, and the resources they use, are also defined to the CSD. You can edit the input data set, to include definitions of user-programs and their resources.

You can add this fragment of JCL to your CSD upgrade (DFHCSDUP) job to define the MQSeries supplied groups to the CICS CSD:

```
//SYSIN DD DSN=thlqua1.SCSQPROC(CSQ4B100),DISP=SHR
//      DD DSN=thlqua1.SCSQPROC(CSQ4D100),DISP=SHR
//      DD DSN=thlqua1.SCSQPROC(CSQ4S100),DISP=SHR
//      DD *
//      ADD GROUP(CSQCAT1) LIST(yourlist)
//      ADD GROUP(CSQKDQ1) LIST(yourlist)
//      ADD GROUP(CSQ4SAMP) LIST(yourlist)
/*
```

Figure 12. JCL fragment for upgrading the CICS CSD

Here, `yourlist` is the name of a CICS list that contains a list of groups to be installed by CICS during a cold start of the system. This is specified in the `GRPLIST` parameter of your CICS system initialization table (SIT). For details of CICS SIT parameters, see the *CICS System Definition Guide*.

Include the new resource groups in the CICS startup group list. For information about resource groups, installing them in CICS, the CICS CSD, and DFHCSDUP, see the *CICS Resource Definition Guide*.

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

Starting a connection automatically during CICS initialization

If you want the adapter to connect to MQSeries automatically during CICS initialization, the `CSQCCODF` program should be included in a CICS PLTPI program. `CSQCCODF` must execute during the third stage of CICS initialization and must therefore be added after the entry for `DFHDELIM`. If there is no entry for `DFHDELIM` in your current PLTPI, you must add one.

Alternatively, if your version of CICS supports it, you can use the `MQCONN` SIT parameter to connect to MQSeries automatically. See the *CICS System Definition Guide* for information about this parameter.

Instead of using `CSQCCODF`, you can write your own program; see “Writing a PLTPI program to start the connection” on page 122.

1. Use the CICS `DFHPLT` macro to add your program to the list of programs executed by CICS during the third stage initialization. Figure 13 on page 120 shows how to code the entry for `CSQCCODF` in a CICS PLT program called `DFHPLT41`. For information about coding PLT entries, see the *CICS Resource Definition Guide*.

Setting up CICS adapter

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

Figure 13. Sample PLT for use with the CICS adapter. This sample assumes that you are using the supplied PLTPI program, CSQCCODF, to start the adapter.

2. Specify the particular list of programs to be run at initialization by naming the suffix of your PLT on the PLTPI system initialization parameter. In Figure 13, the PLT suffix is 41.

Note: You can use the CICS adapter in a CICS system that has interregion communication (IRC) to remote CICS systems. If you are using IRC, you should ensure that the IRC facility is OPEN before you start the adapter. This is essential if the IRC access method is defined as cross memory, that is, ACCESSMETHOD(XM).

System definition

Use the INITPARM parameter in the CICS system initialization table (SIT), or the SYSIN override, to set the default connection parameters. Figure 14 shows you how to do this.

```
INITPARM=(CSQCPARM='SN=CSQ1,TN=001,IQ=CICS01.INITQ')
```

Figure 14. Sample INITPARM statement to set the default connection values for CICS

Where:

- SN The subsystem name.
- TN The trace number to identify the adapter in CICS trace entries. This must be in the range 0 through 199.
- IQ The name of the default initiation queue. If this is blank, and you do not specify an initiation queue name by any other method, an instance of CKTI is not started when the CICS adapter connects to the queue manager.

The INITPARM statement does not accept a parameter string longer than 60 characters. If you specify a 4-character subsystem name and a 3-character trace number, the maximum allowable length of the initiation queue name is 42 characters. If you need a queue name longer than 42 characters, you cannot use the INITPARM statement to specify the default initiation queue.

At connect time, you must override the INITPARM setting, either by using the CKQC transaction, or in a PLTPI program.

1. If you are using a PLTPI program to start the adapter, code the suffix of your PLT on the PLTPI system initialization parameter. See Figure 13 for an example of this.
2. Add the sample DCT entries (CSQ4DCT1 and CSQ4DCT2) to those of the existing CICS DCT and then reassemble the DCT.

3. Add the following MQSeries libraries to the STEPLIB concatenation in your CICS procedure in this order:

- thlqual.SCSQANLx
- thlqual.SCSQAUTH

Where x is the language letter for your national language.

4. Add the following MQSeries libraries to the DFHRPL concatenation in your CICS procedure in this order:

- thlqual.SCSQANLx
- thlqual.SCSQCICS
- thlqual.SCSQAUTH

Where x is the language letter for your national language.

If you are using any CICS programs that dynamically call the MQSeries CICS stub, CSQCSTUB, also add thlqual.SCSQLOAD to the DFHRPL concatenation.

5. Update CSQINP2. You can use the sample CSQ4INP2, but you might need to change the initiation queue name to match your system definition.

For more information about:

- The CICS initiation queue, see “Task initiator” on page 113.
- The CKQC transaction, see “Starting a connection” on page 127.
- PLTPI programs, see “Writing a PLTPI program to start the connection” on page 122.
- Coding CICS system initialization parameters, see the *CICS System Definition Guide*.

Completing the connection from CICS

The connection is completed when the CICS adapter completes these steps:

1. Enable the CICS adapter and initialize the control blocks.
2. Attach the OS/390 subtasks and identify CICS generic *applid* (as specified in the CICS system initialization parameters as the connection ID) to MQSeries. This is described in the *CICS System Definition Guide*.

These two steps are done for you automatically if you use the INITPARM parameter or the CKQC transaction (see Chapter 7, “Operating the CICS adapter” on page 123). You can also use a PLTPI program to do this; see “Writing a PLTPI program to start the connection” on page 122.

When the connection is complete, a pending event called a *termination notification* is activated. This pending event remains active until MQSeries terminates in either an orderly or a forced way. When the pending event expires (or matures), it causes a FORCE shutdown request to be issued to the CICS adapter, and the pending event is canceled.

Controlling CICS application connections

Every CICS transaction that issues calls to MQSeries is assigned a unique thread ID to service the requests and keep track of changes made to MQSeries resources. The thread ID is created the first time a transaction issues an MQSeries request, and accompanies all subsequent MQSeries requests made by that transaction.

While executing work under the CICS main task TCB, the CICS adapter queues MQSeries requests for processing by any of the eight subtask TCBs. These subtask TCBs are attached by the adapter when the connection to MQSeries is established.

Customizing the CICS adapter

You can customize the CICS adapter by:

- Writing a user version of CSQCCODF that can be included in a CICS PLTPI program. See “Writing a PLTPI program to start the connection” for more information.
- Writing an API-crossing exit program. See “The API-crossing exit” on page 114 for more information.

Writing a PLTPI program to start the connection

You can write your own PLTPI program, based on the supplied assembler sample `thlqual.SCSQASMS(CSQCSPLT)`.

Although this sample is written in assembler, you can write your own program in any language supported by CICS. A typical use of PLTPI programs is for overriding the INITPARM settings if your CICS adapter initiation queue name is too long. (You cannot use more than 42 characters for an initiation queue name in an INITPARM statement.) If your PLTPI program gets its input parameters from a data set, you do not need an INITPARM statement.

Your PLTPI program must link to the adapter connect program, `thlqual.SCSQCIC(CSQCQCON)`, and pass a parameter list that specifies the connection values to be used. The parameter list is described in “Starting a connection from a CICS application program” on page 129. Figure 15 shows the LINK command that your PLTPI program must issue. In this example, the parameter list is named CONNPL. Because no terminals are available at this stage of CICS start up, you must use the COMMAREA option to pass the parameter list.

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

Figure 15. Linking to the adapter connect program, CSQCQCON, from a PLT program. The COMMAREA option is used, because no terminals are currently available.

For more information about writing CICS PLTPI programs, see the *CICS Customization Guide*.

Chapter 7. Operating the CICS adapter

This chapter describes how you can use the CICS adapter control functions to initiate and manage connections between MQSeries and CICS. It describes these tasks:

- “Invoking the adapter's control functions”
- “Preparing to use the CICS adapter” on page 125
- “Accessing the CICS adapter control panels” on page 126
- “Starting a connection” on page 127
- “Stopping a connection” on page 130
- “Modifying a connection” on page 132
- “Displaying details of connections and CICS tasks” on page 135
- “Starting an instance of the task initiator CKTI” on page 136
- “Stopping an instance of CKTI” on page 138
- “Displaying the current instances of CKTI” on page 140
- “Displaying CICS task information” on page 141
- “Purging tasks that are using the CICS adapter” on page 143
- “Shutting down a connection between MQSeries and the CICS adapter” on page 144

In this chapter, the MQSeries CICS adapter control panels are referred to as the CICS adapter control panels.

Invoking the adapter's control functions

You can invoke the control functions of the CICS adapter in three different ways:

1. From the CICS adapter control panels.
2. From the CICS command line.
3. From an application program.

From the CICS adapter control panels

You can use these CICS adapter controls panels to monitor and control connections between MQSeries and CICS.

From the initial panel, you first select an item from the menu bar at the top of the panel, and then select an action from one of the pull-down menus. In the displayed panel or secondary window, you can then type new values in the fields, as required.

From the CICS command line

You can take a “fast-path” approach and bypass the CICS adapter control panels, by specifying command line parameters on the CKQC transaction. The syntax of these command parameters, and examples of them, are given for each of the tasks described later in this chapter.

Note: You can also issue these commands from the console using OS/390 commands.

Commands take this form:

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

From CICS application programs

You can use the EXEC CICS LINK command to invoke most adapter control functions from CICS application programs. The syntax of the EXEC CICS LINK commands you need, and examples, are given for each of the tasks described later in this chapter.

Command syntax in application programs

Some commands issued in this way must be padded with trailing spaces to make the length of the command 10 characters. When an argument follows the command, an extra space character must be added as a separator. See Figure 16. The commands affected by this restriction and the number of trailing spaces required for each command are:

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

With all other commands the padding is optional.

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

Figure 16. Padding adapter commands. The MODIFY command must be padded with 4 trailing spaces plus another space as a separator. Starting at the 'M' in MODIFY, the argument 'Y' is the twelfth character.

Note: This restriction applies only to commands issued from an application program; it does not apply to commands issued from the command line.

Passing parameters from a CICS transaction

Use the following rules to determine how to pass the parameters:

- The CICS transaction must be running on an attached terminal. If it is not, all MQSeries commands are ignored.
- If a CICS application program on an attached terminal is connected to MQSeries, you must use the INPUTMSG option with EXEC CICS LINK to pass parameters, except at PLTPI time.
- If you connect to MQSeries at PLTPI time, you must use the COMMAREA option to pass parameters. If you use the INPUTMSG option, the command is ignored.

However, the adapter STOP commands:

```
CKQC STOP
CKQC STOP FORCE
```

cannot be run at PLTPI time, regardless of whether you use the INPUTMSG option or the COMMAREA option.

EXEC CICS LINK interface messages

If you invoke the adapter operation functions START and STOP from an application program using EXEC CICS LINK, the resultant messages are written to both the system console and a transient data queue (TDQ) named CKQQ. When the application program returns from the LINK, it can read back the messages by repeating EXEC CICS READQ TD QUEUE(CKQQ) until the queue is empty. The following restrictions apply:

- The TDQ queue name is CKQQ and cannot be changed. A sample TDQ definition is provided (in CSQ4DCT2), which defines CKQQ as an intra-partition TDQ.
- The queue is not cleared before it is written to.
- The messages are not time-stamped.
- If you have more than one application writing to the TDQ, the messages are not serialized. It is the responsibility of the invoking programs to serialize themselves.
- The same set of messages also appear on the system console.
- The server subtask messages are not written to CKQQ.

Preparing to use the CICS adapter

Before you can use the CICS adapter for messaging, you must start the MQSeries subsystem, if you have not already done so. To do this, use the START command with the appropriate command prefix string (CPF). Figure 17 shows the start command you issue from the operator console if the CPF is '+cpf'.

```
+cpf START QMGR
```

Figure 17. Starting a queue manager that is identified by the CPF '+cpf'

For more information about CPFs, see “Using command prefix strings” on page 42.

Accessing the CICS adapter control panels

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

1. Type CKQC and press Enter.
The CICS adapter control initial panel, shown in Figure 18, is displayed.
2. In the menu bar at the top of the screen, use the TAB key to move between the three options **Connection**, **CKTI**, and **Task**.
3. Press Enter to select your choice.
4. Select the required option from one of the pull-down menus by typing the number of your choice and then pressing Enter to confirm or function key F12 to cancel.
5. Press function key F1 to get help on any panel or window.

```

      Connection      CKTI      Task
-----
CKQCM0  IBM MQSeries for OS/390 - CICS adapter control initial panel

Select menu bar item using Tab key. Then press Enter.

#           #           #####           #####
##          ## ##          ## ##          ##
###         ### ##          ## ##          ##
####        #### ##          ## #####          ##### ## ##          ##### #####
## ##      ## ## ##          ##          ##### # ##### ## # #
## ## ##   ## ##          ## ##          ## ##### ## ## ##### #####
## ## ##   ## ##          ## ##          ## #          ## ## #
## #       ##          ##### ##          ##### ##          ## ##### #####

                                                    for OS/390

(C) Copyright IBM Corporation 1993, 1999. All rights reserved.

F1=Help  F3=Exit
    
```

Figure 18. The CICS adapter control initial panel

Note: You can access the adapter control panels *without* starting the MQSeries subsystem. You can also start a connection but it will not be active until MQSeries is started.

Starting a connection

You can start a connection from:

- The CICS adapter control panels
- The CICS command line
- A CICS application program
- A PLTPI program
- The CICS MQCONN SIT parameter

Starting a connection from the CICS adapter control panels

To start a connection from the CICS adapter control initial panel:

1. Select **Connection** from the menu bar.
2. Select the **Start** action from the pull-down menu. See Figure 19.
3. Modify the connection values displayed in the **Start a Connection** secondary parameter window. Alternatively, use the defaults derived from the INITPARM settings, if defined.
4. Press Enter to confirm.

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

```

Connection      CKTI      Task
-----+-----+-----
| Select an action. | for OS/390 - CICS adapter control initial panel
| 1 1. Start...    | sing Tab key. Then press Enter.
| 2 2. Stop...     |
| 3 3. Modify...   |
| 4 4. Display     |
+-----+-----+-----+
| F1=Help F12=Cancel |
+-----+-----+-----+
## ## ## ## ##
## ## ## ## ## #
## ### ## ###
## # ## #####
+-----+-----+-----+
|                                     | Start a Connection
| Type parameters. Then press Enter. |
+-----+-----+-----+
| 1. Queue Manager Name (SN) . . . QMGR      ##
| 2. Initiation Queue Name (IQ) . . . . . ##
| CICS.INITIATION.QUEUE1                    ##
| 3. Trace Number (TN) . . . . . 123        #
|                                           #
+-----+-----+-----+
| F1=Help F12=Cancel |
+-----+-----+-----+

(C) Copyright IBM Corporation 1993, 1999. All rights reserved.

F1=Help F3=Exit

```

Figure 19. Starting a connection

Starting a connection from the CICS command line

The example shown in Figure 20 starts a connection, using the default connection values set at system initialization.

```
CKQC START
```

Figure 20. Starting a connection from the command line

The command shown in Figure 21 starts a connection, using the explicitly defined connection parameter values. The parameters are positional—every field must be entered to its maximum length if you want to override the default.

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

Figure 21. Starting a connection from the command line specifying parameters

Where:

Y|N

Specify either:

'Y' Use the default values, that is, substitute default values for any blank arguments.

'N' Do not use the default values.

<subsystem ID>

OS/390 subsystem name of the target queue manager.

<trace number>

The trace number. It must be in the range 0 through 199.

<initiation queue name>

The name of the default initiation queue.

Specifying lowercase queue names

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

```
CKQC START Y CSQ1 199 CICS01.INITQ  
ckqc start y csq1 199 cics01.initq
```

Figure 22. Specifying lowercase queue names

If you want to use lowercase queue names, you must:

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

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

For details of TYPETERM and PROFILE definitions, see the *CICS Resource Definition Guide*.

Starting a connection from a CICS application program

You can start a connection by linking the adapter connect program, CSQCQCON, from a CICS application program. Your program, which can be written in C/370, COBOL, PL/I, or assembler language, must pass a parameter list that specifies the connection values to be used. The parameter list is:

CKQC	4-character transaction ID—must be 'CKQC'.
DISPMODE	1-byte field—must contain a blank.
CONNREQ	10-character field—must contain 'START '.
DELIM1	1-byte delimiter field—must contain a blank.
INITP	1-character field that specifies whether this connection is to use the default parameters set by INITPARM. The possible values are: <ul style="list-style-type: none"> 'Y' Use the default values, that is, substitute default values for any blank arguments. 'N' Do not use the default values. If you code 'N', you must supply all the new connection values, to override the INITPARM settings, in the CONSSSN, CONNTN, and CONNIQ fields. ' ' Equivalent to 'Y'.
DELIM2	1-byte delimiter field—must contain a blank.
CONSSSN	4-character field used to specify the OS/390 subsystem name of the target queue manager.
DELIM3	1-byte delimiter field—must contain a blank.
CONNTN	3-character trace number. If supplied, it must be in the range 0 through 199.
DELIM4	1-byte delimiter field—must contain a blank.
CONNIQ	48-character field that specifies the name of the default initiation queue.

Figure 23 shows the LINK command that your CICS program must issue.

```
EXEC CICS LINK PROGRAM('CSQCQCON')
          INPUTMSG(CONNPL) INPUTMSGLEN(length of CONNPL)
```

Figure 23. Linking to the adapter connect program, CSQCQCON, from a CICS program. In this example, the name of the parameter list is CONNPL.

Output messages from CSQCQCON are displayed on the system console.

Stopping a connection

You can stop a connection from:

- The CICS adapter control panels
- The CICS command line
- A CICS application program

Stopping a connection from the CICS adapter control panels

From the initial panel:

1. Select **Connection** from the menu bar.
2. Select the **Stop** action from the pull-down menu.
3. Use the **Stop Connection** secondary parameter window to select the type of shutdown that you require. Methods of shutting down the CICS adapter are summarized in Table 13 on page 144.

```

Connection      CKTI      Task
-----
Select an action.  for OS/390 - CICS adapter control initial panel
2 1. Start...    sing Tab key. Then press Enter
 2. Stop...
 3. Modify...
 4. Display
-----
| F1=Help F12=Cancel |
-----
## ## ## ## ## #
## ## ## ## ## #
## ## ## ## ## #
## # ## ##### #
-----
| F1=Help F12=Cancel |
-----
#
Stop Connection
Select stop type.
Then press Enter
##### ##
1 1. Quiesce # ##### # #
 2. Force   ### ## ## ##### #####
##### ## ## # #
-----
| F1=Help F12=Cancel |
-----
for OS/390

(C) Copyright IBM Corporation 1993, 1999. All rights reserved.

F1=Help F3=Exit
    
```

Figure 24. Stopping a connection from the CKQC initial panel

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

Stopping a connection from the CICS command line

The command shown in Figure 25 initiates a *quiesced* shutdown. The connection shuts down only after the last task has completed its work.

```

CKQC STOP
    
```

Figure 25. Stopping a connection from the command line—a quiesced shutdown

The command shown in Figure 26 initiates a *forced* shutdown. The connection shuts down immediately, regardless of the state of any in-flight tasks.

```
CKQC STOP FORCE
```

Figure 26. Stopping a connection from the command line—a forced shutdown

Stopping a connection from a CICS application program

To stop a connection from a CICS program, the program must link to the adapter shutdown program, CSQCDSC. Figures 27 and 28 show examples of LINK commands initiating quiesced and forced shutdowns. When you do an EXEC CICS LINK to CSQCDSC, the program requires a terminal associated task.

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

Figure 27. Stopping a connection from a CICS application program—a quiesced shutdown. The QUIESCE parameter is optional.

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

Figure 28. Stopping a connection from a CICS application program—a forced shutdown

Output messages from CSQCDSC are displayed on the system console.

Modifying a connection

You can modify a connection to reset the connection statistics, enable or disable the API-crossing exit, or change the adapter's trace number. You can do this from:

- The CICS adapter control panels
- The CICS command line
- A CICS application program

Modifying a connection from the CICS adapter control panels

From the initial panel:

1. Select **Connection** from the menu bar.
2. Select the **Modify** action from the pull-down menu.
3. Use the **Modification Options** secondary parameter window to specify the option you require.

To change the trace number:

- Enter 4 in the options selection field
 - Enter a number, in the range 0 through 199, in the trace number field. Do not change the 4 in the options selection field before you press Enter. If you do, the trace number will not be changed.
4. Press Enter to confirm your choice.
 5. Repeat steps 1 through 4, as required.

```

      Connection      CKTI      Task
+-----+-----+-----+
| Select an action. | for OS/390 - CICS adapter control initial panel
|                 |
| 3 1. Start...   | sing Tab key. Then press Enter.
| 2. Stop...     |
| 3. Modify...   |
| 4. Display     |
|                 |
| F1=Help F12=Cancel |
+-----+-----+-----+
## ## ## ## ##
## ## ## ## ## #
## ## ## ## ##
## # ## #####
+-----+-----+-----+
| Modification Options |
| Select modify option. Then |
| press Enter.         | ##
|                       | ## #####
| 4 1. Reset statistics | ## # #
| 2. Enable API Exit   | ## #####
| 3. Disable API Exit  | ## #
| 4. Change Trace Number 123 | ## #####
|                       |
| F1=Help F12=Cancel | for OS/390
+-----+-----+-----+

(C) Copyright IBM Corporation 1993, 1999. All rights reserved.

F1=Help F3=Exit
    
```

Figure 29. Modifying a connection

Modifying a connection from the CICS command line

You can use the CKQC MODIFY command to modify a connection.

```
CKQC MODIFY Y|N E|D <trace-number>
```

Figure 30. Format of command to modify connection parameters from the command line

The command syntax is shown in Figure 30, where:

Y|N Specify one of:

Y Reset connection statistics.

N Do not reset connection statistics.

This parameter is required.

E|D Specify one of:

E Enable the API-crossing exit.

D Disable the API-crossing exit.

This parameter is optional, the default is to disable the API-crossing exit.

<trace number>

Specify a valid trace number in the range 0 through 199. This parameter is optional. If it is not specified, the trace number is not changed.

The command shown in Figure 31 resets the connection statistics only. The command shown in Figure 32 disables the API-crossing exit and changes the trace number to 121.

```
CKQC MODIFY Y
```

Figure 31. Resetting connection statistics from the command line

```
CKQC MODIFY N D 121
```

Figure 32. Changing the adapter's trace number and disabling the API-crossing exit from the command line

Modifying a connection from a CICS application program

To modify a connection from a CICS program, the program must link to the adapter reset program, CSQCRST.

Figure 33 shows the format of the LINK command. It has the same effect as the command-line requests described in "Modifying a connection from the CICS command line."

```
EXEC CICS LINK PROGRAM('CSQCRST ')
          INPUTMSG('CKQC MODIFY      Y E <trace-number>')
```

Figure 33. Format of the MODIFY command issued from a CICS adapter application program

Modifying a connection

The command shown in Figure 34 on page 134 resets the connection statistics only.

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

Figure 34. Resetting connection statistics from a CICS program

The command shown in Figure 35 disables the API-crossing exit and changes the trace number to 121.

```
EXEC CICS LINK PROGRAM('CSQCRST ')
        INPUTMSG('CKQC MODIFY    N D 121')
```

Figure 35. Linking to the adapter reset program, CSQCRST, from a CICS program

Note: The MODIFY command must be padded to 10 characters, see “Command syntax in application programs” on page 124.

Displaying details of connections and CICS tasks

You can use the CICS adapter control panels to display details of the current connection. The equivalent functionality is not available from the CICS command line or from a CICS application program. However, you can obtain some status information using the CKQC DISPLAY command, see “Displaying connection status and in-flight tasks” on page 142.

Displaying details of a connection from the CICS adapter control panels

From the initial panel:

1. Select **Connection** from the menu bar.
2. Select the **Display** action from the pull-down menu.

Figure 36 shows the details provided:

```

CKQCM2                Display Connection panel
Read connection information. Then press F12 to cancel.

  CICS Applid = VICIC14  Connection Status = Connected      Qmgrname = VCA
  Trace No.   = 124      Tracing                = On         API Exit = Off
  Initiation Queue Name = VICIC14.INITIATION.QUEUE
----- S T A T I S T I C S -----
Number of in-flight tasks = 1          Total No. of API calls = 43912
Number of running CKTI   = 1
      APIs and flows analysis                Syncpoint                Recovery
-----
Run OK      43874  MQINQ      6806  Tasks      26  Indoubt    0
Futile     0      MQSET      0      Backout    0  UnResol   0
MQOPEN     6833  ----- Flows -----  Commit     10  Commit    0
MQCLOSE    6823  Calls      43952  S-Phase    10  Backout   0
MQGET     10032  SyncComp   43922  2-Phase    0
GETWAIT    3399  SuspReqd   0
MQPUT     13399  MsgWait    7      InitTCBs  8  StrtTCBs  8  BusyTCBs  0
MQPUT1     5      Switched   43940
-----
F1=Help  F12=Cancel  Enter=Refresh
    
```

Figure 36. The display connection panel

The display is organized into three areas:

- Top: parameters used for the connection, and current status.
- Middle: connection statistics. These are totals for the current connection, since statistics were last reset.
- Bottom: statistics produced by the adapter.

For an explanation of specific fields on this screen, view the online help panels by pressing function key F1.

Starting an instance of the task initiator CKTI

CKTI is the MQSeries-supplied task initiator³ used in a CICS environment to start a transaction when the trigger conditions on any of its associated MQSeries queues are met. For more information, see “Task initiator” on page 113.

You can start a CKTI instance from:

- The CICS adapter control panels
- The CICS command line
- A CICS application program
- From emulated terminals (see “Automating starting of CKTI” on page 446)

Starting CKTI from the CICS adapter control panels

From the initial panel:

1. Select **CKTI** from the menu bar.
2. Select the **Start** action from the pull-down menu.
3. In the **Start Task Initiator** secondary window, use the **Initiation Queue Name** field to specify the name of the initiation queue to be serviced by this CKTI instance. If you leave this field blank, the default initiation queue is used, if defined.

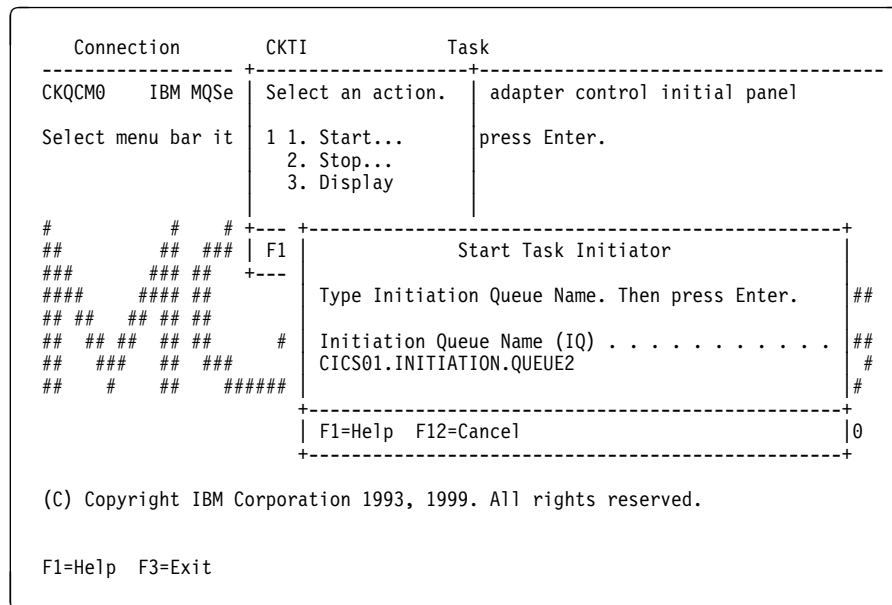


Figure 37. Starting an instance of CKTI

³ Trigger monitor in MQSeries terminology.

Starting CKTI from the CICS command line

The command shown in Figure 38 starts an instance of CKTI to serve the default initiation queue, if defined.

```
CKQC STARTCKTI
```

Figure 38. Starting an instance of CKTI—for the default initiation queue

The command shown in Figure 39 starts an instance of CKTI to serve a specified initiation queue.

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

Figure 39. Starting an instance of CKTI—for a specified initiation queue

Starting CKTI from a CICS application program

To start an instance of CKTI from a CICS program, the program must link to the adapter task initiation program, CSQCSSQ. Figures 40 through 41 show suitable LINK commands. When you do an EXEC CICS LINK to CSQCSSQ, the program requires a terminal associated task.

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

Figure 40. Linking to the adapter task-initiator program CSQCSSQ from CICS. This starts a CKTI that uses the default initiation queue.

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

Figure 41. Linking to the adapter task-initiator program CSQCSSQ from CICS. This starts a CKTI that uses a named initiation queue.

Output messages from CSQCSSQ are displayed on the system console.

Note: The STARTCKTI command must be padded to 10 characters; see “Command syntax in application programs” on page 124.

Stopping an instance of CKTI

You can stop an instance of CKTI by using:

- The CICS adapter control panels
- The CICS command line
- A CICS application program

Stopping an instance of CKTI from the CICS adapter control panels

From the initial panel:

1. Select **CKTI** from the menu bar.
2. Select the **Stop** action from the pull-down menu.
3. Use the **Stop Task Initiator** secondary window to specify the name of the initiation queue serviced by this instance of CKTI. If you leave the name blank, the default initiation queue, if defined, is used.

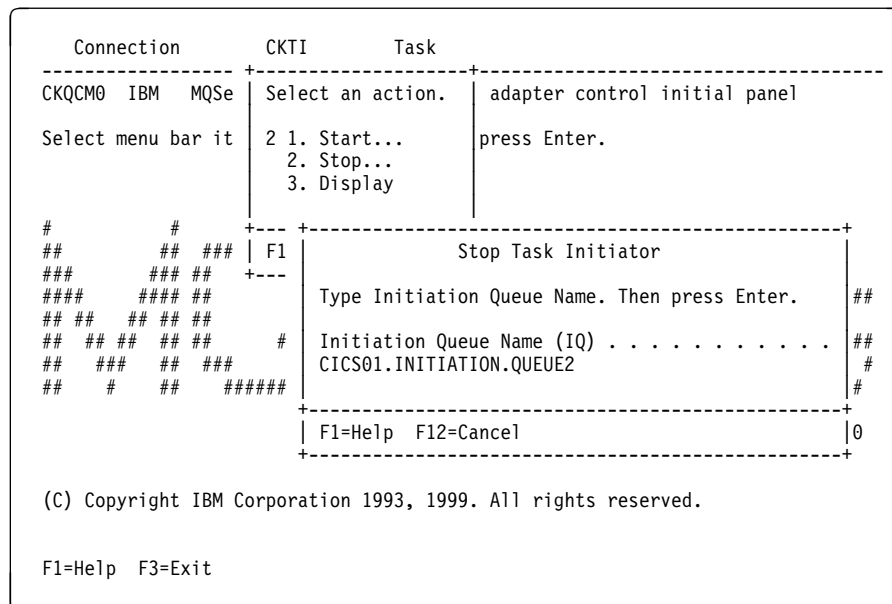


Figure 42. Stopping an instance of the task initiator CKTI

Stopping an instance of CKTI from the command line

The command shown in Figure 43 stops an instance of CKTI that is serving the default initiation queue, if there is one.

```
CKQC STOPCKTI
```

Figure 43. Stopping an instance of CKTI from the command line—for the default initiation queue

The command shown in Figure 44 stops the instance of CKTI that is serving a specified initiation queue.

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

Figure 44. Stopping an instance of CKTI from the command line—for a specified initiation queue

Stopping an instance of CKTI from an application program

You can stop an instance of CKTI by linking to the adapter task-initiator program, CSQCSSQ. Figures 45 through 46 show alternative LINK commands to stop an instance of CKTI from a CICS program. The first command stops the CKTI that is serving the default initiation queue; the second stops the CKTI serving a specified initiation queue.

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

Figure 45. Stopping an instance of CKTI from a program—for the default initiation queue from CICS

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

Figure 46. Stopping an instance of CKTI from a program—for a specified initiation queue from CICS

Note: The STOPCKTI command must be padded to 10 characters; see “Command syntax in application programs” on page 124.

Displaying the current instances of CKTI

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

Displaying the current instances of CKTI from the CICS adapter control panels

From the initial panel:

1. Select **CKTI** from the menu bar
2. Select the **Display** action from the pull-down menu

Figure 47 shows the details provided for each instance of CKTI:

- CICS task number
- Task status
- Thread status
- Number of API calls it has issued
- Most recent API call it has issued
- Name of the initiation queue it is serving

Press function key F1 to display help information about each of the fields in this panel.

```
CKQCM4                      Display CKTI panel
Read CKTI status information. Then press F12 to cancel.
CKTI  1 to  1 of  1
-----
Task No.   Task Status   Thread Status   No-of-APIs   Last API
-----
0000123   Normal           Msg Wait           2           MQGET
Initiation Queue Name: CICS01.INITIATION.QUEUE1

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

Figure 47. The CKQC Display CKTI panel

Displaying CICS task information

You can use the CICS adapter control panels to display information about CICS tasks using MQI calls. The equivalent functionality is not available from the CICS command line or from a CICS application program. However, you can obtain some status information using the CKQC DISPLAY command, see “Displaying connection status and in-flight tasks” on page 142.

Displaying CICS tasks from the CICS adapter control panels

You can display information about the CICS tasks that are currently using MQI calls. From the initial panel:

1. Select **Task** from the menu bar.
2. Select an action from the pull-down menu.

Select option 1, **List all tasks** to obtain information about all tasks that are currently active. To limit the scope of the display, select option 2, **List from task**, to specify the starting number of the first task to be displayed.

CKQCM3 Display Task panel

Read task status information. Then press F12 to cancel.

Tasks 1 to 3 of 3

Txn Id	User Id	Task No.	Task Status	Thread Status	Total APIs	Res Sec	In API-X	Last MQ-Call	Thread ID
PUTQ	CICSUSER	00065	Normal	InQueue	102	No	No	MQPUT1	00012420
GETQ	CICSUSER	00067	Normal	BtnCalls	22	No	No	MQOPEN	00012620
CKTI	CICSUSER	00123	Normal	Msg Wait	2	No	No	MQGET	00012C20

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

Figure 48. The CKQC Display Task panel

Figure 48 shows the details provided for each CICS task:

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

Displaying connection status and in-flight tasks

You can use the CKQC DISPLAY command, from both the CICS command line or from a CICS application program to display limited information about the current connection and CICS tasks. The information from this command is returned in a message CSQC453I, see Figure 49. This message contains:

- The name of the MQSeries subsystem.
- The status of the connection.
- The number of in-flight tasks that are still using the connection.

```
CSQC453I VICY06 CSQCDSPL Status of connection to JAC2 is Connected. 2
tasks are in-flight
```

Figure 49. Message showing the status of a connection

To obtain more detailed information, use the CICS adapter control panels. See “Displaying details of a connection from the CICS adapter control panels” on page 135 and “Displaying CICS tasks from the CICS adapter control panels” on page 141, respectively.

From the CICS command line

You can use the CKQC DISPLAY command, shown in Figure 50, from the CICS command line.

```
CKQC DISPLAY
```

Figure 50. Displaying the status of a connection

Figure 49 shows a typical response to this command. The response messages are sent to your CICS terminal.

From a CICS application program

Figure 51 shows the LINK command for displaying the status of a connection from a CICS application program.

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

Figure 51. Linking to the adapter program CSQCDSPL from a CICS program

Figure 49 shows a typical output from this command. The response messages are sent to the CKQQ queue (the transient data queue).

The COMMAREA option can be used instead of INPUTMSG but only when the program is run at PLT time.

Purging tasks that are using the CICS adapter

You can use the CICS CEMT transaction to purge user tasks that are using the CICS adapter. Tasks that are waiting on the adapter respond only to CEMT SET TASK FORCEPURGE commands—CEMT SET TASK PURGE commands are ignored. The way the adapter handles a FORCEPURGE command depends on the kind of wait state that the task is in:

- If a task is waiting for a message to arrive, for example, the application has issued an **MQGET WAIT** call, the task is abended with code AEXY immediately.
- If the task is waiting for an MQI request to be completed by MQSeries, message CSQC413I is displayed on the system console.

The adapter waits for the request to complete, and then checks whether it is suitable to abend the task:

- If the task is in a critical state, the CICS adapter lets the task continue and ignores the attempt to purge it. This is done to preserve data and system integrity. Message CSQC415I is displayed.

A task is in a critical state is when, for example, it is in the process of completing phase 2 of a two-phase commit sequence.

- If the task is not in a critical state, the adapter abends it with code AEXY. Message CSQC414I is displayed.

For information about CEMT commands, see the *CICS-Supplied Transactions* manual.

Shutting down a connection between MQSeries and the CICS adapter

You can shut down a connection between MQSeries and the CICS adapter by using the CKQC transaction or an application program. There are two types of shutdown:

- Forced
- Quiesced

Other forms of connection shutdown result from a termination of CICS or MQSeries. Table 13 summarizes how the adapter handles different forms of connection shutdown.

Method of shutdown	How this is handled by the adapter
CKQC STOP (A quiesced shutdown)	Mark the status of the adapter as <i>Quiescing</i> . Allow both active and waiting tasks to complete. Allow syncpoint. Do not allow calls from a new task. The last task initiates disconnection from MQSeries.
CKQC STOP FORCE	Mark the status of the adapter as <i>StoppingForce</i> . Disconnect from MQSeries. Resume waiting tasks. Fail any in-flight or following MQI calls.
CICS warm shutdown	Issue message CSQC411I. Initiate a quiesced shutdown of the connection; see CKQC STOP, above.
CICS immediate shutdown	Issue message CSQC410I. Any in-flight tasks using MQSeries are backed out.
CICS abend	Issue message CSQC412I.
MQSeries quiesced	Initiate a quiesced shutdown of the connection; see CKQC STOP, above.
MQSeries abend or forced shutdown	Initiate a forced shutdown of connection; see CKQC STOP FORCE, above.
Notes:	
<ol style="list-style-type: none"> 1. If the connection is not active (for example, quiesced) when CICS or MQSeries shuts down, no action is taken and no messages are issued. 2. "Waiting tasks" includes instances of CKTI, which you must stop before shutdown completes. 	

Orderly shutdown

An orderly shutdown of the connection lets each CICS transaction terminate before thread subtasks are detached. When you use this method, there should be no in-doubt units of work when you reconnect CICS. An orderly termination occurs in each of the following situations:

- The CICS terminal operator issues a CKQC STOP command. CICS and MQSeries remain active. The command can be issued from the command line, from a terminal using the CKQC panels, or from a program, see page 131.
- The CICS terminal operator issues the CICS command:
CEMT PERFORM SHUTDOWN

For information about the CEMT PERFORM SHUTDOWN command, see the *CICS-Supplied Transactions* manual.

- MQSeries is quiesced by the command:

```
+cpf STOP QMGR MODE(QUIESCE)
```

This stops the MQSeries subsystem, allows the currently identified tasks to continue normal execution, and does not allow new tasks to identify themselves to MQSeries. CICS remains active.

Forced shutdown

A forced shutdown of the connection can abend CICS transactions connected to MQSeries. Therefore, there might be in-doubt units of work when the system is reconnected. A forced shutdown occurs in each of these situations:

- The CICS terminal operator issues the CKQC STOP FORCE command. The command can be issued from the command line, from a terminal using the CKQC panels, or from a program, see page 131.
- The CICS terminal operator issues the CICS immediate termination command:

```
CEMT PERFORM SHUTDOWN IMMEDIATE
```

For information about this command, see the *CICS-Supplied Transactions* manual.

MQSeries remains active.

- The MQSeries forced termination command is issued:

```
+cpf STOP QMGR MODE(FORCE) or +cpf STOP QMGR MODE(RESTART)
```

CICS remains active.

- An MQSeries abend occurs. CICS remains active.
- CICS abend occurs. MQSeries remains active.

Shutting down a connection

Chapter 8. The MQSeries-CICS bridge

This chapter describes the CICS bridge.

This chapter contains the following sections:

- “Introduction to the CICS bridge”
- “Customizing the CICS bridge” on page 152
- “Starting the CICS bridge” on page 154
- “Shutting down the CICS bridge” on page 155

Introduction to the CICS bridge

The MQSeries-CICS bridge enables an application, not running in a CICS environment, to run a *program* or *transaction* on CICS and get a response back. This non-CICS application can be run from any environment that has access to an MQSeries network that encompasses MQSeries for OS/390.

A *program* is a CICS program that can be invoked using the EXEC CICS LINK command. It must conform to the DPL subset of the CICS API, that is, it must not use CICS terminal or syncpoint facilities.

A *transaction* is a CICS transaction designed to run on a 3270 terminal. This transaction can use BMS or TC commands. It can be conversational or part of a pseudoconversation. It is permitted to issue syncpoints. For further details about the transactions that can be run, see Part 5 of the *CICS Internet and External Interfaces Guide* (Bridging to 3270 transactions).

When to use the CICS bridge

The CICS bridge allows an application to run a single CICS program or a ‘set’ of CICS programs (often referred to as a unit of work). It caters for the application that waits for a response to come back before it runs the next CICS program (synchronous processing) and for the application that requests one or more CICS programs to run, but doesn't wait for a response (asynchronous processing).

The CICS bridge also allows an application to run a 3270-based CICS transaction, without knowledge of the 3270 data stream.

The CICS bridge uses standard CICS and MQSeries security features and can be configured to authenticate, trust, or ignore the requestor's user ID.

Given this flexibility, there are many instances where the CICS bridge can be used. For example, when you want:

- To write a new MQSeries application that needs access to logic or data (or both) that reside on your CICS server.
- To be able to run CICS programs from a Lotus Notes® application.
- To be able to access your CICS applications from
 - Your MQSeries Classes for Java client application
 - A web browser using the MQSeries Internet gateway

Introduction to the CICS bridge

For information about how to write an MQSeries-CICS bridge application, see the *MQSeries Application Programming Guide*.

System configuration for the CICS bridge

When you are setting your system up, you should ensure that:

- Both MQSeries and CICS are running in the same OS/390 image.
- The MQSeries request queue is local to the CICS bridge, however the response queue can be local or remote.
- The CICS bridge tasks run in the same CICS as the bridge monitor. The user programs can be in the same or a different CICS system.
- The MQSeries-CICS adapter is enabled.

Running CICS DPL programs

Data necessary to run the program is provided in the MQSeries message. The bridge builds a COMMAREA from this data, and runs the program using EXEC CICS LINK. Figure 52 shows the step sequence taken to process a single message to run a CICS DPL program:

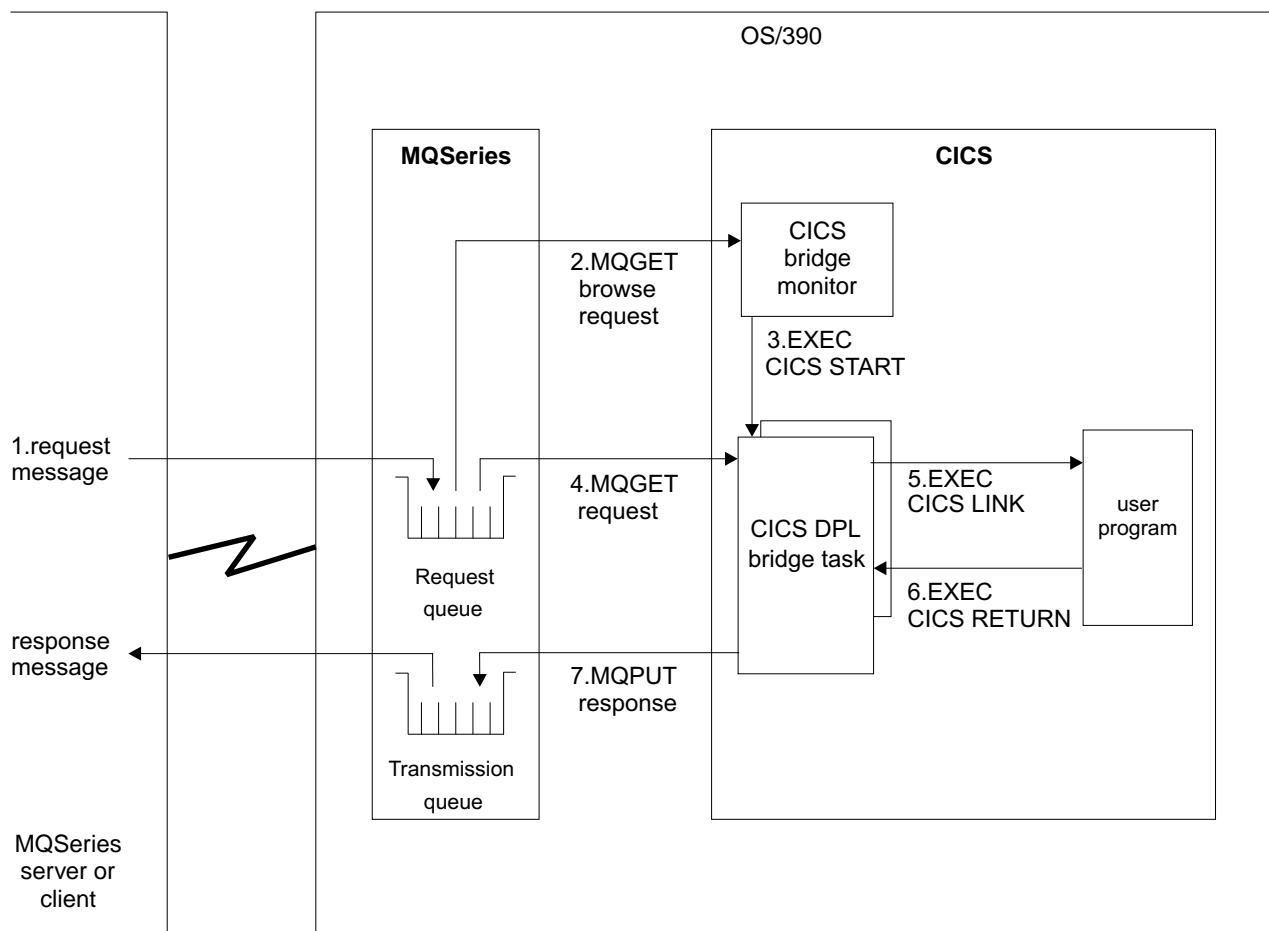


Figure 52. Components and data flow to run a CICS DPL program

The following takes each step in turn, and explains what takes place:

1. A message, with a request to run a CICS program, is put on the request queue.
2. The CICS bridge monitor task, which is constantly browsing the queue, recognizes that a 'start unit of work' message is waiting (*CorrelId*=MQCI_NEW_SESSION).
3. Relevant authentication checks are made, and a CICS DPL bridge task is started with the appropriate authority (see "Security considerations for the CICS bridge" on page 447 for more information).
4. The CICS DPL bridge task removes the message from the request queue.
5. The CICS DPL bridge task builds a COMMAREA from the data in the message and issues an EXEC CICS LINK for the program requested in the message.
6. The program returns the response in the COMMAREA used by the request.
7. The CICS DPL bridge task reads the COMMAREA, creates a message, and puts it on the reply-to queue specified in the request message. All response messages (normal and error, requests and replies) are put to the reply-to queue with default context.
8. The CICS DPL bridge task ends.

A unit of work can be just a single user program, or it can be multiple user programs. There is no limit to the number of messages you can send to make up a unit of work.

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

Running CICS 3270 transactions

Data necessary to run the transaction is provided in the MQSeries message. The CICS transaction runs as if it has a real 3270 terminal, but instead uses one or more MQ messages to communicate between the CICS transaction and the MQSeries application

Unlike traditional 3270 emulators, the bridge does not work by replacing the VTAM® flows with MQSeries messages. Instead, the message consists of a number of parts called vectors, each of which corresponds to an EXEC CICS request. Therefore the application is talking directly to the CICS transaction, rather than via an emulator, using the actual data used by the transaction (known as application data structures or ADSs).

Figure 53 on page 150 shows the step sequence taken to process a single message to run a CICS 3270 transaction.

The following takes each step in turn, and explains what takes place:

1. A message, with a request to run a CICS transaction, is put on the request queue.
2. The CICS bridge monitor task, which is constantly browsing the queue, recognizes that a 'start unit of work' message is waiting (*CorrelId*=MQCI_NEW_SESSION).

Introduction to the CICS bridge

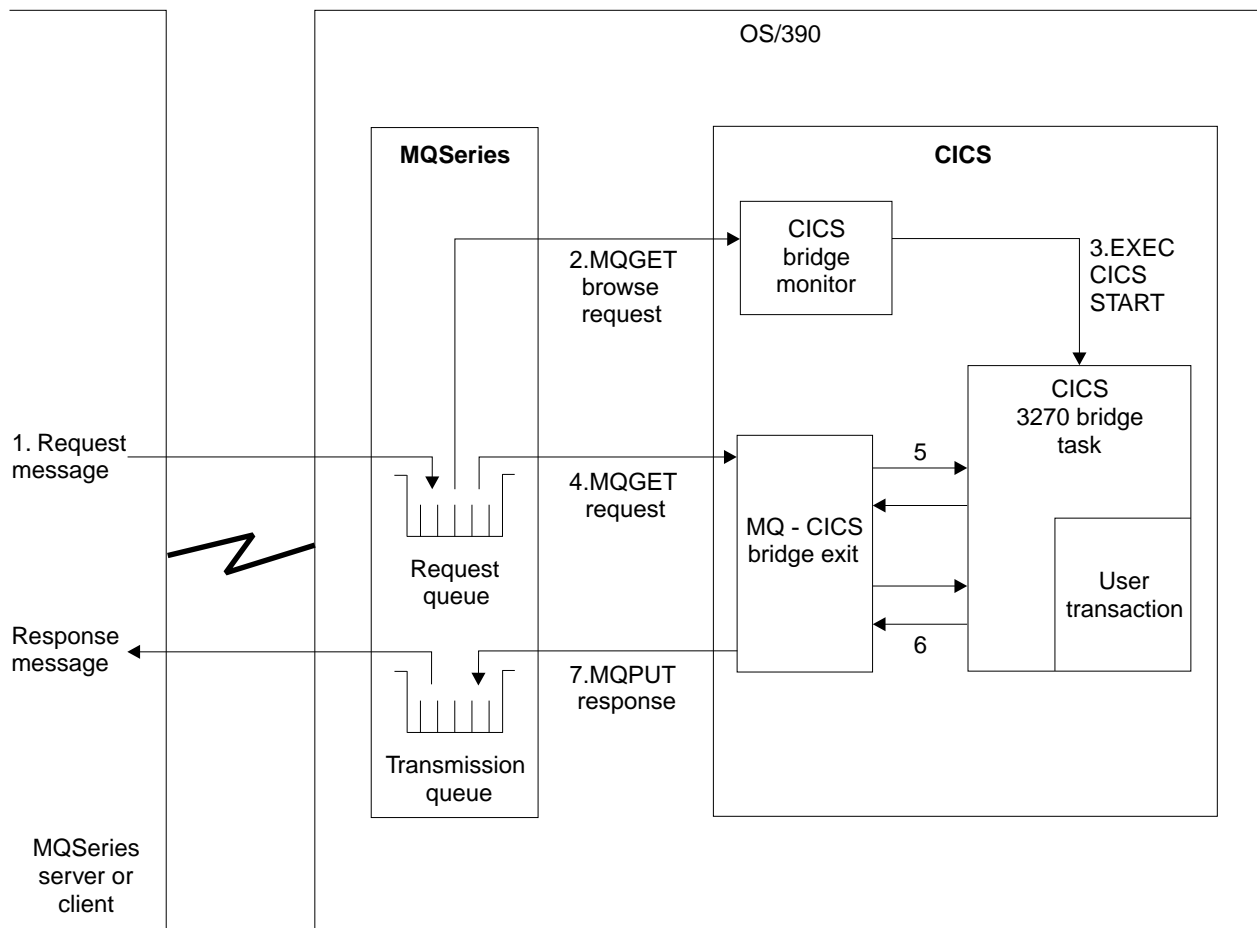


Figure 53. Components and data flow to run a CICS 3270 transaction

3. Relevant authentication checks are made, and a CICS 3270 bridge task is started with the appropriate authority (see “Security considerations for the CICS bridge” on page 447 for more information).
4. The MQ-CICS bridge exit removes the message from the queue and changes task to run a user transaction
5. Vectors in the message provide data to answer all terminal related input EXEC CICS requests in the transaction.
6. Terminal related output EXEC CICS requests result in output vectors being built.
7. The MQ-CICS bridge exit builds all the output vectors into a single message and puts this on the reply-to queue.
8. The CICS 3270 bridge task ends.

Note: The CICS bridge exit is an MQSeries supplied CICS exit associated with the bridge transaction.

A traditional CICS application usually consists of one or more transactions linked together as a pseudoconversation. In general, each transaction is started by the 3270 terminal user entering data onto the screen and pressing an AID key. This model of application can be emulated by an MQSeries application. A message is built for the first transaction, containing information about the transaction, and input

vectors. This is put on the queue. The reply message will consist of the output vectors, the name of the next transaction to be run, and a token that is used to represent the pseudoconversation. The MQSeries application builds a new input message, with the transaction name set to the next transaction and the facility token set to the value returned on the previous message. Vectors for this second transaction are added to the message, and the message put on the queue. This process is continued until the application ends.

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

For more information about this, see the *CICS Internet and External Interfaces Guide*.

Customizing the CICS bridge

This section describes what you have to do to customize the MQSeries-CICS bridge. The bridge is described in “Introduction to the CICS bridge” on page 147.

Prerequisite APARs

To run 3270 transactions, you must be using CICS Transaction Server for OS/390 Release 2 or later, with APARs PQ13011 and PQ13012 applied.

Before you can run the bridge you must ensure that your OS/390 system has both the CICS and MQSeries components in place:

1. On your CICS system:
 - a. Run the resource definition utility DFHCSDUP, using the sample thlqual.SCSQPROC(CSQ4CKBC) as input, to define the bridge transactions and programs:

CKBR	Bridge monitor transaction
CSQCBR00	Bridge monitor program
CKBP	Bridge ProgramLink transaction
CSQCBP00	Bridge ProgramLink program
CSQCBP10	Bridge ProgramLink abend handler program
CSQCBTX	Bridge error messages
CSQCBE00	3270 bridge exit for MQSeries
CBR1 through CBR8	3270 bridge transaction definitions

- b. Add the group, CSQCKB, to your startup group list.

Notes:

- a. The bridge uses CICS temporary storage IDs with the prefix CKB. You should make sure these are not recoverable.
 - b. By default, your CICS DPL programs will be run under transaction code CKBP. You will need to change the TASKDATALOC attribute to 'BELOW' if you are going to run 24-bit programs, otherwise you will get a CICS abend AEZC. If you wish to run your programs under different transaction codes you will need to install copies of the definition of CKBP, changing the transaction name to the ones of your choice. DPL bridge transactions must not be routed to a remote system.
2. On your MQSeries system:

- a. Define a local queue for the request messages.

You can use the sample thlqual.SCSQPROC(CSQ4CKBM) to define a queue named SYSTEM.CICS.BRIDGE.QUEUE, or define your own. If you define your own, you must set the following attributes:

SHARE

So that both the monitor and the bridge tasks can read it.

MSGDLVSQ(FIFO)

So that messages are processed in FIFO sequence (not priority sequence).

If recovery is required, set the following attributes:

DEFPSIST(YES)

Set messages as persistent on the queue by default.

HARDENBO

Set HARDENBO to ensure that messages are not re-processed erroneously after an emergency restart.

- b. Define one or more queues to hold the responses, as required. If your response queue is remote, you must define a transmission queue to hold the responses before they are forwarded to the response queue.

3. Security:

You might need to add RACF definitions, depending on the authentication option you choose to use. See “Security considerations for the CICS bridge” on page 447 for more information about this.

If the bridge is to be accessed remotely from MQSeries for OS/390, you need channel and transmission queue definitions, and a remote queue definition for the request queue. For more information about using remote queues see the *MQSeries Intercommunication* manual.

Note: The MQSeries queue defined to hold requests for the CICS bridge must not be used by any other application. Each CICS bridge monitor task started requires its own MQSeries queue to hold requests.

Starting the CICS bridge

To start the bridge, you need to run the CKBR transaction providing a maximum of three parameters:

- `Q=qqq`, where `qqq` is the name of the queue holding requests. If you don't specify one, the default is:

```
SYSTEM.CICS.BRIDGE.QUEUE
```

Remember that names of objects within MQSeries are case-sensitive.

- `WAIT=nnn`, where `nnn` is the number of seconds that you want the bridge task to wait for second and subsequent requests before timing out when processing a unit of work that runs many user programs.

The default wait time is unlimited.

You are recommended to specify a wait time. If you don't, the CICS bridge might inhibit CICS or MQSeries shut down.

- `AUTH=xxx`, where `xxx` is the security option. The default is `LOCAL`. See "Security considerations for the CICS bridge" on page 447 for more information.

Start the CKBR task running by using one of the following methods:

- Input a single line from a terminal (3270 or other). Note that the terminal is not freed until the monitor ends. The format is:

```
<trancode> <parameters>
```

```
CKBR Q = <queue name>, AUTH = <auth option>, WAIT = nnn
```

For example:

```
CKBR Q = MyQueue, AUTH = IDENTIFY, WAIT = 30
```

- Issue an EXEC CICS START for the CKBR program with the parameters as data.
- Issue an EXEC CICS LINK to the program CSQCBR00 with the parameters as data in the commarea.
- Use TRIGGER TRIGTYPE(FIRST) from the bridge request queue to a process specifying APPLICID(CKBR), with any parameters for the AUTH and WAIT options in USERDATA.

The level of security you want to use will influence how you start the monitor task. See "Security considerations for the CICS bridge" on page 447 for more information on the security options available to you.

Tuning considerations

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

If a high volume of requests is expected, you could consider starting a second or subsequent monitor task. To do this, you must create another request queue for the sole use of this monitor (and the bridge tasks it starts).

Shutting down the CICS bridge

There are various ways in which you can shut down the CICS bridge:

- By altering the attributes of the request queue by setting GET(DISABLED)
- By shutting CICS down
- By shutting MQSeries down

Whichever method you choose, it will attempt to allow all the requests in progress to complete first.

However in the event that this is not possible, the problems encountered are reported on the CICS CSMT log.

Note: The CICS bridge does not stop CICS or MQSeries if either of them are in the process of shutting down, unless bridge tasks started with WAIT_UNLIMITED have **MQGET** calls outstanding for second or subsequent messages in a unit of work.

Restarting the monitor

The monitor requires exclusive use of the request queue during its initialization, so the monitor cannot be restarted until all bridge tasks for the queue have terminated.

Shutting down the CICS bridge

Part 4. MQSeries and IMS

Chapter 9. The MQSeries-IMS adapter	159
Introduction to the IMS adapter	159
Using the adapter	159
The IMS trigger monitor	160
Setting up the IMS adapter	161
Defining MQSeries to IMS	162
Defining the MQSeries subsystem to the IMS adapter	165
The IMS trigger monitor	167
Chapter 10. Operating the IMS adapter	169
Controlling IMS connections	169
Connecting from the IMS control region	170
Initializing the adapter and connecting to MQSeries	170
Thread attachment	171
Displaying in-doubt units of recovery	172
Recovering in-doubt units of recovery	172
Resolving residual recovery entries	173
Controlling IMS dependent region connections	174
Connecting from dependent regions	174
Region error options	174
Monitoring the activity on connections	174
Disconnecting from dependent regions	175
Disconnecting from IMS	176
Controlling the IMS trigger monitor	177
Starting CSQQTRMN	177
Stopping CSQQTRMN	178
Chapter 11. The MQSeries-IMS bridge	179
Introduction to the IMS bridge	179
What is OTMA?	179
Submitting IMS transactions from MQSeries	180
Customizing the IMS bridge	181
Controlling the IMS bridge	182
Controlling IMS connections	182
Controlling bridge queues	183
Deleting messages from IMS	183
Resynchronizing the IMS bridge	184
Security	185

Chapter 9. The MQSeries-IMS adapter

This chapter introduces the MQSeries-IMS adapter (also referred to as the IMS adapter). It contains these sections:

- “Introduction to the IMS adapter”
- “Setting up the IMS adapter” on page 161

Introduction to the IMS adapter

The MQSeries adapters enable different application environments to send and receive messages through a message queuing network. The IMS adapter is the interface between IMS application programs and an MQSeries subsystem. It makes it possible for IMS application programs to use the MQI.

The IMS adapter provides access to MQSeries resources for programs running in:

- Task (TCB) mode
- Problem state
- Non-cross-memory mode
- Non-access register mode

The adapter provides a connection thread from an application task control block (TCB) to MQSeries.

The adapter supports a two-phase commit protocol for changes made to resources owned by MQSeries with IMS acting as the syncpoint coordinator.

The adapter also provides a trigger monitor transaction (CSQQTRMN). This is described in “The IMS trigger monitor” on page 160.

Using the adapter

The application programs and the IMS adapter run in the same address space. MQSeries for OS/390 is a separate OS/390 subsystem, in its own address space.

Each program that issues one or more MQI calls must be link-edited to a suitable IMS language interface module, and, unless it uses dynamic MQI calls, the MQSeries-supplied stub program, CSQQSTUB. When the application issues an MQI call, the stub transfers control to the adapter through the IMS external subsystem interface, which manages the processing of the request by the message queue manager.

Note: If the adapter detects an unexpected MQSeries subsystem error, it issues an OS/390 SNAP dump to DDname CSQSNAP, and issues reason code MQRC_UNEXPECTED_ERROR to the application.

If this occurs, rerun the application with a CSQSNAP DD statement included in the IMS dependent region JCL, and contact your IBM support center.

The IMS trigger monitor

CSQQTRMN is an MQSeries-supplied IMS application that starts an IMS transaction when an MQSeries event occurs, for example, when a message is put onto a specific queue.

How it works

When a message is put onto a message queue, a trigger is generated if the trigger conditions are met. The queue manager then writes a message (containing some user defined data), known as a *trigger message*, to the initiation queue that has been specified for that message queue. In an IMS environment, an instance of CSQQTRMN can be started to monitor an initiation queue and to retrieve the trigger messages from it as they arrive. Typically, CSQQTRMN schedules another IMS transaction by an ISRT to the IMS message queue. The IMS application reads the message from the message queue and then processes it. CSQQTRMN must run as a non-message BMP.

Each copy of CSQQTRMN services a single initiation queue. Once started, the trigger monitor runs until MQSeries or IMS ends.

The APPLCTN macro for CSQQTRMN must specify SCHDTYP=PARALLEL.

Because the trigger monitor is a batch-oriented BMP, IMS transactions started by the trigger monitor will contain:

- Blanks in the LTERM field of the IOPCB
- The PSB name of the trigger monitor BMP in the Userid field of the IOPCB

If the target IMS transaction is RACF protected, you might need to define CSQQTRMN as a user ID to RACF.

Setting up the IMS adapter

This section tells you how to make the MQSeries-IMS adapter (referred to in this book as the IMS adapter) available to your IMS subsystem. If you are not familiar with tailoring an IMS subsystem, see the *IMS Customization Guide*.

To make the IMS adapter available to IMS applications, follow these steps:

1. Define MQSeries to IMS as an external subsystem using the IMS external subsystem attach facility (ESAF). See “Defining MQSeries to IMS” on page 162.
2. Include the MQSeries load library thlqual.SCSQAUTH in the JOBLIB or STEPLIB concatenation in the JCL for your IMS control region and for any dependent region that connects to MQSeries. If your JOBLIB or STEPLIB is not authorized, also include it in the DFSESL concatenation after the library containing the IMS modules (usually IMS RESLIB).

Also include thlqual.SCSQANLx (where x is the language letter).
3. Copy the MQSeries assembler program CSQQDEFV from thlqual.SCSQASMS to a user library.
4. The supplied program, CSQQDEFV, contains one subsystem name CSQ1 identified as default with an IMS language interface token (LIT) of MQM1. You can retain this name for testing and installation verification. For production subsystems, you can change the NAME=CSQ1 to your own subsystem name or use CSQ1. You can add further subsystem definitions as required. See “Defining the MQSeries subsystem to the IMS adapter” on page 165.
5. Assemble and link-edit the program to produce the CSQQDEFV load module. For the assembly, include the library thlqual.SCSQMACS in your SYSLIB concatenation; use the link-edit parameters RENT,AMODE=31,RMODE=ANY. This is shown in the sample JCL in thlqual.SCSQPROC(CSQ4DEFV).
6. Include the user library containing the module CSQQDEFV that you created in the JOBLIB or STEPLIB concatenation in the JCL for your IMS control region and for any dependent region that connects to MQSeries. If you do not do this, you will receive a user 3041 abend from IMS.
7. If you want to use dynamic MQI calls (described in the *MQSeries Application Programming Guide*), build the dynamic stub, as shown in Figure 54 on page 162.
8. If you want to use the IMS trigger monitor, define the IMS trigger monitor application CSQQTRMN, and perform PSBGEN and ACBGEN. See “The IMS trigger monitor” on page 167.
9. If you are using RACF to protect resources in the OPERCMDS class, ensure that your MQSeries system has authority to issue the MODIFY command to any IMS system to which it might connect.

```
//DYNSTUB EXEC PGM=IEWL,PARM='RENT,REUS,MAP,XREF'  
//SYSPRINT DD SYSOUT=*  
//ACSQMOD DD DISP=SHR,DSN=th1qua1.SCSQLOAD  
//IMSLIB DD DISP=SHR,DSN=ims.reslib  
//SYSLMOD DD DISP=SHR,DSN=private.load4  
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,1)  
//SYSLIN DD *  
    INCLUDE ACSQMOD(CSQQSTUB)  
    INCLUDE IMSLIB (DFSLI000)  
    ALIAS MQCONN,MQDISC           MQI entry points  
    ALIAS MQGET,MQPUT,MQPUT1      MQI entry points  
    ALIAS MQOPEN,MQCLOSE         MQI entry points  
    ALIAS MQBACK,MQCMIT          MQI entry points  
    ALIAS CSQBBAK,CSQBCMT        MQI entry points  
    ALIAS MQINQ,MQSET            MQI entry points  
    ALIAS DFSPLI,PLITDLI         IMS entry points  
    ALIAS DFSCOBOL,CBLTDLI       IMS entry points  
    ALIAS DFSFOR,FORTDLI         IMS entry points  
    ALIAS DFSASM,ASMTDLI         IMS entry points  
    ALIAS DFSPASCL,PASTDLI       IMS entry points  
    ALIAS DFHEI01,DFHEI1         IMS entry points  
    ALIAS DFSAIBLI,AIBTDLI       IMS entry points  
    ALIAS DFSESS,DSNWLI,DSNHLI   IMS entry points  
    MODE AMODE(31),RMODE(ANY)    Note RMODE  
    NAME CSQQDYN(S)  
/*
```

Figure 54. Sample JCL to link-edit the dynamic call stub. This includes the IMS language interface module and the MQSeries IMS stub CSQQSTUB.

Defining MQSeries to IMS

An MQSeries instance must be defined to the control region, and to each dependent region accessing that MQSeries subsystem. To do this, you must create a subsystem member (SSM) in the IMS.PROCLIB library, and identify the SSM to the applicable IMS regions.

Placing the subsystem member entry in IMS.PROCLIB

Each SSM entry in IMS.PROCLIB defines a connection from an IMS region to a different subsystem.

To name an SSM member, concatenate the value (one to four alphanumeric characters) of the IMSID field of the IMS IMSCTRL macro with any name (one to four alphanumeric characters) defined by your site.

One SSM member can be shared by all of the IMS regions, or a specific member can be defined for each region. This member contains as many entries as there are connections to external subsystems. Each entry is an 80-character record.

⁴ Specify the name of a library accessible to IMS applications that wish to make dynamic calls to MQSeries.

Positional parameters: The fields in this entry are:

SSN,LIT,ESMT,RTT,REO,CRC

where:

SSN Specifies the MQSeries subsystem name. It is required, and must contain one through four characters. This name must be the name you specified in the subsystem name table (see “Updating the subsystem name table” on page 41).

LIT Specifies the language interface token (LIT) supplied to IMS. This field is required, its value must match one in the CSQQDEFV module.

ESMT Specifies the external subsystem module table (ESMT). This table specifies which attachment modules must be loaded by IMS. CSQQESMT is the required value for this field.

RTT This option is not supported by MQSeries.

REO Specifies the region error option (REO) to be used if an IMS application tries to reference a non-operational external subsystem or if resources are unavailable at create thread time. This field is optional and contains a single character, which can be:

R Passes a return code to the application, indicating that the request for MQSeries services failed.

Q Abends the application with an abend code U3051, backs out activity to the last commit point, does a PSTOP of the transaction, and requeues the input message. This option only applies when an IMS application tries to reference a non-operational external subsystem or if the resources are unavailable at create thread time.

MQSeries completion and reason codes are returned to the application if the MQSeries problem occurs while MQSeries is processing the request; that is, after the adapter has passed the request on to MQSeries.

A Abends the application with an abend code of U3047 and discards the input message. This option only applies when an IMS application tries to reference a non-operational external subsystem or if the resources are unavailable at create thread time.

MQSeries completion and reason codes are returned to the application if the MQSeries problem occurs while MQSeries is processing the request; that is, after the adapter has passed the request on to MQSeries.

CRC This option can be specified but is not used by MQSeries.

Setting up the IMS adapter

An example SSM entry is:

```
CSQ1, MQM1, CSQQESMT, , R,
```

where:

CSQ1	The default subsystem name as supplied with MQSeries. You can change this to suit your installation.
MQM1	The default LIT as supplied in CSQQDEFV.
CSQQESMT	The external subsystem module name. You must use this value.
R	REO option.

Keyword parameters: MQSeries parameters can be specified in keyword format; to do this you must specify SST=DB2. Other parameters are as described in “Positional parameters” on page 163, and shown in the following example:

```
SST=DB2, SSN=SYS3, LIT=MQM3, ESMT=CSQQESMT
```

where:

SYS3	The subsystem name
MQM3	The LIT as supplied in CSQQDEFV
CSQQESMT	The external subsystem module name

Specifying the SSM EXEC parameter

Specify the SSM EXEC parameter in the start up procedure of the IMS control region. This parameter specifies the one-character to four-character subsystem member name (SSM).

If you specify the SSM for the IMS control region, any dependent region running under the control region can attach to the MQSeries subsystem named in the IMS.PROCLIB member specified by the SSM parameter. The IMS.PROCLIB member name is the IMS ID (IMSID=xxxx) concatenated with the one to four characters specified in the SSM EXEC parameter. The IMS ID is the IMSID parameter of the IMSCTRL generation macro.

IMS lets you define as many external subsystem connections as are required. More than one connection can be defined for different MQSeries subsystems. All MQSeries connections must be within the same OS/390 system. For a dependent region, you can specify a dependent region SSM or use the one specified for the control region. You can specify different region error options (REOs) in the dependent region SSM member and the control region SSM member. Table 14 on page 165 shows the different possibilities of SSM specifications.

Table 14. SSM specifications options

SSM for control region	SSM for dependent region	Action	Comments
No	No	None	No external subsystem can be connected.
No	Yes	None	No external subsystem can be connected.
Yes	No	Use the control region SSM	Applications scheduled in the region can access external subsystems identified in the control region SSM. Exits and control blocks for each attachment are loaded into the control region and the dependent region address spaces.
Yes	Yes (empty)	No SSM is used for the dependent region	Applications scheduled in this region can access DL/I databases only. Exits and control blocks for each attachment are loaded into the control region address space.
Yes	Yes (not empty)	Check the dependent region SSM with the control region SSM	Applications scheduled in this region can access only external subsystems identified in both SSMs. Exits and control blocks for each attachment are loaded into the control region and the dependent region address spaces.

There is no specific parameter to control the maximum number of SSM specification possibilities.

Defining the MQSeries subsystem to the IMS adapter

The IMS adapter cannot access the IMS PROCLIB so the names of the MQSeries subsystems and their corresponding LITs must be defined in the subsystem definition table, CSQQDEFV. Use the supplied CSQQDEFX macro to create the CSQQDEFV load module. Figure 55 shows the syntax of this assembler macro.

```
CSQQDEFX TYPE=ENTRY|DEFAULT,NAME=subsystem,LIT=token
or
CSQQDEFX TYPE=END
```

Figure 55. CSQQDEFX macro syntax

Parameters

TYPE=ENTRY|DEFAULT

Specify either TYPE=ENTRY or TYPE=DEFAULT as follows:

TYPE=ENTRY

Specifies that a table entry describing an MQSeries subsystem available to an IMS application is to be generated. If this is the first entry, the table header is also generated, including a CSQQDEFV CSECT statement.

TYPE=DEFAULT

As for TYPE=ENTRY. The subsystem specified is the default subsystem to be used when **MQCONN** specifies a name that is all blanks. There must be only one such entry in the table.

NAME=*subsystem*

Specifies the name of the subsystem, as specified with **MQCONN**.

LIT=*token*

Specifies the name of the language interface token (LIT) that IMS uses to identify the subsystem. The LIT for each CSQQDEFX entry must be unique.

Note: An **MQCONN** call associates the *name* input parameter and the *hconn* output parameter with the name label and, therefore, the LIT in the CSQQDEFV entry. Further MQSeries calls passing the *hconn* parameter use the LIT from the CSQQDEFV entry identified in the **MQCONN** call to direct calls to the MQSeries instance defined in the IMS SSM PROCLIB member with that same LIT.

In summary, the *name* parameter on the **MQCONN** call identifies a LIT in CSQQDEFV and the same LIT in the SSM member identifies an MQSeries instance. (For information about the **MQCONN** call, see the *MQSeries Application Programming Reference* manual.)

TYPE=END

Specifies that the table is complete. If this parameter is omitted, TYPE=ENTRY is assumed.

Using the CSQQDEFX macro

Figure 56 shows the general layout of a subsystem definition table.

```
CSQQDEFX NAME=subsystem1,LIT=token1
CSQQDEFX NAME=subsystem2,LIT=token2,TYPE=DEFAULT
CSQQDEFX NAME=subsystem3,LIT=token3
...
CSQQDEFX NAME=subsystemN,LIT=tokenN
CSQQDEFX TYPE=END
END
```

Figure 56. Layout of a subsystem definition table

The IMS trigger monitor

Define the application to IMS using the model CSQQTAPL in the thlqual.SCSQPROC library (see Figure 57).

Generate the PSB and ACB using the model CSQQTPSB in the thlqual.SCSQPROC library (see Figure 58).

```

          TITLE 'CSQQTAPL - Transaction Definition for CSQQTRMN'
          SPACE 1
*
*****
*
* This is the application definition
* for the IMS Trigger Monitor BMP
*
* The class parameter on the PGMTYPE keyword can be modified
* to meet installation conventions.
*
*****
          SPACE 1
          APPLCTN PSB=CSQQTRMN,
              PGMTYPE=BATCH,
              SCHDTYP=PARALLEL
          SPACE 1
X
X

```

Figure 57. Example CSQQTAPL transaction definition for CSQQTRMN

```

          TITLE 'CSQQTPSB - PSB for IMS trigger monitor'
          SPACE 1
*****
*
* This is the PSB for the MQSeries IMS trigger monitor program,
* CSQQTRMN.
*
*****
          SPACE 1
          PCB TYPE=TP,          ALTPCB for transaction messages
              MODIFY=YES,      To "triggered" IMS transaction
              PCBNAME=CSQQTRMN
          PCB TYPE=TP,          ALTPCB for diagnostic messages
              MODIFY=YES,      To LTERM specified or "MASTER"
              PCBNAME=CSQQTRMG,
              EXPRESS=YES
          PSBGEN LANG=ASSEM,
              PSBNAME=CSQQTRMN,  Runs program CSQQTRMN
              CMPAT=YES
          END
X
X
X
X
X
X

```

Figure 58. Example CSQQTPSB PSB definition for CSQQTRMN

Setting up the IMS adapter

Chapter 10. Operating the IMS adapter

This chapter describes how to operate the IMS adapter, which connects MQSeries to IMS systems.

Note: The IMS adapter does not incorporate any operations and control panels.

This chapter contains the following sections:

- “Controlling IMS connections”
- “Connecting from the IMS control region” on page 170
- “Displaying in-doubt units of recovery” on page 172
- “Controlling IMS dependent region connections” on page 174
- “Disconnecting from IMS” on page 176
- “Controlling the IMS trigger monitor” on page 177

Controlling IMS connections

IMS provides these operator commands to control and monitor the connection to MQSeries:

/CHANGE SUBSYS

Deletes an in-doubt unit of recovery from IMS.

/DISPLAY OASN SUBSYS

Displays outstanding recovery elements.

/DISPLAY SUBSYS

Displays connection status and thread activity.

/START SUBSYS

Connects the IMS control region to an MQSeries subsystem.

/STOP SUBSYS

Disconnects IMS from an MQSeries subsystem.

/TRACE

Controls the IMS trace.

For more information about these commands, see the *IMS/ESA Operator's Reference* manual for the level of IMS that you are using.

IMS command responses are sent to the terminal from which the command was issued. Authorization to issue IMS commands is based on IMS security.

Connecting from the IMS control region

IMS makes one connection from its control region to each MQSeries subsystem. IMS can make the connection in these ways:

- Automatically during either:
 - A cold-start initialization.
 - A warm start of IMS, if the MQSeries connection was active when IMS was shut down.
- In response to the IMS command:
`/START SUBSYS sysid`
where *sysid* is the MQSeries subsystem name.

This command causes the following message to be displayed at the logical terminal (LTERM):

```
DFS058  START COMMAND COMPLETED
```

The command is issued regardless of whether MQSeries is active or not, and does not imply that the connection has been established.

The order in which you start IMS and MQSeries is not significant. If you start IMS first, then, when MQSeries starts, MQSeries posts the control region modify task, and IMS again tries to reconnect.

IMS cannot reconnect to MQSeries automatically if MQSeries is stopped with a STOP QMGR command, the /STOP SUBSYS IMS command, or an abnormal end.

Therefore, you must make the connection by using the /START SUBSYS IMS command.

Initializing the adapter and connecting to MQSeries

The adapter is a set of modules loaded into the IMS control and dependent regions, using the IMS external subsystem attach facility.

This procedure initializes the adapter and connects to MQSeries:

1. Read the subsystem member (SSM) from IMS.PROCLIB. The SSM chosen is an IMS EXEC parameter. There is one entry in the member for each MQSeries subsystem to which IMS can connect. Each entry contains control information about an MQSeries adapter.
2. Load the IMS adapter.
Note: IMS loads one copy of the adapter modules for each MQSeries instance that is defined in the SSM member.
3. Attach the external subsystem task for MQSeries.
4. Run the adapter with the CTL EXEC parameter (IMSID) as the connection name.

The process is the same whether the connection is part of initialization or a result of the /START SUBSYS IMS command.

If MQSeries is active when IMS tries to make the connection, the following messages are sent:

- To the OS/390 console:
DFS3613I ESS TCB INITIALIZATION COMPLETE
- To the IMS master terminal:
CSQQ000I IMS/TM *imsid* connected to queue manager *ssnm*

When IMS tries to make the connection and *MQSeries is not active*, the following messages are sent to the IMS master terminal each time an application makes an MQI call:

```
CSQQ001I IMS/TM imsid not connected to queue manager ssnm.
      Notify message accepted
DFS3607I MQM1      SUBSYSTEM ID EXIT FAILURE, FC = 0286, RC = 08,
JOBNAME = IMSEMPR1
```

If you get DFS3607I messages when you start the connection to IMS or on system startup, this indicates that MQSeries is not available. To prevent a large numbers of messages being generated, you must do one of the following:

1. Start the relevant MQSeries subsystem.
2. Issue a /STOP SUBSYS IMS command so that IMS does not expect to connect to the MQSeries subsystem.

If you do neither, a DFS2607I message and the associated CSQQ001I message are issued each time a job is scheduled in the region and each time a connection request to MQSeries is made by an application.

Thread attachment

In an MPP or IFP region, IMS makes a thread connection when the first application program is scheduled into that region, even if that application program does not make an MQSeries call. In a BMP region, the thread connection is made when the application makes its first MQSeries call (**MQCONN**). This thread is retained for the duration of the region or until the connection is stopped.

For both the message driven and non-message driven regions, the recovery thread cross-reference identifier, *Thread-xref*, associated with the thread is:

PSTid + PSBname

where:

PSTid Partition specification table region identifier
 PSBname Program specification block name

You can use connection IDs as unique identifiers in MQSeries commands; if you do, MQSeries automatically inserts these IDs into any operator message that it generates.

Displaying in-doubt units of recovery

The operational steps used to list and recover in-doubt units of recovery are discussed here for relatively simple cases only. The subject of in-doubt units of recovery is treated in more detail in Chapter 16, “Understanding termination and restart” on page 279.

If MQSeries ends abnormally while connected to IMS, it is possible for IMS to commit or back out work without MQSeries being aware of it. When MQSeries restarts, that work is termed *in doubt*. A decision must be made about the status of the work.

To display a list of in-doubt units of recovery, issue the command:

```
+cpf DISPLAY THREAD(*) TYPE(INDOUBT)
```

MQSeries responds with the following messages:

```
CSQV401I +cpf DISPLAY THREAD REPORT FOLLOWS -
CSQV406I +cpf INDOUBT THREADS - 154
NAME      THREAD-XREF      URID      NID
IMSJ      0002MQSPRG1             IMSJ.5600000000
IMSJ      0001MQSINQ             IMSJ.5700000000
DISPLAY THREAD REPORT COMPLETE
CSQ9022I +cpf CSQVDT ' DISPLAY THREAD' NORMAL COMPLETION
```

where:

NAME The connection name, which is the IMS system ID (the IMSID parameter from the region JCL).

THREAD-XREF The associated thread cross-reference, see “Thread attachment” on page 171.

NID The associated *net-node.number* taken from the IMS recovery token, where *net-node* is the IMS system ID (with trailing blanks suppressed), *number* is the OASN and commit number (leading zeros suppressed).

For a formal explanation of the displayed list, see the description of message CSQV406I in the *MQSeries for OS/390 Messages and Codes* manual.

Recovering in-doubt units of recovery

To recover in-doubt units of recovery, issue this command:

```
+cpf RESOLVE INDOUBT(connection-name) ACTION(COMMIT|BACKOUT)
NID(net-node.number)
```


where:

connection-name The IMS system ID.

ACTION Indicates whether to commit (COMMIT) or back out (BACKOUT) this unit of recovery.

net-node.number The associated net-node.number.

One of the following messages is generated after the RESOLVE INDOUBT command:

```
CSQV414I +cpf THREAD network-id COMMIT SCHEDULED
CSQV415I +cpf THREAD network-id BACKOUT SCHEDULED
```

Resolving residual recovery entries

At given times, IMS builds a list of residual recovery entries (RREs). RREs are units of recovery about which MQSeries could be in doubt. They arise in several situations:

- If MQSeries is not operational, IMS has RREs that cannot be resolved until MQSeries is operational. These RREs are not a problem.
- If MQSeries is operational and connected to IMS, and if IMS backs out the work that MQSeries has committed, the IMS adapter issues message CSQQ010E. If the data in the two systems must be consistent, there is a problem. Resolving it is discussed in “Recovering IMS units of recovery manually” on page 288.
- If MQSeries is operational and connected to IMS, there might still be RREs even though no messages have informed you of this problem. After the MQSeries connection to IMS has been established, you can issue the following IMS command to find out if there is a problem:

```
/DISPLAY OASN SUBSYS sysid
```

To purge the RRE, issue one of the following IMS commands:

```
/CHANGE SUBSYS sysid RESET
/CHANGE SUBSYS sysid RESET OASN nnnn
```

where *nnnn* is the originating application sequence number listed in response to your +cpf DISPLAY command. This is the schedule number of the program instance, giving its place in the sequence of invocations of that program since the last IMS cold start. IMS cannot have two in-doubt units of recovery with the same schedule number.

These commands reset the status of IMS; they do not result in any communication with MQSeries.

Controlling IMS dependent region connections

Controlling IMS dependent region connections involves these activities:

- Connecting from dependent regions
- Region error options
- Monitoring the activity on connections
- Disconnecting from dependent regions

Connecting from dependent regions

The IMS adapter used in the control region is also loaded into dependent regions. A connection is made from each dependent region to MQSeries. This connection is used to coordinate the commitment of MQSeries and IMS work. To initialize and make the connection, IMS does the following:

1. It reads the subsystem member (SSM) from IMS.PROCLIB.

A subsystem member can be specified on the dependent region EXEC parameter. If it is not specified, the control region SSM is used. If the region is never likely to connect to MQSeries, to avoid loading the adapter, specify a member with no entries.

2. It loads the MQSeries adapter.

For a batch message program, the load is not done until the application issues its first messaging command. At that time, IMS tries to make the connection.

For a message-processing program region or IMS fast-path region, the attempt is made when the region is initialized.

Region error options

If MQSeries is not active, or if resources are not available when the first messaging command is sent from application programs, the action taken depends on the error option specified on the SSM entry. The options are:

- R** The appropriate return code is sent to the application.
- Q** The application ends abnormally with abend code U3051. The input message is re-queued.
- A** The application ends abnormally with abend code U3047. The input message is discarded.

Monitoring the activity on connections

A thread is established from a dependent region when an application makes its first successful MQSeries request. Information on connections and the applications currently using them can be displayed by issuing the following command from MQSeries:

```
+cpf DISPLAY THREAD (connection-name)
```

The command produces the following messages:

```

CSQV401I +cpf DISPLAY THREAD REPORT FOLLOWS -
CSQV402I +cpf ACTIVE THREADS -
NAME  ST  A  REQ  THREAD-XREF  USERID  ASID  URID
name  s  *  ct  thread-xref  auth-id  asid  urid
name  s  *  ct  thread-xref  auth-id  asid  urid
DISPLAY ACTIVE REPORT COMPLETE
CSQ9022I +cpf CSQVDT ' DIS THREAD' NORMAL COMPLETION
    
```

For the control region, *thread-xref* is the special value CONTROL. For dependent regions, it is the PSTid concatenated with the PSBname. *auth-id* is either the user field from the job card, or the ID from the OS/390 started procedures table.

For an explanation of the displayed list, see the description of message CSQV402I in the *MQSeries for OS/390 Messages and Codes* manual.

IMS provides a display command to monitor the connection to MQSeries. It shows which program is active on each dependent region connection, the LTERM user name, and the control region connection status. The command is:

```
/DISPLAY SUBSYS name
```

The status of the connection between IMS and MQSeries is shown as one of:

```

CONNECTED
NOT CONNECTED
CONNECT IN PROGRESS
STOPPED
STOP IN PROGRESS
INVALID SUBSYSTEM NAME=name
SUBSYSTEM name NOT DEFINED BUT RECOVERY OUTSTANDING
    
```

The thread status from each dependent region one of the following:

```

CONN
CONN, ACTIVE (includes LTERM of user)
    
```

Disconnecting from dependent regions

To change values in the SSM member of IMS.PROCLIB, you disconnect a dependent region. To do this, you must:

1. Issue the /STOP REGION IMS command
2. Update the SSM member
3. Issue the /START REGION IMS command

Disconnecting from IMS

The connection is ended when either IMS or MQSeries terminates. Alternatively, the IMS master terminal operator can explicitly break the connection by issuing the following IMS command:

```
/STOP SUBSYS sysid
```

The command sends the following message to the terminal that issued it, usually the master terminal operator (MTO):

```
DFS058I STOP COMMAND IN PROGRESS
```

The `/START SUBSYS sysid` IMS command is required to re-establish the connection.

Note: The `/STOP SUBSYS` IMS command will not be completed if an IMS trigger monitor is running.

Controlling the IMS trigger monitor

The IMS trigger monitor (the CSQQTRMN transaction) is described in “The IMS trigger monitor” on page 160.

Starting CSQQTRMN

1. Start a batch oriented BMP running the program CSQQTRMN for each initiation queue you want to monitor.
2. Modify your batch JCL (step 2 in “Setting up the IMS adapter” on page 161) to add a DDname of CSQQUT1 that points to a data set containing the following information:

```
QMGRNAME=q_manager_name      Comment: queue manager name
INITQUEUEUENAME=init_q_name  Comment: initiation queue name
LTERM=lterm                  Comment: LTERM to remove error messages
CONSOLEMESSAGES=YES         Comment: Send error messages to console
```

where:

q_manager_name

The name of the queue manager (if this is blank, the default nominated in CSQQDEFV is assumed)

init_q_name

The name of the initiation queue to be monitored

lterm

The IMS LTERM name for the destination of error messages (if this is blank, the default value is MASTER).

CONSOLEMESSAGES=YES

Requests that messages sent to the nominated IMS LTERM are also sent to the OS/390 console. If this parameter is omitted or misspelled then the default is NOT to send messages to the console.

3. Add a DD name of CSQQUT2 if you want a printed report of the processing of CSQQUT1 input.

Notes:

1. The data set CSQQUT1 is defined with LRECL=80. Other DCB information is taken from the data set. The DCB for data set CSQQUT2 is RECFM=VBA and LRECL=125.
2. You can put only one keyword on each record. The keyword value is delimited by the first blank following the keyword; this means that you can include comments. An asterisk in column 1 means that the whole input record is a comment.
3. If you misspell either of the QMGRNAME or LTERM keywords, CSQQTRMN will use the default for that keyword.
4. Ensure that the subsystem is started in IMS (by the /START SUBSYS command) before submitting the trigger monitor BMP job. If it is not started, your trigger monitor job will terminate, with abend code U3042.

Stopping CSQQTRMN

Once started, CSQQTRMN runs until either the connection between MQSeries and IMS is broken due to one of the following events:

- MQSeries ending
- IMS ending

or an OS/390 STOP jobname command is entered.

Chapter 11. The MQSeries-IMS bridge

This chapter describes the IMS bridge.

This chapter contains the following sections:

- “Introduction to the IMS bridge”
- “Customizing the IMS bridge” on page 181
- “Controlling the IMS bridge” on page 182
- “Security” on page 185

Introduction to the IMS bridge

The MQSeries-IMS bridge is the component of MQSeries for OS/390 that allows direct access from MQSeries applications to applications on your IMS system. The bridge enables *implicit MQSeries API support*. This means that you can re-engineer legacy applications that were controlled by 3270-connected terminals to be controlled by MQSeries messages, without having to rewrite, recompile, or re-link them. The bridge is an IMS *Open Transaction Manager Access (OTMA)* client.

In bridge applications there are no MQSeries calls within the IMS application. The application gets its input using a GET UNIQUE (GU) to the IOPCB and sends its output using an INSERT (ISRT) to the IOPCB. MQSeries applications use the IMS header (the MQIIH structure) in the message data to ensure that the applications can execute as they did when driven by nonprogrammable terminals.

If you are using an IMS application that processes multi-segment messages, note that all segments should be contained within one MQSeries message. The IMS bridge is illustrated in Figure 59 on page 180.

A queue manager can connect to one or more IMS systems, and more than one queue manager can connect to one IMS system. The only restriction is that they must all belong to the same XCF group and must all be in the same sysplex.

What is OTMA?

The IMS OTMA facility is a transaction-based connectionless client/server protocol that runs on version 5.1 of IMS. It functions as an interface for host-based communications servers accessing IMS TM applications through the OS/390 *Cross Systems Coupling Facility (XCF)*.

OTMA enables clients to connect to IMS in a high performance manner enabling the client to support interactions with IMS for a large network or large number of sessions. OTMA is implemented in an OS/390 sysplex environment. Therefore, the domain of OTMA is restricted to the domain of XCF. See the *IMS/ESA Open Transaction Manager Access Guide* for more information.

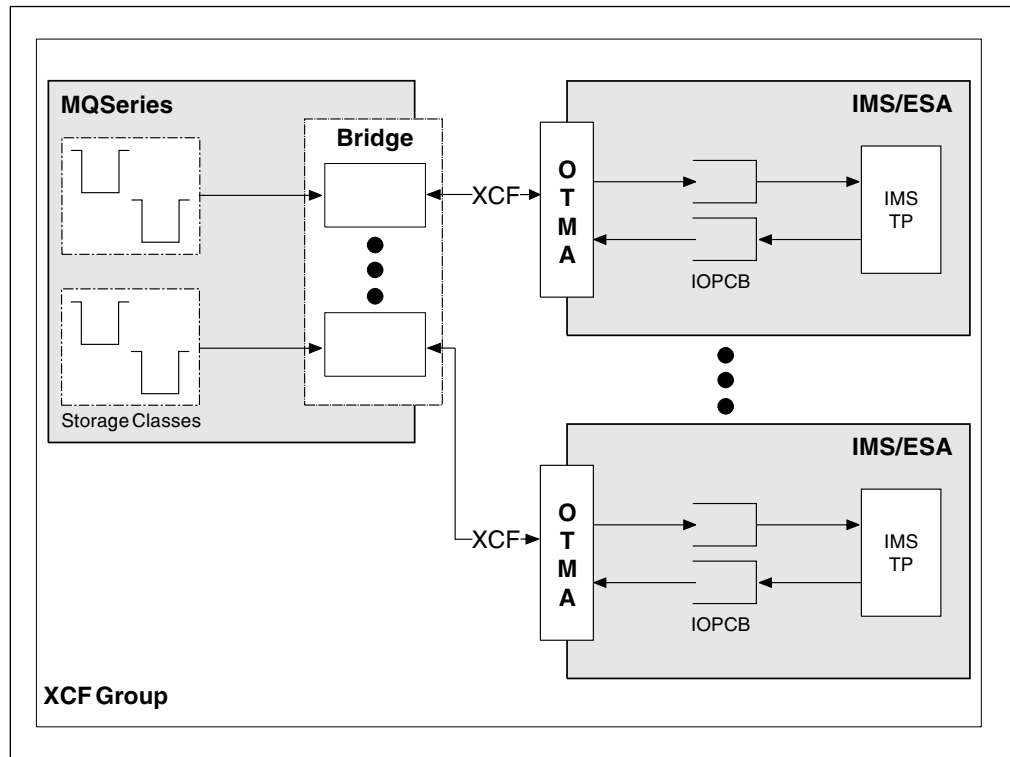


Figure 59. The MQSeries-IMS bridge

Submitting IMS transactions from MQSeries

To submit an IMS transaction that uses the bridge, applications put messages on an MQSeries queue as usual. The messages contain IMS transaction data; they can have an IMS header (the MQIIH structure) or allow the MQSeries-IMS bridge to make assumptions about the data in the message. See the *MQSeries Application Programming Guide* for more information.

MQSeries then puts the message to an IMS queue (it is queued in MQSeries first to enable the use of syncpoints to assure data integrity). The storage class of the MQSeries queue determines whether the queue is an *OTMA queue* (that is, a queue used to transmit messages to the MQSeries-IMS bridge) and the particular IMS partner to which the message data is sent.

Remote queue managers can also start IMS transactions by writing to these OTMA queues on MQSeries for OS/390.

Data returned from the IMS system is written directly to the MQSeries reply-to queue specified in the message descriptor structure (MQMD). (This might be a transmission queue to the queue manager specified in the *ReplyToQMgr* field of the MQMD.)

Customizing the IMS bridge

This section describes what you have to do to customize the MQSeries-IMS bridge, and start sending messages across it to IMS. The bridge is described in “Introduction to the IMS bridge” on page 179.

1. Define the XCF and OTMA parameters for MQSeries.

This step defines the XCF group and member names for your MQSeries system, and other OTMA parameters. MQSeries and IMS must belong to the same XCF group. Use the OTMACON keyword of the CSQ6SYSP macro to tailor these parameters in the system parameter load module.

See “Using CSQ6SYSP” on page 68 for information about this.

2. Define the XCF and OTMA parameters to IMS.

This step defines the XCF group and member names for the IMS system. IMS and MQSeries must belong to the same XCF group.

Add the following parameters to your IMS parameter list, either in your JCL or in member DFSPBxxx in the IMS PROCLIB:

OTMA=Y

This starts OTMA automatically when IMS is started. (This is optional, if you specify OTMA=N you can also start OTMA by issuing the IMS command /START OTMA.)

GRNAME=

This gives the XCF group name.

This is the same as the group name specified in the storage class definition (see Step 3), and in the Group parameter of the OTMACON keyword of the CSQ6SYSP macro.

USERVAR=

This gives the XCF member name of the IMS system.

This is the same as the member name specified in the storage class definition (see Step 3).

If you do not specify a name for USERVAR, the value of APPLID1 is used.

3. Tell MQSeries the XCF group and member name of the IMS system.

This is specified by the storage class of a queue. If you want to send messages across the MQSeries-IMS bridge you need to specify this when you define the storage class for the queue. In the storage class, you need to define the XCF group and the member name of the target IMS system. To do this, either use the MQSeries operations and control panels, or use the MQSC commands as described in the *MQSeries Command Reference* manual.

4. Set up the security that you require.

See “Security considerations for the IMS bridge” on page 449 for information about this.

Controlling the IMS bridge

There are no MQSeries commands to control the MQSeries-IMS bridge.

Start the MQSeries bridge by starting OTMA. Either use the IMS command `/START OTMA`, or start it automatically by specifying `OTMA=YES` in the IMS system parameters. If OTMA is already started, the bridge starts automatically when MQSeries startup has completed. An MQSeries event message is produced when OTMA is started.

Use the IMS command `/STOP OTMA` to stop OTMA communication. When this command is issued, an MQSeries event message is produced.

Controlling IMS connections

IMS provides these operator commands to control and monitor the connection to MQSeries:

/DEQUEUE TMEMBER *tmember* **TPIPE** *tpipe*

Removes messages from a Tpipe, specify `PURGE` to remove all messages or `PURGE1` to remove the first message only.

/DISPLAY OTMA

Displays summary information about the OTMA server and clients, and client status.

/DISPLAY TMEMBER *name*

Displays information about an OTMA client.

/DISPLAY TRACE TMEMBER *name*

Displays information about what is being traced.

/SECURE OTMA

Sets security options.

/START OTMA

Enables communications through OTMA.

/START TMEMBER *tmember* **TPIPE** *tpipe*

Starts the named Tpipe.

/STOP OTMA

Stops communications through OTMA.

/STOP TMEMBER *tmember* **TPIPE** *tpipe*

Stops the named Tpipe.

/TRACE

Controls the IMS trace.

For more information about these commands, see the *IMS/ESA Operator's Reference* manual for the level of IMS that you are using.

IMS command responses are sent to the terminal from which the command was issued. Authorization to issue IMS commands is based on IMS security.

Controlling bridge queues

Issue the following IMS command to stop communicating with the MQSeries system with XCF member name *tmember* through the bridge:

```
/STOP TMEMBER tmember TPIPE ALL
```

Issue the following IMS command to resume communication:

```
/START TMEMBER tmember TPIPE ALL
```

To stop communication with the MQSeries system on a single Tpipe, issue the following IMS command:

```
/STOP TMEMBER tmember TPIPE tpipe
```

One or two Tpipes are created for each active bridge queue, so issuing this command stops communication with the MQSeries queue. Use the following IMS command to resume communication:

```
/START TMEMBER tmember TPIPE tpipe
```

Alternatively, you can alter the attributes of the MQSeries queue to make it get inhibited.

Deleting messages from IMS

A message that is destined for MQSeries via the IMS bridge can be deleted if the Tmember/Tpipe is stopped. To delete one message for the MQSeries system with XCF member name *tmember*, issue the following IMS command:

```
/DEQUEUE TMEMBER tmember TPIPE tpipe PURGE1
```

To delete all the message on the Tpipe, issue the following IMS command:

```
/DEQUEUE TMEMBER tmember TPIPE tpipe PURGE
```

Resynchronizing the IMS bridge

The IMS bridge is automatically restarted whenever MQSeries, IMS, or OTMA are restarted.

The first task undertaken by the IMS bridge is to resynchronize with IMS. This involves MQSeries and IMS checking sequence numbers on every synchronized Tpipe. A synchronized Tpipe is used when persistent messages are sent to IMS from an MQSeries-IMS bridge queue using commit mode 0 (commit-then-send).

If the bridge is unable to resynchronize with IMS at this time, the IMS sense code is returned in message CSQ2023E and the connection to OTMA is stopped. If the bridge is unable to resynchronize with an individual IMS Tpipe at this time, the IMS sense code is returned in message CSQ2025E and the Tpipe is stopped. If a Tpipe has been cold started, the recoverable sequence numbers are automatically reset to 1.

If the bridge discovers mismatched sequence numbers when resynchronizing with a Tpipe, message CSQ2020E is issued. Use the MQSeries command `RESET TPIPE` to initiate resynchronization with the IMS Tpipe. You need to provide the XCF group and member name, and the name of the Tpipe; this information is provided by the message.

You can also specify:

- A new recoverable sequence number to be set in the Tpipe for messages sent by MQSeries, and to be set as the partners receive sequence number. If you do not specify this, the partners receive sequence number is set to the current MQSeries send sequence number
- A new recoverable sequence number to be set in the Tpipe for messages received by MQSeries, and to be set as the partners send sequence number. If you do not specify this, the partners send sequence number is set to the current MQSeries receive sequence number

If there is an unresolved unit of recovery associated with the Tpipe, this is also notified in the message. Use the `RESET TPIPE MQSeries` command to specify whether to commit it or back it out. If you commit the unit of recovery, the batch of messages has already been sent to IMS, and is deleted from the bridge queue. If you back the unit of recovery out, the messages are returned to the bridge queue, to be subsequently sent to IMS.

Commit mode 1 (Send-then-commit) Tpipes are not synchronized.

Considerations for Commit mode 1 transactions

In IMS, commit mode 1 (CM1) transactions send their output replies before syncpoint.

It is possible that a CM1 transaction is unable to send its reply, for example because:

- The Tpipe on which the reply is to be sent is stopped
- OTMA is stopped
- The OTMA client (that is, MQSeries) has gone away
- The reply-to queue and dead-letter queue are unavailable

For all of the above reasons, the IMS application sending the message will pseudo-abend with code U0119. The IMS transaction and program are not stopped in this case.

These reasons often prevent messages being sent into IMS, as well as replies being delivered from IMS. A U0119 abend can occur if:

- The Tpipe, or OTMA, or MQSeries are stopped while the message is in IMS
- IMS replies on a different Tpipe to the incoming message, and that Tpipe is stopped
- IMS replies to a different OTMA client, and that client is unavailable.

Whenever a U0119 abend occurs, both the incoming message to IMS and the reply messages to MQSeries are lost. If the output of a CM0 transaction cannot be delivered for any of the above reasons, it is queued on the Tpipe within IMS.

Security

The `/SECURITY OTMA IMS` command determines the level of security to be applied to **every** MQSeries subsystem that connects to IMS through OTMA. See “Security considerations for the IMS bridge” on page 449 for information about what this should be set to.

Part 5. Operating and administering MQSeries

Chapter 12. Operating MQSeries for OS/390	189
Issuing commands	189
Issuing commands from an OS/390 console or its equivalent	190
Issuing commands from a TSO terminal	191
Issuing commands from the utility program CSQUTIL	191
Starting and stopping MQSeries	191
Before you start MQSeries	191
Starting MQSeries	192
Stopping MQSeries	195
Using the operations and control panels	197
Invoking the operations and control panels	197
Objects and Actions	198
Choosing a queue manager	200
Using the function keys	200
Using the initial panel	202
Defining a local queue	203
Defining other types of objects	207
Working with object definitions	207
Working with namelists	208
Rules for the operations and control panels	209
Chapter 13. Writing programs to administer MQSeries	211
Before you begin	211
Understanding how it all works	212
Using the command server	213
Identifying the queue manager that processes your commands	213
Starting the command server	213
Preparing queues for administration programs	214
Defining the system-command input queue	214
Defining a reply-to queue	214
Opening the system-command input queue	214
Opening a reply-to queue	215
Sending commands to the command server	216
Building a message that includes MQSeries commands	216
Putting messages on the system-command input queue	217
Using MQPUT1 and the system-command input queue	217
Retrieving replies to your commands	218
Waiting for a reply	218
The reply message descriptor	219
Interpreting the replies	220
Examples of commands and their replies	220
Messages from DISPLAY commands for MQSeries objects	222
Command attributes	225
If you do not receive a reply	226
Passing commands using MGCR or MGCRE	226
Chapter 14. Using the MQSeries utilities	227
How to read syntax diagrams	228
MQSeries utility program (CSQUTIL)	229
Invoking the MQSeries utility program	229

Return codes	230
Monitoring the progress of the MQSeries utility program	230
Page set management functions	231
Formatting page sets (FORMAT)	231
Expanding a page set (COPYPAGE)	233
Copying a page set and resetting the log (RESETPAGE)	235
MQSeries command management functions	238
Issuing commands to MQSeries (COMMAND)	238
Producing a list of MQSeries define commands (SDEFS)	243
Queue management functions	245
Syncpoints	245
Copying queues into a data set while the queue manager is running (COPY)	246
Copying queues into a data set while the queue manager is not running (SCOPY)	249
Emptying a queue of all messages (EMPTY)	252
Restoring messages from a data set to a queue (LOAD)	254
The change log inventory utility (CSQJU003)	256
Invoking the CSQJU003 utility	256
Adding information about a data set to the BSDS (NEWLOG)	258
Deleting information about a data set from the BSDS (DELETE)	260
Supplying a password for archive log data sets (ARCHIVE)	261
Controlling the next restart (CRESTART)	262
Setting checkpoint records (CHECKPT)	263
Updating the highest written log RBA (HIGHRBA)	264
The print log map utility (CSQJU004)	265
Invoking the CSQJU004 utility	265
The log print utility (CSQ1LOGP)	266
Invoking the CSQ1LOGP utility	266
Input control parameters	267
Output	268

Chapter 12. Operating MQSeries for OS/390

This chapter describes the basic procedures you can use to operate MQSeries for OS/390. It includes information on how to use the MQSeries operations and control panels. The simplest elements of MQSeries operation are discussed in these sections:

- “Issuing commands”
- “Starting and stopping MQSeries” on page 191
- “Using the operations and control panels” on page 197

Normal operation also requires some more complex tasks. These are discussed in the following chapters:

- Chapter 7, “Operating the CICS adapter” on page 123 describes how to manage and operate the CICS adapter, and control and display the status of the connection.
- Chapter 17, “Understanding the log and the bootstrap data set” on page 299 describes the roles of the log and the log control data sets in preparing for restart and recovery, what happens when MQSeries terminates normally or abnormally, and how to restart it while maintaining data integrity.
- Chapter 18, “Planning for backup and recovery” on page 307 explains how to prepare for recovery.
- Chapter 21, “Example recovery scenarios” on page 337 explains how to recover both at your own site and at a remote site, and how to recover even when the BSDS or log is damaged.

Also, for information about connections to other MQSeries subsystems in distributed queuing applications, see the *MQSeries Intercommunication* manual.

Issuing commands

You can control most of the operational environment of MQSeries using the MQSeries commands. For details of the syntax of the MQSeries commands, see the *MQSeries Command Reference* manual. If you are a suitably authorized user, you can issue MQSeries commands from:

- The initialization input data sets (see “Task 10: Customize the initialization input data sets” on page 52).
- An OS/390 console
- The OS/390 master get command routine, MGCR and MGCRE (SVC 34)
- SDSF
- A TSO console
- The MQSeries utility, CSQUTIL
- The operations and control panels
- A user application, which can be:
 - A CICS program
 - A TSO program
 - An OS/390 batch program

Issuing commands

- An IMS program

See Chapter 13, “Writing programs to administer MQSeries” on page 211 for information about this.

Issuing commands from an OS/390 console or its equivalent

You can issue all MQSeries commands from an OS/390 console or its equivalent. This means you can also issue MQSeries commands from anywhere where you can issue OS/390 commands, such as SDSF or by a program using the MGCR macro.

The maximum amount of data that can be displayed as a result of a command typed in at the console is 32 KB.

Notes:

1. You cannot issue MQSeries commands using the IMS /SSR command format from an IMS terminal. This function is not supported by the IMS adapter.
2. The input field provided by SDSF might not be long enough for some commands, particularly those for channels.

Command prefix strings

Each MQSeries command must be prefixed with a command prefix string (CPF), as shown in Figure 61 on page 192.

Because more than one MQSeries subsystem can run under OS/390, the CPF is used to indicate which MQSeries subsystem processes the command. For example, to start a subsystem called CSQ1, whose CPF is '+cpf', you issue the command '+cpf START QMGR' from the operator console. This CPF must be defined in the subsystem name table (for the subsystem CSQ1). For more information about CPFs, see “Using command prefix strings” on page 42. In the examples, the string '+cpf' is used as the command prefix.

Using the OS/390 console to issue commands

You can type simple commands from the OS/390 console, for example, the DISPLAY command in Figure 60. However, for complex commands or for sets of commands that you issue frequently, the other methods of issuing commands are better.

```
+cpf DISPLAY QUEUE(TRANSMIT.QUEUE.PROD) TYPE(QLOCAL)
```

Figure 60. Issuing a DISPLAY command from the OS/390 console. The command is prefixed by a CPF of '+cpf'.

Command responses

Direct responses to commands are sent to the console that issued the command. MQSeries supports the *Extended Console Support* (EMCS) function available in OS/390, and therefore consoles with 4-byte IDs can be used. Additionally, all commands except START QMGR and STOP QMGR support the use of Command and Response Tokens (CARTs) when the command is issued by a program using the MGCRE macro.

Issuing commands from a TSO terminal

MQSeries uses ISPF to provide a set of operations and control panels. You can use the panels to construct system administrator commands for managing MQSeries objects. Figure 64 on page 202 shows an example of a MQSeries operations and control panel.

Using these panels, you can create MQSeries commands quickly and easily. The panels ensure that you supply all the information that MQSeries needs to complete the task. All the parameters of the command are displayed, together with their possible values.

“Using the operations and control panels” on page 197 describes how to use these panels.

Issuing commands from the utility program CSQUTIL

You can issue commands from a sequential data set using the COMMAND function of the utility program CSQUTIL. This utility transfers the commands to the *system-command input queue* and waits for the response, which is printed together with the original commands in SYSPRINT. For details of this, see the “MQSeries utility program (CSQUTIL)” on page 229.

Starting and stopping MQSeries

Starting and stopping MQSeries is relatively straightforward. When MQSeries stops under normal conditions, its last action is to take a termination checkpoint. This checkpoint, and the logs, give MQSeries the information it needs to restart.

This section discusses the START and STOP commands, and contains a brief overview of start up after an abnormal termination has occurred.

Before you start MQSeries

After you have installed MQSeries, it is defined as a formal OS/390 subsystem. This message appears during any initial program load (IPL) of OS/390:

```
CSQ3110I +cpf CSQ3UR00 - SUBSYSTEM ssnm READY FOR START COMMAND
```

where *ssnm* is the MQSeries subsystem name.

From now on, you can start MQSeries *from any OS/390 console that has been authorized to issue system control commands*; that is, an OS/390 SYS command group. The START command must be issued from the authorized console, and cannot be submitted through JES or TSO.

Starting MQSeries

You start MQSeries by issuing a START QMGR command. However, you cannot successfully use the START command unless you have appropriate authority. See Part 8, “Security” on page 391 for more information about MQSeries security. Figure 61 shows examples of the START command. See the *MQSeries Command Reference* manual for details about the syntax of this command.

```
+cpf START QMGR
+cpf START QMGR PARM(NEWLOG)
```

Figure 61. Starting the MQSeries subsystem from an OS/390 console. The second example specifies a system parameter module name.

Remember that you must prefix an MQSeries command with a command prefix string (CPF).

You cannot run the MQSeries subsystem as a batch job or start it using an OS/390 START command. These methods are likely to start an address space for MQSeries that then abends. You also cannot start MQSeries from the CSQUTIL utility program or a similar user application.

You can, however, start MQSeries from an APF-authorized program by passing a START QMGR command to the OS/390 MGCR or MGCRC (SVC 34) service.

User messages on start-up

When you start MQSeries successfully, it produces a set of start up messages similar to the ones in Figure 62 on page 193.

Notes:

1. If you are starting MQSeries for the first time, the messages are slightly different.
2. If any of the values in message CSQR004I is not zero, message CSQR007I is issued to provide the restart status table.
3. Messages CSQP018I and CSQP019I are issued every time a checkpoint is taken (see “Checkpoint records” on page 301). At checkpoint time, all pages that have not been changed for the two checkpoints are written out to DASD. Message CSQP019I is issued for each buffer pool, giving the number of pages written. You can use this information when balancing page sets in buffer pools.

If you want to suppress these messages, see “Task 18: Suppress information messages” on page 91.
4. There might be periods during startup when no messages are produced; for example, if you are using indexed queues, no messages are produced while the queue indexes are being rebuilt.

```

$HASP373 CSQ1MSTR STARTED
IEF403I CSQ1MSTR - STARTED - TIME=17.06.43
CSQY000I +cpf IBM MQSeries for OS/390 - V2.1
CSQY001I +cpf SUBSYSTEM STARTING, USING PARAMETER MODULE CSQ1ZPRM
CSQY100I +cpf System parameters ...
CSQY101I +cpf CTHREAD=300, IDBACK=20, IDFORE=100, LOGLOAD=16000
CSQY102I +cpf CMDUSER=CSQOPR, QMCCSID=500, ROUTCDE=( 1)
CSQY103I +cpf SMFACT=NO (00000000), SMFSTAT=NO (00000000), STATIME=30
CSQY104I +cpf OTMACON=
(
,
,DFSYDRU0,2147483647,CSQ)
CSQY105I +cpf TRACSTR=( 1), TRACTBL=500
CSQY106I +cpf EXITTCB=8, EXITLIM=30, WLMTIME=30
CSQY110I +cpf Logging parameters ...
CSQY111I +cpf INBUFF=28, OUTBUFF=400, MAXALLC=3, MAXARCH=500
CSQY112I +cpf TWOACTV=YES, TWOARCH=YES, TWOBSDS=YES
CSQY113I +cpf OFFLOAD=YES, WRTHRS=20
CSQY120I +cpf Archive parameters ...
CSQY121I +cpf UNIT=SYSDA, ALCUNIT=BLK, PRIQTY=100, SECQTY=50,
BLKSIZE=24576
CSQY122I +cpf ARCPFX1=ABCD.CSQ1.ARCLOG1, ARCPFX2=ABCD.CSQ1.ARCLOG2,
TSTAMP=NO
CSQY123I +cpf ARCRTN=0, ARCWTOR=YES, ARCWRTC=( 1 ,3 ,4)
CSQY124I +cpf CATALOG=YES, COMPACT=NO, PROTECT=NO, QUIESCE=5
CSQJ127I +cpf SYSTEM TIME STAMP FOR BSDS=1998-07-10 17:05:47.71
CSQJ001I +cpf CSQJW007 CURRENT COPY 1 ACTIVE LOG
DATA SET IS DSNAME=ABCD.CSQ1.LOGCOPY1.DS01,
STARTRBA=000000000000,ENDRBA=00000021BFFF
CSQJ001I +cpf CSQJW007 CURRENT COPY 2 ACTIVE LOG
DATA SET IS DSNAME=ABCD.CSQ1.LOGCOPY2.DS01,
STARTRBA=000000000000,ENDRBA=00000021BFFF
CSQJ099I +cpf LOG RECORDING TO COMMENCE WITH
STARTRBA=000000167000
CSQR001I +cpf RESTART INITIATED
CSQR003I +cpf RESTART...PRIOR CHECKPOINT RBA=000000164B78
CSQR004I +cpf RESTART...UR STATUS COUNTS
IN COMMIT=0, INDOUBT=0, INFLIGHT=0, IN BACKOUT=0
CSQR005I +cpf RESTART...COUNTS AFTER FORWARD RECOVERY
IN COMMIT=0, INDOUBT=0
CSQR006I +cpf RESTART...COUNTS AFTER BACKWARD RECOVERY
INFLIGHT=0, IN BACKOUT=0
CSQR002I +cpf RESTART COMPLETED
CSQP018I +cpf CSQPBCWK CHECKPOINT STARTED FOR ALL BUFFER POOLS
+cpf DISPLAY THREAD(*) TYPE(INDOUBT)
CSQP019I +cpf CSQP1DWP CHECKPOINT COMPLETED FOR
BUFFER POOL 2, 2 PAGES WRITTEN
CSQP019I +cpf CSQP1DWP CHECKPOINT COMPLETED FOR
BUFFER POOL 3, 2 PAGES WRITTEN
CSQP019I +cpf CSQP1DWP CHECKPOINT COMPLETED FOR
BUFFER POOL 1, 5 PAGES WRITTEN
CSQP019I +cpf CSQP1DWP CHECKPOINT COMPLETED FOR
BUFFER POOL 0, 25 PAGES WRITTEN
CSQV401I +cpf DISPLAY THREAD REPORT FOLLOWS -
CSQV402I +cpf NO INDOUBT THREADS FOUND
CSQ9022I +cpf CSQVDT 'DISPLAY THREAD' NORMAL COMPLETION
CSQY022I +cpf QUEUE MANAGER INITIALIZATION COMPLETE
CSQ9022I +cpf CSQYASCP 'START QMGR' NORMAL COMPLETION

```

Figure 62. MQSeries startup messages for subsystem CSQ1. The command prefix string is +cpf.

Start options

When you start a queue manager, a special routine called the system parameter module is invoked. You can specify the name of a system parameter module if you use the PARM keyword. A system parameter module provides information specified when the queue manager was customized. In Figure 62 on page 193, the user message CSQY0011 indicates the name of the system parameter module that was used, in this case, CSQ1ZPRM. For more information about this, see “Task 13: Tailor your system parameter module” on page 67.

You can also use the ENVPARM option to substitute one or more parameters in the JCL procedure for the queue manager.

For example, you can update your MQSeries startup procedure, so that the DDname CSQINP2 is a variable. This means that you can change the CSQINP2 DDname without changing the startup procedure. This is very useful for implementing changes, providing backouts for operators, and so on.

Suppose your start-up procedure, for queue manager CSQ1, looked like this:

```
//CSQ1MSTR PROC INP2=NORM
//MQMESA EXEC PGM=CSQYASCP
//STEPLIB DD DISP=SHR,DSN=thqua1.SCSQANLE
// DD DISP=SHR,DSN=thqua1.SCSQAUTH
//BSDS1 DD DISP=SHR,DSN=myqua1.BSDS01
//BSDS2 DD DISP=SHR,DSN=myqua1.BSDS02
//CSQP0000 DD DISP=SHR,DSN=myqua1.PSID00
//CSQP0001 DD DISP=SHR,DSN=myqua1.PSID01
//CSQP0002 DD DISP=SHR,DSN=myqua1.PSID02
//CSQP0003 DD DISP=SHR,DSN=myqua1.PSID03
//CSQINP1 DD DISP=SHR,DSN=myqua1.CSQINP(CSQ1INP1)
//CSQINP2 DD DISP=SHR,DSN=myqua1.CSQINP(CSQ1&INP2.)
//CSQOUT1 DD SYSOUT=*
//CSQOUT2 DD SYSOUT=*
```

If you then start the your queue manager with the command:

```
+cpf START QMGR
```

the CSQINP2 actually used is a member called CSQ1NORM.

However, suppose you are putting a new suite of programs into production so that the next time you start queue manager CSQ1, the CSQINP2 definitions are to be taken from member CSQ1NEW. To do this, you would start MQSeries with this command:

```
+cpf START QMGR ENVPARM(' INP2=NEW')
```

and CSQ1NEW would be used instead of CSQ1NORM. Note that OS/390 limits the KEYWORD=value specifications for symbolic parameters (as in INP2=NEW) to 48 characters.

Starting after an abnormal termination

MQSeries automatically detects whether restart follows a normal shutdown or an abnormal termination.

Starting MQSeries after it abends is different from starting it after the +cpf STOP QMGR command has been issued. After +cpf STOP QMGR, the system finishes its work in an orderly way and takes a termination checkpoint before stopping. When you restart MQSeries, it uses information from the system checkpoint and recovery log to determine the system status at shutdown.

However, if MQSeries abends, it terminates without being able to finish its work or take a termination checkpoint. When you restart MQSeries after an abend, it refreshes its knowledge of its status at termination using information in the log, and notifies you of the status of various tasks. Normally, the restart process resolves all inconsistent states. But, in some cases, you must take specific steps to resolve inconsistencies. For a discussion of the causes of inconsistencies, and how you can prepare to recover from them, see Chapter 16, “Understanding termination and restart” on page 279.

Stopping MQSeries

Before stopping MQSeries, all MQSeries-related write-to-operator-with-reply (WTOR) messages must receive replies, for example, getting log requests. Each of the commands in Figure 63 terminates a running MQSeries subsystem.

```
+cpf STOP QMGR
+cpf STOP QMGR MODE(QUIESCE)
+cpf STOP QMGR MODE(FORCE)
+cpf STOP QMGR MODE(RESTART)
```

Figure 63. Stopping MQSeries

The command +cpf STOP QMGR defaults to +cpf STOP QMGR MODE(QUIESCE).

In QUIESCE mode, MQSeries does not allow any new connection threads to be created, but allows existing threads to continue; it terminates only when all threads have ended. Applications can request to be notified in the event of the queue manager quiescing. Therefore, use the QUIESCE mode where possible so that applications that have requested notification have the opportunity to disconnect. See the *MQSeries Application Programming Guide* for details.

If MQSeries does not terminate in a reasonable time in response to a +cpf STOP QMGR MODE(QUIESCE) command, use the +cpf DISPLAY THREAD(*) TYPE(ACTIVE) command to determine whether any connection threads exist, and take the necessary steps to terminate the associated applications. If there are no threads then issue a +cpf STOP QMGR MODE(FORCE) command.

The +cpf STOP QMGR MODE(QUIESCE) and +cpf STOP QMGR MODE(FORCE) commands deregister MQSeries from the MVS Automatic Restart Manager (ARM), preventing ARM from restarting the queue manager automatically.

Starting and stopping MQSeries

The +cpf STOP QMGR MODE(RESTART) command works in the same way as the +cpf STOP QMGR MODE(FORCE) command, except that it does not deregister MQSeries from ARM. This means that the queue manager is eligible for immediate automatic restart.

If the MQSeries subsystem is not registered with ARM, the STOP QMGR MODE(RESTART) command is rejected and the following message sent to the OS/390 console:

```
CSQY205I  ARM element arm-element is not registered
```

If this message is not issued, the queue manager is restarted automatically. For more information about ARM, see “Using the OS/390 Automatic Restart Manager (ARM)” on page 294.

Do not cancel the MQSeries address space unless +cpf STOP QMGR MODE(FORCE) does not cause MQSeries to terminate.

If MQSeries is stopped by either canceling the address space or by using the command +cpf STOP QMGR MODE(FORCE), consistency is maintained with connected CICS or IMS systems. Resynchronization of resources is started when MQSeries restarts and is completed when the connection to the CICS or IMS system is established. See Chapter 16, “Understanding termination and restart” on page 279.

Note: When you stop your MQSeries subsystem, you might find message IEF352I is issued. OS/390 issues this message if it detects that failing to mark the address space as unusable would lead to an integrity exposure. You can ignore this message.

Stop messages

After issuing a +cpf STOP QMGR command, you get the messages CSQY009I and CSQY002I, for example:

```
CSQY009I +cpf ' STOP QMGR' COMMAND ACCEPTED FROM  
USER(userid), STOP MODE(FORCE)  
CSQY002I +cpf SUBSYSTEM STOPPING
```

Where *userid* is the user ID that issued the +cpf STOP QMGR command, and the MODE parameter depends on that specified in command.

When the STOP command has completed successfully, these messages are displayed on the OS/390 console:

```
CSQ9022I +cpf CSQYASCP ' STOP QMGR' NORMAL COMPLETION  
CSQ3104I +cpf CSQ3EC0X - TERMINATION COMPLETE
```


If you are using ARM, the following message is also displayed if you did not specify MODE(RESTART):

```
CSQY204I +cpf ARM DEREGISTER for element arm-element type
arm-element-type successful
```

You cannot restart MQSeries until the following message has been generated:

```
CSQ3100I +cpf CSQ3EC0X - SUBSYSTEM ssnm READY FOR START COMMAND
```

Using the operations and control panels

You can use the MQSeries operations and control panels to perform administration tasks on MQSeries objects. You use these panels to run commands for defining, displaying, altering, or deleting MQSeries objects.

The operations and control panels support the system control commands for distributed queuing without CICS (for example, to start a channel or a TCP/IP listener), for clustering, and for security. They also enable you to display information about threads and page set usage. The other system control commands are not available through the panels. These commands must be issued explicitly using one of the other methods, see “Issuing commands” on page 189.

Note: You cannot issue the MQSeries commands directly from the command line in the panels.

To use the operations and control panels, you must have the correct security authorization; see “Security checking for the operations and control panels” on page 397 for information.

Invoking the operations and control panels

If the ISPF/PDF primary options menu has been updated for MQSeries, you can access the MQSeries operations and control panels from that menu. For details about updating the menu, see “Updating the ISPF menu” on page 89.

You can access the MQSeries operations and control panels from the TSO command processor panel (usually option 6 on the ISPF/PDF primary options menu). The name of the exec that you run to do this is CSQOREXX. It has two parameters; *thlqual* is the high-level qualifier for the MQSeries libraries to be used, and *langletter* is the letter identifying the national language libraries to be used (for example, E for US English). The parameters can be omitted if the MQSeries libraries are permanently installed in your ISPF setup. Alternatively, you can issue CSQOREXX from the TSO command line.

These panels are designed to be used by operators and administrators with a minimum of formal training. Read these instructions with the panels running and try out the different tasks suggested.

Note: While using the panels, temporary dynamic queues with names of the form SYSTEM.CSQOREXX.★ will be created.

Objects and Actions

The operations and control panels offer you many different types of object and a number of actions that you can perform on them. The actions are listed on the initial panel and enable you to manipulate the objects and display information about them. These objects include all the MQSeries objects, together with some extra ones. The objects fall into five categories.

1. Queues, Processes, Namelists, and Storage classes

These are the basic MQSeries objects. There can be many of each type. They can be defined and deleted, and have attributes that can be displayed and altered, using the DEFINE, DELETE, DISPLAY, and ALTER actions.

This category consists of the following objects:

QLOCAL	Local queue
QREMOTE	Remote queue
QALIAS	Alias queue for indirect reference to a queue
QMODEL	Model queue for defining queues dynamically
QUEUE	Any of QLOCAL, QREMOTE, QALIAS, or QMODEL
PROCESS	Information about an application to be started when a trigger event occurs
NAMELIST	List of names, such as queues or clusters
STGCLASS	Storage class

2. Channels

They are used for distributed queuing (not for the CICS mover). There can be many of each type, and they can be defined, deleted, displayed, and altered. They also have other functions available; the PERFORM action provides reset, ping, or resolve channel functions.

This category consists of the following objects:

CHANNEL	Any type of channel below
CHLSENDER	Sender channel
CHLSERVER	Server channel
CHLRECEIVER	Receiver channel
CHLREQUESTER	Requester channel
CHLCLUSRCVR	Cluster-receiver channel
CHLCLUSSDR	Cluster-sender channel
CHLSVRCONN	Server-connection channel (PERFORM not allowed)
CHLCLNTCONN	Client-connection channel (PERFORM, START, STOP not allowed)

3. Cluster objects

They are created automatically for queues and channels that belong to a cluster. The base queue and channel definitions can be on another queue manager. There can be many of each type, and names can be duplicated. They can only be displayed, using the DISPLAY action.

This category consists of the following objects:

CLUSQ	Cluster queue, created for a queue that belongs to a cluster
CLUSCHL	Cluster channel, created for a channel that belongs to a cluster
CLUSQMGR	Cluster queue manager, the same as a cluster channel but identified instead by its queue manager name

Cluster channels and cluster queue managers do have the PERFORM, START and STOP actions, but only indirectly through the DISPLAY action.

4. Queue Manager and Security

These have a single instance. They have attributes that can be displayed and altered (using the DISPLAY and ALTER actions), and have other functions available using the PERFORM action.

This category consists of the following objects:

MANAGER Queue manager - the PERFORM action provides suspend and resume clustering functions

SECURITY Security functions - the PERFORM action provides refresh and reverify functions

5. System

A collection of other functions.

This category consists of the following objects:

SYSTEM System functions

CONTROL (Synonym for SYSTEM)

The functions available are:

DISPLAY Display distributed queuing, threads, or page set usage information

PERFORM Refresh or reset clustering

START, STOP Start or stop the channel initiator or listeners

The actions that you can perform for each type of object are described in the following table:

<i>Table 15 (Page 1 of 2). Valid operations and control panels actions for MQSeries objects</i>							
Object	Alter	Define	Delete	Display	Perform	Start	Stop
CHANNEL	√	√	√	√	√	√	√
CHLCLNTCONN	√	√	√	√			
CHLCLUSRCVR	√	√	√	√	√	√	√
CHLCLUSSDR	√	√	√	√	√	√	√
CHLRECEIVER	√	√	√	√	√	√	√
CHLREQUESTER	√	√	√	√	√	√	√
CHLSENDER	√	√	√	√	√	√	√
CHLSERVER	√	√	√	√	√	√	√
CHLSVRCONN	√	√	√	√		√	√
CLUSCHL				√	√(1)	√(1)	√(1)
CLUSQ				√			
CLUSQMGR				√	√(1)	√(1)	√(1)
MANAGER	√			√	√		
NAMELIST	√	√	√	√			
PROCESS	√	√	√	√			
QALIAS	√	√	√	√			
QLOCAL	√	√	√	√			
QMODEL	√	√	√	√			
QREMOTE	√	√	√	√			

Operations and control panels

Table 15 (Page 2 of 2). Valid operations and control panels actions for MQSeries objects

Object	Alter	Define	Delete	Display	Perform	Start	Stop
QUEUE	√	√	√	√			
SECURITY	√			√	√		
STGCLASS	√	√	√	√			
SYSTEM/CONTROL				√	√	√	√

Note:
1. Via Display

Choosing a queue manager

While you are viewing the initial panel, you are not connected to any queue manager. However, as soon as you press Enter, you are connected to the queue manager shown in “Connect to queue manager”. After this, any requests that you make are directed to that queue manager.

To change the queue manager, press function key F6 in the initial panel and then complete the **Change the Queue Manager** secondary window. From this window, you can also specify a target queue manager that is different from the one you connect to. If you do, any further requests you make are directed to that queue manager.

Queue manager defaults

When you first use the operations and control panels, the “Connect to queue manager” and “Target queue manager” fields are normally blank. This means after you press Enter in the initial panel, you are using the default queue manager for batch applications. This is defined in CSQBDEFV (see “Task 15: Set up Batch, TSO, and RRS adapters” on page 87). If you return to the initial panel after having made some requests, you find fields filled in with the actual name. Similarly, if you set the “Connect to queue manager” field to blanks, any requests that follow are processed by the default queue manager for batch applications.

Using the function keys

To use the panels, you must use the function keys or enter the equivalent commands in the command area. The function keys have special settings for MQSeries. These settings can optionally be displayed on the panels, as shown in Figure 64 on page 202.

The function key settings in the operations and control panels conform to CUA® standards. Although you can change the key setting through normal ISPF procedures, you are not recommended to do so.

Displaying the function key menu

Type PFSHOW in the command area of any operations and control panel and then press Enter.

Use the command PFSHOW OFF to remove the display of the function key settings from the panels.

Note: PFSHOW causes the function key settings to be displayed on any other logical ISPF screens that you have. The keys remain displayed when you leave the operations and control panels.

Getting things done

Press Enter to carry out the action requested on a panel. The information from the panel is sent to the queue manager for processing.

Each time you press Enter in the panels, MQSeries generates one or more operator messages. If the operation was successful, you get a confirmation message, CSQ9022I, otherwise you get some error messages.

Displaying MQSeries user messages

Press function key F10 in any panel to see the MQSeries user messages.

Ignoring what you have done

On the initial panel, both F3 and F12 exit the operations and control panels and return you to ISPF. No information is sent to the queue manager.

On any other panel, press function keys F3 or F12 to leave the current panel *ignoring any data you have typed since last pressing Enter*. Again, no information is sent to the queue manager.

- F3 takes you straight back to the initial panel.
- F12 takes you back to the previous panel.

Getting help

Each panel has help panels associated with it. The help panels use the ISPF protocols:

- Press function key F1 on any panel to see general help (extended help) about the task.
- Press function key F1 with the cursor on any field to see specific help about that field.
- Press function key F5 from any field help panel to get the general help.
- Press function key F3 to return to the base panel, that is, the panel from which you pressed function key F1.
- Press function key F6 from any help panel to get help about the function keys.

If the help information carries on into a second or subsequent pages, a **More** indicator is displayed in top right of the panel. Use these function keys to navigate through the help pages:

- F11 to get to the next help page (if there is one).
- F10 to get back to the previous help page (if there is one).

Using the initial panel

Figure 64 shows the panel that is displayed when you start a panel session.

```
IBM MQSeries for OS/390 - Main Menu

Complete fields. Then press Enter.

Action . . . . . _      1. Display  5. Perform
                        2. Define  6. Start
                        3. Alter   7. Stop
                        4. Delete

Object type . . . . . _____ +
Name . . . . . _____
Like . . . . . _____

Connect to queue
manager . . . . . : _____
Target queue manager : _____
Response wait time . : _____ seconds

(C) Copyright IBM Corporation 1993,1999. All rights reserved.

Command ==> _____
F1=Help      F2=Split    F3=Exit     F4=Prompt   F6=QueueMgr F9=Swap
F10=Messages F12=Cancel
```

Figure 64. The MQSeries operations and control initial panel

From this panel you can:

- Choose the local queue manager you want and whether you want the commands issued on that queue manager or on some remote queue manager. Press function key F6 if you need to change the queue manager name. For more information, see “Choosing a queue manager” on page 200.
- Select the action you want to perform by typing in the appropriate number in the **Action** field.
- Specify the object type that you want to work with. Press function key F4 for a list of object types if you are not sure what they are, or see “Objects and Actions” on page 198.
- Display a list of objects of the type specified. Type in an asterisk (*) in the **Name** field and press Enter to display a list of objects (of the type specified) that have already been defined on this subsystem. You can then select one or more objects to work with in sequence. Figure 65 on page 203 shows a list of queues produced in this way.
- Define an object with the same attributes as an existing object. See “Defining a local queue using the Like field” on page 207.

```

                                List Queues                                ROW 1 OF 12

Type action codes. Then press Enter.
1=Display 2=Define like 3=Alter 4=Delete

      Name                                                    Type
-   CICS01.INITQ                                             QLOCAL
-   PROTO.APPL                                              QLOCAL
-   PROTO.TRIG                                              QLOCAL
-   LOCAL.QUEUE                                             QLOCAL
-   SYSTEM.CHANNEL.SEQNO                                     QLOCAL
-   SYSTEM.COMMAND.INPUT                                    QLOCAL
-   SYSTEM.COMMAND.REPLY.MODEL                             QMODEL
-   SYSTEM.DEFAULT.ALIAS.QUEUE                             QALIAS
-   SYSTEM.DEFAULT.LOCAL.QUEUE                             QLOCAL
-   SYSTEM.DEFAULT.MODEL.QUEUE                             QMODEL
-   SYSTEM.DEFAULT.REMOTE.QUEUE                            QREMOTE
-   TRANSMIT.QUEUE.PROD                                     QLOCAL
                                ***** End of list *****

Command ==>
F1=Help      F2=Split      F3=Exit      F5=Refresh   F6=Clinfno   F7=Bkwd
F8=Fwd       F9=Swap       F10=Messages F12=Cancel

```

Figure 65. Listing queues

Defining a local queue

To define a local queue object from the operations and control panels, there are several panels to complete. When you have completed *all* the panels and you are satisfied that the attributes are correct, you press Enter to send your definition to the queue manager, which then creates the actual queue.

Starting from the initial panel, complete these fields:

Field	Value
Action	2 (Define)
Object type	QLOCAL
Name	QUEUE.YOU.LIKE

Press Enter to display the **Define a Local Queue** panel as shown in Figure 66 on page 204. The queue name displayed is the name you specified in the previous panel. You can type in your own description in the **Description** field. Complete the other fields as required. For example, type **Y** in the **Put enabled** field if suitably authorized applications can put messages on this queue.

```

                                Define a Local Queue
                                Complete fields, then press F8 for further fields, or Enter to define queue.

                                                                                               More:  +

Queue name . . . . . QUEUE.YOU.LIKE
Description . . . . . Default local queue definition
-----
Put enabled . . . . . Y  Y=Yes,N=No
Get enabled . . . . . Y  Y=Yes,N=No
Usage . . . . . N  N=Normal,X=XmitQ
Storage class . . . . . SYSTEM

Command ==>
F1=Help      F2=Split   F3=Exit    F6=C!usinfo  F7=Bkwd     F8=Fwd
F9=Swap      F10=Messages F12=Cancel
```

Figure 66. Defining a local queue - first panel

You get field help by moving the cursor into a field and pressing function key F1. Field help provides information about the values that can be used for each attribute.

When you have completed the first panel, press function key F8 to display the second panel, see Figure 67 on page 205.

Hints:

- 1. Do *not* press Enter at this stage, otherwise the queue will be created before you have a chance to complete the remaining fields.
- 2. Do not press function key F3 or F12 either, or the data you typed will be lost.
- 3. If you do press Enter prematurely, do not worry; you can always alter your definition later on.

Press function key F8 repeatedly to see and complete the remaining panels, including the trigger definition, event control, and backout reporting panels.


```

                                Define a Local Queue
Press F7 or F8 to see other fields, or Enter to define queue.

                                                                    More: - +

Default persistence . . . . . N  Y=Yes,N=No
Default priority . . . . . 5   0 - 9
Message delivery sequence . . P  P=Priority,F=FIFO
Permit shared access . . . . . Y  Y=Yes,N=No
Default share option . . . . . S  E=Exclusive,S=Shared
Index type . . . . . N       N=None,M=MsgId,C=CorrelId,T=MsgToken
Maximum queue depth . . . . . 10000  0 - 999999999
Maximum message length . . . 1000000  0 - 4194304
Retention interval . . . . . 999999999  0 - 999999999 hours

Cluster name . . . . . _____
Cluster namelist name . . . . _____
Default bind . . . . . 0   0=Open,N=Notfixed

Command ==> _____
F1=Help      F2=Split   F3=Exit    F6=Clusinfo F7=Bkwd    F8=Fwd
F9=Swap      F10=Messages F12=Cancel
    
```

Figure 67. Defining a local queue - second panel

```

                                Define a Local Queue
Press F7 or F8 to see other fields, or Enter to define queue.

                                                                    More: - +

Trigger Definition

Trigger type . . . . . F  F=First,E=Every,D=Depth,N=None

Trigger set . . . . . N  Y=Yes,N=No
Trigger message priority . 0  0 - 9
Trigger depth . . . . . 1      1 - 999999999
Trigger data . . . . . _____

Process name . . . . . _____
Initiation queue . . . . . _____

Command ==> _____
F1=Help      F2=Split   F3=Exit    F6=Clusinfo F7=Bkwd    F8=Fwd
F9=Swap      F10=Messages F12=Cancel
    
```

Figure 68. Defining a local queue - trigger conditions

```

                                Define a Local Queue
Press F7 or F8 to see other fields, or Enter to define queue.

                                                                 More: - +
Event Control

  Queue full . . . . . E  E=Enabled,D=Disabled
  Upper queue depth . . . . D  E=Enabled,D=Disabled
  Threshold . . . . . 80  0 - 100 %

  Lower queue depth . . . . D  E=Enabled,D=Disabled
  Threshold . . . . . 40  0 - 100 %

  Service interval . . . . N  H=High,0=OK,N=None
  Interval . . . . . 999999999 0 - 999999999 milliseconds

Command ==> _____
F1=Help      F2=Split   F3=Exit    F6=C!usinfo F7=Bkwd    F8=Fwd
F9=Swap      F10=Messages F12=Cancel

```

Figure 69. Defining a local queue - event control

```

                                Define a Local Queue
Press F7 to see previous fields, or Enter to define queue.

                                                                 More: -
Backout Reporting

  Backout threshold . . . . . 0          0=No backout reporting

  Harden backout counter . . N  Y=Yes,N=No
  Backout requeue name . . . _____

Command ==> _____
F1=Help      F2=Split   F3=Exit    F6=C!usinfo F7=Bkwd    F8=Fwd
F9=Swap      F10=Messages F12=Cancel

```

Figure 70. Defining a local queue - backout reporting

When your local queue definition is complete

When your definition is complete, press Enter to send the information to the queue manager for processing. The queue manager creates the queue according to the definition you have supplied. If you do not want the queue to be created, press function key F3 to exit and cancel the definition.

Defining a local queue using the Like field

You can use the **Like** field to define a local queue that has the same attributes as an existing local queue. This field is ignored for actions other than **Define**.

To do this, starting from the initial panel, use these values for the fields specified:

<i>Field</i>	<i>Value</i>
Action	2 (Define).
Object type	QLOCAL
Name	TRANSMIT.QUEUE.NEW This is the name of the queue you are defining.
Like	TRANSMIT.QUEUE.PROD This is the name of an existing queue.

When you press Enter, the attributes of the new queue, which are initially those of the queue named in the **Like** field, are displayed. You can modify these attributes as required.

You can use this method to define any type of object providing that the object specified in the **Like** field exists and it is the same kind of object as that specified in the **Object type** field.

Defining other types of objects

You can define objects other than just local queues. To do this, you start from the initial panel and complete these fields:

<i>Field</i>	<i>Value</i>
Action	2 (Define).
Object type	QALIAS, NAMELIST, PROCESS, CHANNEL, and so on.
Name	The name of the object you are defining.
Like	Leave blank or enter the name of an existing object of the same type.

Press Enter to display the corresponding DEFINE panels. Complete the fields as required and then press Enter again to send the information to the queue manager.

Like defining a local queue, defining a model queue requires several panels to be completed. Defining a namelist requires some additional work, as described in “Working with namelists” on page 208. A single panel is required to define each of the remaining objects.

Working with object definitions

Once an object has been defined, you can specify an action in the **Action** field, to alter, display, or delete an object definition. In each case, you start from the initial panel, where you specify the object you are working with by completing the **Object type** and **Name** fields.

Altering an object definition

To alter an object definition, specify action 3 and press Enter to see the ALTER panels. These panels are very similar to the DEFINE panels. You can alter any values you want. When your changes are complete, press Enter to send the information to the queue manager.

Displaying an object definition

If you just want to see the details of an object without being able to change them, specify action 1 and press Enter to see the DISPLAY panels. Again, these panels are similar to the DEFINE panels except that you cannot change any of the fields.

Deleting an object

To delete an object, specify action 4 and press Enter to see the DELETE panels. Again these are similar to the DEFINE panels except that when you press Enter, you are asked to confirm your request. If you press function key F3 or F12, the request is canceled. If you press Enter, the request is confirmed and passed to the queue manager. The object you specified is then deleted.

Note: You cannot delete most types of channel object unless the channel initiator is started.

Working with namelists

Start from the initial panel and complete these fields:

<i>Field</i>	<i>Value</i>
Action	1 to 4, as required
Object type	NAMELIST
Name	Specify the fully qualified name of a namelist (for example: MY.CSQ.NAMES)
Like	You can type the name of an existing namelist if you are defining a new namelist

Then press Enter. The panel you are shown lists the contents of that namelist.

For the actions DEFINE or ALTER, you must press function key F11 to add names to the list or to change the names in the list. This involves working with the ISPF editor and all the normal ISPF edit commands are available. Each name in the namelist must be entered on a separate line.

Note: When you use the ISPF editor in this way, the function key settings are the normal ISPF settings, and **not** those used by the other operations and control panels.

If you need to specify lowercase queue names, specify CAPS(OFF) on the editor panel command line. When you do this, all the namelists that you edit in the future are in lowercase until you specify CAPS(ON).

When you have finished editing the namelist, press function key F3 to end the ISPF edit session. Then press Enter to send the changes to the queue manager.

Attention: If you do not press Enter at this stage but press function key F3 instead, you lose any updates that you have typed in.

Rules for the operations and control panels

The *MQSeries Command Reference* manual defines the general rules for MQSeries character strings and names. However, there are some rules that apply only to the operations and control panels:

- Do not enclose strings, for example descriptions, in single or double quotes.
- If you need to use a quote mark in a description or other text field, for example:
This is Maria's queue
use just one quote. The panel processor doubles them for you to pass them to MQSeries. However, if it has to truncate your data to do this, it will do so.
- You can use uppercase or lowercase characters in most fields, and they are translated to uppercase characters when you press Enter. The exceptions are:
 - Storage class which must start with uppercase A through Z and be followed by uppercase A through Z or 0 through 9 characters.
 - The following fields, which are not translated:
 - Application ID
 - Description
 - Environment data
 - Object names (but if you use a lowercase object name, you might not be able to enter it at an OS/390 console)
 - Remote system name
 - Trigger data
 - User data
- In names, leading blanks and leading underscores are ignored. Therefore, you cannot have object names beginning with blanks or underscores.
- Underscores are used to show the extent of blank fields. When you press Enter, trailing underscores are replaced by blanks.
- Many description and text fields are presented in multiple parts, each part being handled by MQSeries independently. This means that trailing blanks *are retained* and the text is not contiguous.

Blank fields

When you specify the **define** action for an MQSeries object, each field on the define panel contains a value. See the general help (extended help) for the display panels for information on where MQSeries gets the values. If you type over a field with blanks, and blanks are not allowed, MQSeries puts the installation default value in the field.

When you specify the **Alter** action for an MQSeries object, each field on the alter panel contains the current value for that field. If you type over a field with blanks, and blanks are not allowed, the ALTER command fails and an error message is displayed.

Chapter 13. Writing programs to administer MQSeries

General-use programming interface

This chapter contains hints and guidance to enable you to issue MQSeries commands from an MQSeries application program.

It contains these sections:

- “Before you begin”
- “Understanding how it all works” on page 212
- “Using the command server” on page 213
- “Preparing queues for administration programs” on page 214
- “Sending commands to the command server” on page 216
- “Putting messages on the system-command input queue” on page 217
- “Retrieving replies to your commands” on page 218
- “Interpreting the replies” on page 220
- “If you do not receive a reply” on page 226
- “Passing commands using MGCR or MGCRE” on page 226

Note: In this chapter, the MQI calls are described using C-language notation. For typical invocations of the calls in the COBOL, PL/I, and assembler languages, see the *MQSeries Application Programming Reference* manual.

Before you begin

Before you can write an application program to issue MQSeries commands, you must be familiar with:

1. Issuing MQSeries commands and the command syntax. See the *MQSeries Command Reference* manual for more information.
2. Writing application programs that use the MQI.

This includes:

- Connecting to a queue manager using the **MQCONN** call.
- Opening a queue using **MQOPEN**.
- Opening a dynamic queue using **MQOPEN** and specifying the name of a model queue.
- Putting messages on a queue using **MQPUT** and **MQPUT1**.
- Getting messages from a queue using **MQGET**.

You need to know about the messages including:

- The message descriptor structure.
- What the persistence attribute of a message means.
- The types of MQSeries messages, in particular, *request messages* and the *reply messages* they generate.

You can find all this information in the *MQSeries Application Programming Guide* and the *MQSeries Application Programming Reference* manual.

Understanding how it all works

3. User messages.

These messages are generated by MQSeries to show the success or failure of, and the responses to, MQSeries commands. Each message is identified by an ID that contains the characters CSQ, for example, CSQN205I. For more information, see the *MQSeries for OS/390 Messages and Codes* manual.

If you want your MQSeries commands to be run on a remote queue manager, see the *MQSeries Intercommunication* manual.

MQSeries can also be set up to perform security checks. For example, to ensure that a user is authorized to issue a particular command for a particular resource. For more information, see Part 8, "Security" on page 391.

Understanding how it all works

In outline, the procedure for issuing commands from an application program is quite simple:

1. You build an MQSeries command into a type of MQSeries message called a *request message*.
2. You put (**MQPUT**) this message onto a special queue called the system-command input queue. The MQSeries command processor runs the command.
3. You retrieve (**MQGET**) the results of the command as *reply messages* on the reply-to queue. These messages contain the user messages that you need to determine whether your command was successful and, if it was, what the results were.

Then it is up to your application program to process the results.

Using the command server

The command server is an MQSeries component that works with the command processor component. The command server reads request messages from the system-command input queue, verifies them, and passes the valid ones as commands to the command processor. The command processor processes the commands and puts any replies as reply messages on to the reply-to queue that you specify. The first reply message contains the user message CSQN205I. See “Interpreting the replies” on page 220 for more information.

Identifying the queue manager that processes your commands

The queue manager that processes the commands you issue from an administration program is the queue manager that owns the system-command input queue that the message is put onto.

Starting the command server

Normally, the command server is started automatically when the queue manager is started. It becomes available as soon as the message CSQ9022I 'START QMGR' NORMAL COMPLETION is returned from the +cpf START QMGR command. The command server is stopped when all the connected tasks have been disconnected during the system termination phase.

You can control the command server yourself using the +cpf START CMDSERV and +cpf STOP CMDSERV commands. To prevent the command server starting automatically when MQSeries is restarted, you can add a +cpf STOP CMDSERV command to your CSQINP1 or CSQINP2 initialization data sets.

The +cpf STOP CMDSERV command stops the command server as soon as it has finished processing the current message or immediately, if no messages are being processed.

If the command server has been stopped by a +cpf STOP CMDSERV command in the program, no other commands from the program can be processed. To restart the command server, you must issue a +cpf START CMDSERV command from the OS/390 console.

If you stop and restart the command server while MQSeries is running, all the messages that are on the system-command input queue when the command server stops are processed when the command server is restarted. However, if you stop and restart MQSeries after the command server is stopped, only the persistent messages on the system-command input queue are processed when the command server is restarted. All nonpersistent messages on the system-command input queue are lost.

Preparing queues for administration programs

Before you can issue any **MQPUT** or **MQGET** calls, you must first define, and then open, the queues you are going to use.

Defining the system-command input queue

The system-command input queue is a local queue called **SYSTEM.COMMAND.INPUT**. The supplied CSQINP2 initialization data set, `thlqual.SCSQPROC(CSQ4INSG)`, contains a default definition for the system-command input queue. See “System command objects” on page 56 for more information.

Defining a reply-to queue

You must define a reply-to queue to receive reply messages from the MQSeries command processor. It can be any queue whose attributes allow reply messages to be put on it. However, for normal operation, specify these attributes:

- **MAXSMSGL(13000)**
- **USAGE(NORMAL)**
- **NOTRIGGER** (unless your application uses triggering)

You should not normally use persistent messages for commands, but if you choose to do so, the reply-to queue must not be a temporary dynamic queue.

The supplied CSQINP2 initialization data set, `thlqual.SCSQPROC(CSQ4INSG)`, contains a definition for a model queue called **SYSTEM.COMMAND.REPLY.MODEL**. You can use this model to create a dynamic reply-to queue.

Note: Replies generated by the command processor can be up to 13 000 bytes in length.

Opening the system-command input queue

Before you can open the system-command input queue, your application program must be connected to your MQSeries subsystem. Use the MQI call **MQCONN** to do this.

Then use the MQI call **MQOPEN** to open the system-command input queue. To use this call:

1. Set the *Options* parameter to **MQOO_OUTPUT**
2. Set the MQOD object descriptor fields as follows:

<i>ObjectType</i>	MQOT_Q (the object is a queue)
<i>ObjectName</i>	SYSTEM.COMMAND.INPUT
<i>ObjectQMgrName</i>	Leave blank if you want to send your request messages to your local queue manager. This means that your commands are processed locally.

If you want your MQSeries commands to be processed on a remote queue manager, put its name here. You must also have set up the correct queues and links, as described in the *MQSeries Intercommunication* manual.

Opening a reply-to queue

To be able to retrieve the replies from an MQSeries command, you must open a reply-to queue. One way of doing this is to specify the model queue, SYSTEM.COMMAND.REPLY.MODEL, in an **MQOPEN** call to create a permanent dynamic queue as your reply-to queue. To use this call:

1. Set the *Options* parameter to MQOO_INPUT_SHARED
2. Set the MQOD object descriptor fields as follows:

<i>ObjectType</i>	MQOT_Q (the object is a queue)
<i>ObjectName</i>	The name of your reply-to queue. If the queue name you specify is the name of a model queue object, the queue manager creates a dynamic queue.
<i>ObjectQMgrName</i>	To receive replies on your local queue manager, leave this field blank.
<i>DynamicQName</i>	Specify the name of the dynamic queue to be created.

Sending commands to the command server

For each command, you build a message containing the command and then you put it onto the system-command input queue.

Building a message that includes MQSeries commands

You can incorporate MQSeries commands in an application program by building request messages that include the required commands. For each such command you:

1. Create a buffer containing a character string representing the command.
2. Issue an **MQPUT** call specifying the buffer name in the *buffer* parameter of the call.

The simplest way to do this in C is to define a buffer using 'char'. For example:

```
char message_buffer[ ] = "ALTER QLOCAL(SALES) PUT(ENABLED)";
```

When you build a command, use a null-terminated character string. Do not specify a command prefix string (CPF) at the start of a command defined in this way. This means that you do not have to alter your command scripts if you want to run them on another queue manager. However, you must take into account that a CPF is included in any response messages that are put onto the reply-to queue.

The command server translates all characters to uppercase unless they are inside single quotes.

Commands can be any length up to a maximum 32762 characters.

Putting messages on the system-command input queue

Use the **MQPUT** call to put request messages containing commands on the system-command input queue. In this call you specify the name of the reply-to queue that you have already opened.

To use the **MQPUT** call:

1. Set these **MQPUT** parameters:

<i>Hconn</i>	The connection handle returned by the MQCONN call.
<i>Hobj</i>	The object handle returned by the MQOPEN call for the system-command input queue.
<i>BufferLength</i>	The length of the formatted command.
<i>Buffer</i>	The name of the buffer containing the command.

2. Set these MQMD fields:

<i>MsgType</i>	MQMT_REQUEST
<i>ReplyToQ</i>	Name of your reply-to queue.
<i>ReplyToQMgr</i>	Leave blank if you want replies sent to your local queue manager. If you want your MQSeries commands to be sent to a remote queue manager, put its name here. You must also have set up the correct queues and links, as described in the <i>MQSeries Intercommunication</i> manual.

3. Set any other MQMD fields, as required. If you are not using the same code page as the queue manager, set *CodedCharSetId* as appropriate, and set *Format* to *MQFMT_STRING*, so that the command server can convert the message. You should normally use nonpersistent messages for commands.
4. Set any *PutMsgOpts* options, as required.

If you specify *MQPMO_SYNCPOINT* (the default), you must follow the **MQPUT** call with a syncpoint call.

Using MQPUT1 and the system-command input queue

If you want to put just one message on the system-command input queue, you can use the **MQPUT1** call. This call combines the functions of an **MQOPEN**, followed by an **MQPUT** of one message, followed by an **MQCLOSE**, all in one call. If you use this call, modify the parameters accordingly. See the *MQSeries Application Programming Guide* for details.

Retrieving replies to your commands

When the command processor processes your commands, any reply messages are put onto the reply-to queue specified in the **MQPUT** call. The command server sends the reply messages with the same persistence as the command message it received.

Waiting for a reply

Use the **MQGET** call to retrieve a reply from your request message. One request message can produce several reply messages. For details, see “Interpreting the replies” on page 220.

You can specify a time interval that an **MQGET** call waits for a reply message to be generated. If you do not get a reply, use the checklist beginning on page 226.

To use the **MQGET** call:

1. Set these parameters:

<i>Hconn</i>	The connection handle returned by the MQCONN call.
<i>Hobj</i>	The object handle returned by the MQOPEN call for the reply-to queue.
<i>Buffer</i>	The name of the area to receive the reply.
<i>BufferLength</i>	The length of the buffer to receive the reply. This must be a minimum of 80 bytes.

2. To ensure that you only get the responses from the command that you issued, you must specify the appropriate *MsgId* and *CorrelId* fields. These depend on the report options, **MQMD_REPORT**, you specified in the **MQPUT** call:

Report option...	Use this <i>MsgId</i>...
MQRO_NONE	Binary zero, '00...00' (24 nulls).
MQRO_NEW_MSG_ID	Binary zero, '00...00' (24 nulls). This is the default if none of these options has been specified.
MQRO_PASS_MSG_ID	The <i>MsgId</i> from the MQPUT .
Report option...	Use this <i>CorrelId</i>...
MQRO_NONE	The <i>MsgId</i> from the MQPUT call.
MQRO_COPY_MSG_ID_TO_CORREL_ID	The <i>MsgId</i> from the MQPUT call. This is the default if none of these options has been specified.
MQRO_PASS_CORREL_ID	The <i>CorrelId</i> from the MQPUT call.

For more details on report options, see the *MQSeries Application Programming Reference* manual.

3. Set the following *GetMsgOpts* fields:

<i>Options</i>	MQGMO_WAIT If you are not using the same code page as the queue manager, set MQGMO_CONVERT, and set <i>CodedCharSetId</i> as appropriate in the MQMD.
<i>WaitInterval</i>	For replies from the local queue manager, try 5 seconds. Coded in milliseconds, this becomes 5 000. For replies from a remote queue manager, and channel control and status commands, try 30 seconds. Coded in milliseconds, this becomes 30 000.

Discarded messages

If the command server finds that a request message is not valid, it discards this message and writes the message CSQN205I to the named reply-to queue. If there is no reply-to queue, the CSQN205I message is put onto the dead-letter queue. The return code in this message shows why the original request message was not valid:

Return code	Message not valid because...
00D5020F	It is not of type MQMT_REQUEST.
00D50210	It has zero length.
00D50212	It is longer than 32 762 bytes.
00D50211	It contains all blanks.
00D5483E	It needed converting, but <i>Format</i> was not MQFMT_STRING.
OTHER	See the <i>MQSeries for OS/390 Messages and Codes</i> manual.

The reply message descriptor

For any reply message, the following MQMD message descriptor fields are set:

<i>MsgType</i>	MQMT_REPLY
<i>Feedback</i>	MQFB_NONE
<i>Encoding</i>	MQENC_NATIVE
<i>Priority</i>	As for the MQMD in the message you issued.
<i>Persistence</i>	As for the MQMD in the message you issued.
<i>CorrelId</i>	Depends on the MQPUT report options.
<i>ReplyToQ</i>	None.

The command server sets the *Options* field of the MQPMO structure to MQPMO_NO_SYNCPOINT. This means that you can retrieve the replies as they are created, rather than as a group at the next syncpoint.

_____ End of General-use programming interface _____

Interpreting the replies

Product-sensitive programming interface

Each request message correctly processed by MQSeries produces at least two reply messages. Each reply message contains a single MQSeries user message.

The length of a reply depends on the command that was issued. The longest reply you can get is from a DISPLAY NAMELIST, and that can be up to 13000 bytes long.

The first user message, CSQN205I, always contains:

- A count of the replies (in decimal), which you can use as a counter in a loop to get the rest of the replies. The count includes this first message.
- The return code from the command preprocessor.
- A reason code, which is the return code from the command processor.

This message does not contain a CPF.

For example:

```
CSQN205I  COUNT=          4, RETURN=0000000C, REASON=00000008
```

The COUNT field is 8 bytes long and is right-justified. It always starts at position 18, that is, immediately after 'COUNT='. The RETURN field is 8 bytes long in character hex and is immediately after 'RETURN=' at position 35. The REASON field is 8 bytes long in character hex and is immediately after 'REASON=' at position 52.

If the RETURN= value is 00000000 and the REASON= value is 00000004, the set of reply messages is incomplete. After retrieving the replies indicated by the CSQN205I message, issue a further **MQGET** call to wait for a further set of replies. The first message in the next set of replies will again be CSQN205I, indicating how many replies there are, and whether there are more to come.

See the *MQSeries for OS/390 Messages and Codes* manual for more details about the individual messages.

If you are using a non-English language feature, the text and layout of the replies are different from those shown here. However, the size and position of the count and return codes in message CSQN205I are the same.

Examples of commands and their replies

Here are some examples of commands that could be built into MQSeries messages, and the user messages that are the replies. Unless otherwise stated, each line of the reply is a separate message.

Messages from the DEFINE command

Command:

```
DEFINE QLOCAL(Q1)
```

Messages:

```
CSQN205I  COUNT=      2, RETURN=00000000, REASON=00000000
CSQ9022I +cpf CSQMMSGP ' DEFINE QLOCAL' NORMAL COMPLETION
```

These reply messages are produced on normal completion.

Messages from the DELETE command

Command:

```
DELETE QLOCAL(Q2)
```

Messages:

```
CSQN205I  COUNT=      4, RETURN=00000000C, REASON=000000008
CSQM094I +cpf CSQMUQLC QLOCAL (Q2) WAS NOT FOUND
CSQM090I +cpf CSQMUQLC FAILURE REASON CODE X'00D44002'
CSQ9023I +cpf CSQMUQLC ' DELETE QLOCAL' ABNORMAL COMPLETION
```

These messages indicate that a local queue called Q2 does not exist.

Using the DISPLAY commands

To obtain information about MQSeries, use the MQSeries DISPLAY commands.

Use these commands rather than **MQINQ** if you want:

- Information about objects on a remote queue manager. **MQINQ** only returns information from the local queue manager.
- Reply messages ready-formatted for printing. **MQINQ** returns information that is not formatted.
- Other information that **MQINQ** does not provide.

The format of the replies from these commands:

```
DISPLAY THREAD
DISPLAY TRACE
DISPLAY CMDSERV
```

is the same regardless of whether you issue the command from an application program or from an OS/390 console. However, if you issue DISPLAY commands for MQSeries objects, the format is different when they are issued from an application program.

Messages from the DISPLAY THREAD command

Command:

```
DISPLAY THREAD(*) TYPE(*)
```

Messages:

```
CSQN205I  COUNT=      20, RETURN=00000000, REASON=00000000.
CSQV401I +cpf DISPLAY THREAD REPORT FOLLOWS -
CSQV402I +cpf ACTIVE THREADS - 668
NAME      ST A   REQ  THREAD-XREF          USERID  ASID  URID
ABCDEFG  T      3                                ABCDEFG 002D  000000000000
MQSCIC1  T      4                                MQSCIC1 0034  000000000000
MQSCIC1  T      0 00011128C3D2C1D40000023C MQSCIC1 0034  000000000000
MQSCIC1  T      3                                MQSCIC1 0034  000000000000
MQSCIC1  T      1                                0034    000000000000
MQSCIC1  T      1                                0034    000000000000
MQSCIC1  T      1                                0034    000000000000
MQSCIC1  T      1                                0034    000000000000
MQSCIC1  T      1                                0034    000000000000
MQSCIC1  T      1                                0034    000000000000
MQSCIC1  T      1                                0034    000000000000
MQSCIC1  T      1                                0034    000000000000
MQSCIC1  T      0 00012020C3D2E3C90000039C MQSCIC1 0034  000000000000
DISPLAY ACTIVE REPORT COMPLETE
CSQV412I +cpf CSQVDT NO INDOUBT THREADS FOUND FOR NAME=MQSCIC1
CSQV412I +cpf CSQVDT NO INDOUBT THREADS FOUND FOR NAME=ABCDEFG
CSQ9022I +cpf CSQVDT ' DISPLAY THREAD' NORMAL COMPLETION
```

The actual number and content of the messages depend on what is running in your queue manager.

Messages from DISPLAY commands for MQSeries objects

These DISPLAY command responses, acting on MQSeries objects, produce user messages with a different format when you issue them from an application program:

```
DISPLAY CHANNEL/CHSTATUS
DISPLAY CLUSQMGR
DISPLAY NAMELIST
DISPLAY PROCESS
DISPLAY QMGR
DISPLAY QUEUE
DISPLAY STGCLASS
```

The user messages in the replies are still in the form of character strings, however, the attribute values in a message have the fixed positions relative to the attribute name.

The format of the reply is:

```
msg_no +cpf attr_name(value) attr_name attr_name(value)
```

where:

msg_no	An 8 character message number
+cpf	The command prefix string
attr_name	The attribute or keyword name
value	The attribute value

Messages from the DEFINE QLOCAL command

The following examples show how the results from a command depend on the attributes specified in that command.

Example 1

You define a local queue using the command:

```
DEFINE QLOCAL(Q1) DESCR('A sample queue') GET(ENABLED) SHARE
```

You then issue the following command from an application program:

```
DISPLAY QUEUE(Q1) SHARE GET DESCR
```

These three user messages are returned:

```
CSQN205I  COUNT=      3, RETURN=00000000, REASON=00000000
CSQM401I +cpf QUEUE(Q1                                ) TYPE(
QLOCAL  ) DESCR(A sample queue
          ) SHARE  GET(ENABLED  )
CSQ9022I +cpf CSQDMSG ' DISPLAY QUEUE' NORMAL COMPLETION
```

Note: The second message, CSQM401I, is shown here occupying three lines.

Example 2

Two queues have names beginning with the letter “A”:

A1 is a local queue with its PUT attribute set to DISABLED.

A2 is a remote queue with its PUT attribute set to ENABLED.

You then issue the following command from an application program:

```
DISPLAY QUEUE(A*) PUT
```

These four user messages are returned:

```
CSQN205I  COUNT=      4, RETURN=00000000, REASON=00000000
CSQM401I +cpf QUEUE(A1                                ) TYPE(
QLOCAL  ) PUT(DISABLED  )
CSQM406I +cpf QUEUE(A2                                ) TYPE(
QREMOTE ) PUT(ENABLED  )
CSQ9022I +cpf CSQDMSG ' DISPLAY QUEUE' NORMAL COMPLETION
```

Note: The second and third messages, CSQM401I and CSQM406I respectively, are shown here occupying two lines each.

Messages from the DEFINE NAMELIST command

A namelist is defined by the command:

```
DEFINE NAMELIST(N1) NAMES(Q1,SAMPLE_QUEUE)
```

You then issue the following command from an application program:

```
DISPLAY NAMELIST(N1) NAMES NAMCOUNT
```

The following three user messages are returned:

```
CSQN205I  COUNT=      3, RETURN=00000000, REASON=00000000
CSQM407I +cpf NAMELIST(N1                                ) NA
MOUNT(      2) NAMES(Q1
,SAMPLE_QUEUE
)
CSQ9022I +cpf CSQMDMSG ' DISPLAY NAMELIST' NORMAL COMPLETION
```

Note: The third message, CSQM407I, is shown here occupying three lines.

Finding out the name of the dead-letter queue

You want to find out the name of the dead-letter queue for a queue manager.

You issue this command from an application program:

```
DISPLAY QMGR DEADQ
```

The following three user messages are returned from which you can extract the required name:

```
CSQN205I  COUNT=      3, RETURN=00000000, REASON=00000000
CSQM409I +cpf QMNAME(CSQ1) DEADQ(SYSTEM.DEAD.QUEUE      )
CSQ9022I +cpf CSQMDRTS ' DISPLAY QMGR' NORMAL COMPLETION
```

Command attributes

When using commands in administration programs, you should consider the following:

1. Not all attributes have associated values.
2. Each attribute or attribute and value pair is separated by one or more blanks.
3. Do not make any assumptions about the order in which attributes are returned.
4. The attribute values returned are fixed length and surrounded by parentheses.
Integer values are ten characters long, right justified, and padded with blanks.
Character values are left justified and padded with blanks. Their lengths are as follows:
 - a. Character string lengths are the same as those given in the *MQSeries Application Programming Reference* manual.
 - b. Attributes that return a keyword (for example, DEFSOPT returns EXCL or SHARED) are 10 characters long, left justified, and padded with blanks.
 - c. Some attribute keywords can take negated values, for example, NOTRIGGER, NOSHARE, and NOHARDENBO. The attribute keywords that can have negated values take their length from the negated value. For example, the negated equivalent of SHARE is NOSHARE; it has a length of 7. These attributes are left justified and padded with blanks.
5. The number of attributes returned depends on what attributes are requested by the command.
6. The NAMES attribute of a namelist returns multiple values. This attribute returns a list of names, each of fixed length, separated by commas. Use the NAMCOUNT attribute to discover the number of names in the list. If there are no names in the list, the NAMES attribute is returned as NAMES().
7. Attributes that normally require quotes around the string because they contain embedded blanks, lowercase characters or special characters, are returned without the quotes.
8. When you want to use the reply to a +cpf DISPLAY command as input to another command, put single quotes (' ') around each attribute. For example, if you define this queue:

```
+cpf DEFINE QLOCAL(SALES) DESCR('Sales enquiries queue')
```

You can display it using the command:

```
+cpf DISPLAY QUEUE(SALES) DESCR
```

The DESCR attribute is displayed as:

```
DESCR(Sales enquiries queue)
```

To use this description in another command you must add the quotes as follows:

```
DESCR('Sales enquiries queue')
```

If the attribute itself contains any quotes, you must double them.

If you do not receive a reply

If you do not receive a reply to your request message, work through this checklist:

- Is the command server running?
- Is the *WaitInterval* long enough?
- Are the system-command input and reply-to queues correctly defined?
- Were the **MQOPEN** calls to these queues successful?
- Are both the system-command input and reply-to queues enabled for **MQPUT** and **MQGET** calls?
- Have you considered increasing the MAXDEPTH and MAXMSGL attributes of your queues?
- Are you are using the *CorrelId* and *MsgId* fields correctly?
- Is the MQSeries subsystem still running?
- Was the command built correctly?
- Are all your remote links defined and operating correctly?
- Were the **MQPUT** calls correctly defined?
- Has the reply-to queue been defined as a temporary dynamic queue instead of a permanent dynamic queue? (If the request message is persistent, you must use a permanent dynamic queue for the reply.)

When the command server generates replies but cannot write them to the reply-to queue that you specify, it tries to write them to the system dead-letter queue.

Passing commands using MGCR or MGCRE

If you have the correct authorization, you can pass MQSeries commands from your program to multiple MQSeries subsystems by the MGCR or MGCRE (SVC 34) OS/390 service. The value of the CPF identifies the particular MQSeries subsystem to which the command is directed. For details about CPFs, see "Updating the subsystem name table" on page 41.

If you use MGCRE, you can use a Command and Response Token (CART) to get the direct responses to the command.

_____ End of Product-sensitive programming interface _____

Chapter 14. Using the MQSeries utilities

This chapter describes the MQSeries utility programs that are provided to help you perform various administrative tasks. Table 16 summarizes what you can do with these utilities.

<i>Table 16. A summary of MQSeries utilities</i>		
Name	Purpose	See page
CSQUTIL (MQSeries utility program) Page set management	FORMAT Format VSAM data sets as MQSeries page sets. COPYPAGE Copy MQSeries page sets. RESETPAGE Copy MQSeries page sets and reset the log information.	231
CSQUTIL Command management	COMMAND Issue MQSeries commands from a sequential data set. Produce a set of DEFINE commands for objects. Produce a client channel definition file. SDEFS Produce a set of DEFINE commands for objects (offline).	238
CSQUTIL Queue management	COPY Copy contents of a queue to a data set. SCOPY Copy contents of a queue to a data set (offline). EMPTY Delete contents of a queue. LOAD Restore contents of a queue.	245
CSQUCVX (Data conversion exit utility)	Generates data conversion exit routines. For information about the CSQUCVX utility, see the <i>MQSeries Application Programming Guide</i> .	–
CSQJU003 (Change log inventory utility)	Changes the bootstrap data set. NEWLOG Add active or archive log data sets. DELETE Delete active or archive log data sets. ARCHIVE Supply passwords for archive logs. CRESTART Control the next restart of MQSeries. CHECKPT Set checkpoint records. HIGHRBA Update the highest written log RBA.	256
CSQJU004 (Print log map utility)	Lists information about the log.	265
CSQ1LOGP (Log print utility)	Prints the log.	266

These utilities are located in the main MQSeries load library thlqual.SCSQAUTH.

Note: Include the appropriate MQSeries language load library thlqual.SCSQANLx (where x is the language letter) in the STEPLIB concatenation before the thlqual.SCSQAUTH or thlqual.SCSQLOAD. The utility control statements are available only in US English.

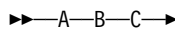
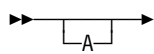
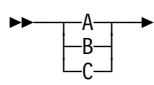
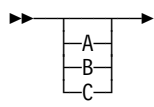
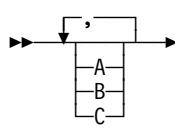
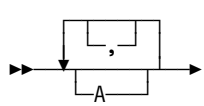
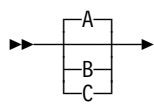
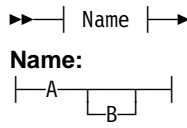
How to read syntax diagrams

This chapter contains syntax diagrams (sometimes referred to as “railroad” diagrams).

Each syntax diagram begins with a double right arrow and ends with a right and left arrow pair. Lines beginning with a single right arrow are continuation lines. You read a syntax diagram from left to right and from top to bottom, following the direction of the arrows.

Other conventions used in syntax diagrams are:

Table 17. How to read syntax diagrams

Convention	Meaning
	You must specify values A, B, and C. Required values are shown on the main line of a syntax diagram.
	You may specify value A. Optional values are shown below the main line of a syntax diagram.
	Values A, B, and C are alternatives, one of which you must specify.
	Values A, B, and C are alternatives, one of which you may specify.
	You may specify one or more of the values A, B, and C. Any required separator for multiple or repeated values (in this example, the comma (,)) is shown on the arrow.
	You may specify value A multiple times. The separator in this example is optional.
	Values A, B, and C are alternatives, one of which you may specify. If you specify none of the values shown, the default A (the value shown above the main line) is used.
	The syntax fragment Name is shown separately from the main syntax diagram.
Punctuation and uppercase values	Specify exactly as shown.
Lowercase values (for example, <i>name</i>)	Supply your own text in place of the <i>name</i> variable.

MQSeries utility program (CSQUTIL)

The CSQUTIL utility program is provided with MQSeries to help you to perform backup, restoration, and reorganization tasks, and to issue MQSeries commands. Through this utility program, you can invoke functions in these groups:

- Page set management, see page 231
- Issuing commands, see page 238
- Queue management, see page 245

All of the page set management functions and some of the other functions operate while the queue manager is not running; for these therefore, you do not need any special authorization other than the appropriate access to the page set data sets. For the functions that operate while the queue manager is running, CSQUTIL runs as an ordinary OS/390 batch MQSeries program, issuing commands through the command server and using the MQSeries API to access queues. You need the necessary authority to use the command server queues (SYSTEM.COMMAND.INPUT, SYSTEM.COMMAND.REPLY.MODEL, and SYSTEM.CSQUTIL.*), to use the MQSC DISPLAY commands, and to use the MQSeries API to access any queues that you wish to manage. See the usage notes for each function for more information.

Invoking the MQSeries utility program

The CSQUTIL utility program runs as an OS/390 batch program, below the 16 MB storage line. In the PARM parameter of the EXEC statement of the JCL, specify the queue manager that the utility is to work with. For example, see Figure 71.

```
// EXEC PGM=CSQUTIL,PARM='CSQ1'
```

Figure 71. Specifying the queue manager that a utility is to work with

If you specify a queue manager name as blanks, CSQUTIL uses the name of the default queue manager specified for OS/390 batch programs in CSQBDEFV. The utility then uses this queue manager for the whole job step. When the utility connects to the queue manager, the authorization of the “signed-on user name” is checked to see which functions the invocation is allowed to use.

You specify the functions required by statements in the SYSIN data set according to these rules:

- The data set must have a record length of 80.
- Only columns 1 through 72 are significant. Columns 73 through 80 are ignored.
- Records with an asterisk (*) in column 1 are interpreted as comments and are ignored.
- Blank records are ignored.
- Each statement must start on a new line.
- A trailing – means continue from column 1 of the next record.
- A trailing + means continue from the first non-blank column of the next record.
- The keywords of statements are not case-sensitive. However, some arguments, such as queue name, are case-sensitive.

The utility statements refer to the default or explicitly named DDnames for input and output. Your job can use the COPY and LOAD functions repeatedly and process different page sets or queues during a single run of the utility.

All output messages are sent to the SYSPRINT data set, which must have a record format VBA and a record length 125.

While running, CSQUTIL uses temporary dynamic queues with names of the form SYSTEM.CSQUTIL.*

Return codes

When CSQUTIL returns to the operating system, the return code can be:

- 0** All functions completed successfully.
- 4** Some functions completed successfully, some did not or forced a syncpoint.
- 8** All the attempted functions failed.
- 12** No functions attempted; there was a syntax error in the statements or the expected data sets were missing.

In most cases, if a function fails or is forced to take a syncpoint, no further functions are attempted. In this case, the message CSQU147I replaces the normal completion message CSQU148I.

See the usage notes for each function for more information about success or failure.

Monitoring the progress of the MQSeries utility program

To record the progress of CSQUTIL, every SYSIN statement is echoed to SYSPRINT.

The utility first checks the syntax of the statements in the SYSIN. The requested functions are started only if all the statements are syntactically correct.

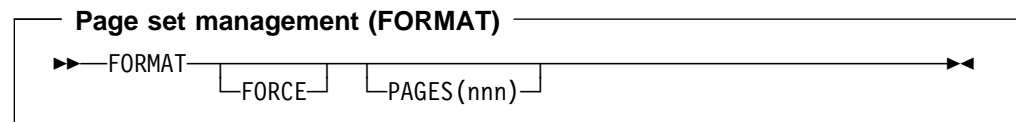
Messages giving a commentary on the progress of each function are sent to SYSPRINT. When the processing of the utility is complete, statistics are printed with an indication of how the functions completed.

Page set management functions

These functions enable you to manage MQSeries page sets. You can format data sets as page sets, you can increase the size of page sets and, if required, reset the log information contained in a page set. The page set must not belong to a queue manager that is currently running.

Formatting page sets (FORMAT)

Use the FORMAT function to format page sets on all data sets specified by DDnames CSQP0000 through CSQP0099. In this way, you can format up to 100 page sets by invoking the utility program once. Use the FORCE parameter to reuse existing page sets.



Keywords and parameters

FORCE

Specifies that existing page sets are to be re-used without having to delete and redefine them first. You must define any page sets you want to re-use with the REUSE attribute in the AMS DEFINE CLUSTER statement. For more information about DEFINE CLUSTER, see the *DFSMS/MVS Access Method Services for VSAM* or the *DFSMS/MVS Access Method Services for the Integrated Catalog Facility* manual.

PAGES(nnn)

Specifies the minimum number of pages to format in each page set. This enables a data set that spans more than one volume to be formatted.

Formatting of the data set is always done in whole space allocations, as specified as primary or secondary quantities when the data set is defined. The number of space allocations formatted is the minimum necessary to provide the requested number of pages; if there is insufficient data set space available, as many extents as can be obtained are formatted. If an existing page set is being reused (with the FORCE keyword), the whole page set is formatted if that is larger.

The number of pages must be in the range 1 through 1 048 576 (because the maximum page set size is 4 GB (gigabytes)). The default is 1.

The number of pages formatted is reported by message CSQU092I for each page set.

Example

Figure 72 on page 232 illustrates how the FORMAT command is invoked from CSQUTIL. In this example, two page sets, referenced by CSQP0000 and CSQP0003 respectively, are formatted by CSQUTIL.

```
//FORMAT EXEC PGM=CSQUTIL
//STEPLIB DD DISP=SHR,DSN=th1qua1.SCSQANLE
// DD DISP=SHR,DSN=th1qua1.SCSQAUTH
//CSQP0000 DD DISP=OLD,DSN=pageset.dsname0
//CSQP0003 DD DISP=OLD,DSN=pageset.dsname3
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
FORMAT
/*
```

Figure 72. Sample JCL for the FORMAT function of CSQUTIL

Usage notes

1. You cannot format page sets that belong to a queue manager that is still running.
2. When you use FORMAT, it is not necessary to specify a queue manager name.
3. If you use data set names in which the queue manager name is a high-level qualifier, you can more easily identify which page sets are used by which MQSeries subsystem, if more than one MQSeries subsystem is defined.
4. If there is an error when formatting a page set, it does not prevent other page sets from being formatted, although the FORMAT function is considered to have failed.
5. If FORMAT fails, no further CSQUTIL functions are attempted.

Expanding a page set (COPYPAGE)

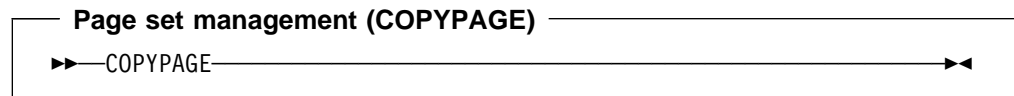
Note: The COPYPAGE function is only used for *expanding* page sets. It is not used for making backup copies of page sets. If you want to do this, use AMS REPRO as described in “Backing up and recovering page sets” on page 331. When you have used the COPYPAGE function, the page sets cannot be used by a queue manager with a different name, so you must not rename your queue manager.

Use the COPYPAGE function to copy one or more page sets. All queues and messages on the page set are copied. If you copy page set zero, all the MQSeries object definitions are also copied. Each page set is copied to a destination data set that must be formatted as a page set. Copying to a smaller page set is not supported.

If you use this function, you must modify the page set definition in the startup procedure to reflect the change of the name of the data set on which the new page set resides.

To use the COPYPAGE function, define DDnames in the range CSQS0000 through CSQS0099 for the source data sets, and define DDnames for the target data sets from CSQT0000 through CSQT0099 respectively.

For more information, see Chapter 20, “Managing page sets” on page 325.



Keywords and parameters

There are no keywords or parameters.

Example

In Figure 73 on page 234, two existing page sets are copied onto two new page sets. The procedure for this is:

1. Set up the required DDnames, where:
 - CSQP0005, CSQP0006** Identify the destination data sets. These DDnames are used by the FORMAT function.
 - CSQS0005, CSQS0006** Identify the source data sets containing the two page sets you want to copy.
 - CSQT0005, CSQT0005** Identify the destination data sets (page sets), but this time for the COPYPAGE function.
2. Format the destination data sets, referenced by DDnames CSQP0005 and CSQP0006, as page sets using the FORMAT function.
3. Copy the two existing page sets onto the new page sets using the COPYPAGE function.

```
//COPYPAGE EXEC PGM=CSQUTIL
//STEPLIB DD DISP=SHR,DSN=thlqual.SCSQANLE
// DD DISP=SHR,DSN=thlqual.SCSQAUTH
//CSQP0005 DD DISP=OLD,DSN=pageset.newname5
//CSQP0006 DD DISP=OLD,DSN=pageset.newname6
//CSQS0005 DD DISP=OLD,DSN=pageset.oldname5
//CSQS0006 DD DISP=OLD,DSN=pageset.oldname6
//CSQT0005 DD DISP=OLD,DSN=pageset.newname5
//CSQT0006 DD DISP=OLD,DSN=pageset.newname6
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
* Format new data sets (CSQP0005 and CSQP0006) as page sets
  FORMAT
* Copy old page sets CSQS0005 and CSQS0006 to new
  * page sets CSQT0005 and CSQT0006
  COPYPAGE
/*
```

Figure 73. Sample JCL showing the use of the COPYPAGE function

Usage notes

1. You cannot use COPYPAGE on page sets of a queue manager that is running.
2. Using COPYPAGE involves stopping the queue manager. This will result in the loss of nonpersistent messages.
3. Before you use COPYPAGE, the new data sets must be pre-formatted as page sets. To do this, use the FORMAT function, as shown in Figure 73.
4. Ensure that the new (destination) data sets are larger than the old (source) data sets.
5. You cannot change the page set identifier (PSID) associated with a page set. For example, you cannot 'make' page set 03 become page set 05.
6. Failure of this function does not prevent other page set management functions from being completed.
7. If you attempt to use the COPYPAGE function after MQSeries has terminated abnormally, it is possible that the page sets have not been closed properly. If a page set has not been closed properly, you cannot successfully run the COPYPAGE function against it.

To avoid this problem, run the AMS VERIFY command before using the COPYPAGE function. The AMS VERIFY command might produce error messages. However, it does close the page sets properly, so that the COPYPAGE function can complete successfully.

For more information about the AMS VERIFY command, see the *DFSMS/MVS Access Method Services for VSAM* or the *DFSMS/MVS Access Method Services for the Integrated Catalog Facility* manual.

Copying a page set and resetting the log (RESETPAGE)

The RESETPAGE function is similar to the COPYPAGE function except that it also resets the log information in the new page sets. RESETPAGE lets you restart MQSeries from a known, valid set of page sets, even if the corresponding log data sets have been corrupted.

RESETPAGE either:

- Copies page sets on all data sets referenced by DDnames CSQS0000 through CSQS0099 to new data sets referenced by DDnames CSQT0000 through CSQT0099 respectively. If you use this function, you must modify the page set definition in the startup procedure to reflect the change of the name of the data set on which the new page set resides.
- Resets the log information in the page set referenced by DDnames CSQP0000 through CSQP0099.

For more information, see Chapter 20, “Managing page sets” on page 325.

Using the RESETPAGE function

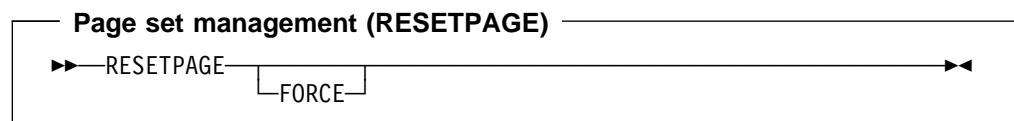
You can use the RESETPAGE function to generate a set of consistent page sets that can be used with a set of new (clean) BSDS and log data sets to start MQSeries. You would only have to do this if both copies of the log have been lost or damaged for some reason; you can restart from backup page set copies (and accept the resulting loss of data from the time the copies were made), or from your existing page sets.

In this situation, you should use the RESETPAGE function on **all** the page sets of the affected queue manager. You must also create new BSDS and log data sets.

Note: The RESETPAGE function should not be used on a subset of the page sets known to MQSeries.

If you run the RESETPAGE function against any page sets, but do not provide clean BSDS and log data sets for the MQSeries subsystem, MQSeries will attempt to recover the logs from RBA zero, and will treat the page sets as empty. For example, the following messages would be produced if you attempted to use the RESETPAGE function to generate page sets 0, 1, 2, and 3 without providing a clean set of BSDS and log data sets:

```
CSQI021I +cpf CSQIECUR PAGE SET 0 IS EMPTY. MEDIA RECOVERY STARTED
CSQI021I +cpf CSQIECUR PAGE SET 1 IS EMPTY. MEDIA RECOVERY STARTED
CSQI021I +cpf CSQIECUR PAGE SET 2 IS EMPTY. MEDIA RECOVERY STARTED
CSQI021I +cpf CSQIECUR PAGE SET 3 IS EMPTY. MEDIA RECOVERY STARTED
```



Keywords and parameters

FORCE

Specifies that the page sets specified by DDnames CSQP0000 through CSQP00nn are to be reset in place.

If FORCE is not specified, the page sets specified by DDnames CSQS0000 through CSQS00nn are copied to new page sets specified by DDnames CSQT0000 through CSQT00nn. This is the default.

Example

An existing page set, referenced by DDname CSQS0007, is copied to a new data set referenced by DDname CSQT0007. The new data set, which is also referenced by DDname CSQP0007, is already formatted as a page set before the RESETPAGE function is called.

```
//RESTPAGE EXEC PGM=CSQUTIL
//STEPLIB DD DISP=SHR,DSN=thlqual.SCSQANLE
// DD DISP=SHR,DSN=thlqual.SCSQAUTH
//CSQP0007 DD DISP=OLD,DSN=pageset.newname7
//CSQS0007 DD DISP=OLD,DSN=pageset.oldname7
//CSQT0007 DD DISP=OLD,DSN=pageset.newname7
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
* Format new data set, CSQP0007, as page set
FORMAT
* Copy page set CSQS0007 to CSQT0007 and reset it
RESETPAGE
/*
```

Figure 74. Sample JCL showing the use of the RESETPAGE function

Usage notes

1. You cannot use RESETPAGE on page sets belonging to a queue manager that is running.
2. Using RESETPAGE involves stopping the queue manager. This will result in the loss of nonpersistent messages.
3. Before you use RESETPAGE, the new data sets must be pre-formatted as page sets. To do this, use the FORMAT function, as shown in Figure 74.
4. Ensure that the new (destination) data sets are larger than the old (source) data sets.
5. You cannot change the page set identifier (PSID) associated with a page set. For example, you cannot 'make' page set 03 become page set 05.
6. Failure of this function does not prevent other page set management functions from being completed.
7. If you attempt to use the RESETPAGE function after MQSeries has terminated abnormally, it is possible that the page sets have not been closed properly. If a page set has not been closed properly, you cannot successfully run the RESETPAGE function against it.

To avoid this problem, run the AMS VERIFY command before using the RESETPAGE function. The AMS VERIFY command might produce error

messages. However, it does close the page sets properly, so that the RESETPAGE function can complete successfully.

For more information about the AMS VERIFY command, see the *DFSMS/MVS Access Method Services for VSAM* or the *DFSMS/MVS Access Method Services for the Integrated Catalog Facility* manual.

MQSeries command management functions

These functions enable you to:

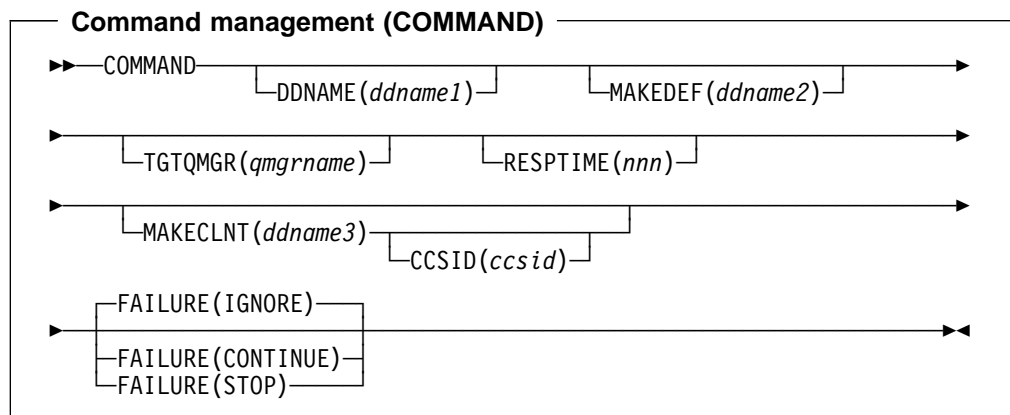
- Issue commands to MQSeries
- Produce a list of define commands describing the objects in your MQSeries subsystem

Issuing commands to MQSeries (COMMAND)

Use the COMMAND function to:

1. Pass MQSeries commands from an input data set to the queue manager.
2. Produce a list of MQSeries DEFINE commands that describe the objects in an MQSeries subsystem. The statements can be used to keep a record of the object definitions or to regenerate all or part of a queue manager's objects as part of a migration from one MQSeries system to another.
3. Make a client channel definition file.

The queue manager specified in the PARM parameter of the EXEC statement must be running.



Keywords and parameters

DDNAME(ddname1)

Specifies that the MQSeries commands are to be read from a named input data set. If this keyword is omitted, the default DDname, CSQUCMD, is used.

ddname1 specifies the DDname that identifies the input data set from which MQSeries commands are to be read.

MAKEDEF(ddname2)

Specifies that DEFINE commands are to be generated from any DISPLAY object commands in the input data set.

There is no default if this keyword is omitted.

ddname2 specifies the DDname that identifies the output data set in which the DEFINE statements are to be stored. The data set should be RECFM=FB, LRECL=80. This data set can then be used as input for a later invocation of the COMMAND function or it can be incorporated into the initialization data sets CSQINP1 and CSQINP2.

TGTQMGR(*qmgrname*)

Specifies the name of the (remote) queue manager where you want the commands to be performed.

The default is that commands are performed on the queue manager to which you are connected, as specified in the PARM field of the EXEC statement.

RESPTIME(*nnn*)

Specifies the time in seconds to wait for a response to each of the commands, in the range 5 through 999.

The default is 30 seconds.

MAKECLNT(*ddname3*)

Specifies that a client channel definition file, in binary format suitable for downloading to a client machine, is to be generated from any DISPLAY CHANNEL commands in the input data set that return information about client-connection channels.

If this keyword is omitted, no file is generated.

ddname3 specifies the DDname that identifies the output data set in which the generated file is to be stored; the data set should be RECFM=U, LRECL=2048, BLKSIZE=2048. The file can then be downloaded as binary data to the client machine by a suitable file transfer program.

CCSID(*ccsid*)

Specifies the coded character set identifier that is to be used for the data in a client channel definition file. The value must be in the range 1 through 65535; the default is 437. You can only specify CCSID if you also specify MAKECLNT.

Note: MQSeries assumes that the data is to be in ASCII, and that the encoding for numeric data is to be MQENC_INTEGER_REVERSED.

FAILURE

Specifies what action to take if an MQSeries command that is issued fails to execute successfully. Values are:

- IGNORE** Ignore the failure; continue reading and issuing commands, and treat the COMMAND function as being successful. This is the default.
- CONTINUE** Read and issue any remaining commands in the input data set, but treat the COMMAND function as being unsuccessful.
- STOP** Do not read or issue any more commands, and treat the COMMAND function as being unsuccessful.

Examples

Issuing commands: In Figure 75 on page 240, the data sets referenced by DDnames CSQUCMD and OTHER contain sets of MQSeries commands. The first COMMAND statement takes MQSeries commands from the default input data set MY.MQSERIES.COMMANDS(COMMAND1) and passes them to the command processor. The second COMMAND statement takes MQSeries commands from the input data set MY.MQSERIES.COMMANDS(OTHER1), which is referenced by DDname OTHER.

```

//COMMAND EXEC PGM=CSQUTIL,PARM='CSQ1'
//STEPLIB DD DISP=SHR,DSN=thlqua1.SCSQANLE
// DD DISP=SHR,DSN=thlqua1.SCSQAUTH
//CSQUCMD DD DSN=MY.MQSERIES.COMMANDS(COMMAND1),DISP=SHR
//OTHER DD DSN=MY.MQSERIES.COMMANDS(OTHER1),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
* NEXT STATEMENT CAUSES COMMANDS TO BE READ FROM CSQUCMD DDNAME
COMMAND
* THE NEXT SET OF COMMANDS WILL COME FROM 'OTHER' DDNAME
COMMAND DDNAME(OTHER)
* THE NEXT STATEMENT CAUSES COMMANDS TO BE READ FROM CSQUCMD
* DDNAME AND ISSUED ON QUEUE MANAGER CSQ2 WITH A RESPONSE TIME
* OF 10 SECONDS
COMMAND TGTQMGR(CSQ2) RESPTIME(10)
/*

```

Figure 75. Sample JCL for issuing MQSeries commands using CSQUTIL

Making a list of DEFINE commands: In Figure 76, the data set referenced by DDname CMDINP contains a set of MQSeries DISPLAY commands. These DISPLAY commands specify generic names for each object type (except the queue manager itself). If you run these commands, a list is produced containing all the MQSeries objects (except the queue manager). In these DISPLAY commands, the ALL keyword is specified to ensure that all the attributes of all the objects are included in the list.

The MAKEDEF keyword causes this list to be converted into a corresponding set of DEFINE commands. These commands are put into a data set referenced by the *ddname2* parameter of the MAKEDEF keyword, that is, OUTPUT1. If you run this set of commands, MQSeries regenerates all the object definitions (except the queue manager) in the MQSeries subsystem.

```

//QDEFS EXEC PGM=CSQUTIL,PARM='CSQ1'
//STEPLIB DD DISP=SHR,DSN=thlqua1.SCSQANLE
// DD DISP=SHR,DSN=thlqua1.SCSQAUTH
//OUTPUT1 DD DISP=OLD,DSN=MY.MQSERIES.COMMANDS(DEFS)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
COMMAND DDNAME(CMDINP) MAKEDEF(OUTPUT1)
/*
//CMDINP DD *
DISPLAY STGCLASS(*)
DISPLAY QUEUE(*) ALL
DISPLAY NAMELIST(*) ALL
DISPLAY PROCESS(*) ALL
DISPLAY CHANNEL(*) ALL
/*

```

Figure 76. Sample JCL for using the MAKEDEF option of the COMMAND function

Making a client channel definition file: In Figure 77 on page 241, the data set referenced by DDname CMDCHL contains an MQSeries DISPLAY CHANNEL command. The DISPLAY command specifies a generic name and the ALL keyword is specified to ensure that all the attributes are included.

The MAKECLNT keyword causes this to be converted into a corresponding set of client channel definitions. These are put into a data set referenced by the *ddname3* parameter of the MAKECLNT keyword, that is, OUTCLNT, which is ready to be downloaded to the client machine.

```
//CLIENT EXEC PGM=CSQUTIL,PARM='CSQ1'
//STEPLIB DD DISP=SHR,DSN=thlqua1.SCSQANLE
// DD DISP=SHR,DSN=thlqua1.SCSQAUTH
//OUTCLNT DD DISP=OLD,DSN=MY.MQSERIES.CLIENTS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
COMMAND DDNAME(CMDCHL) MAKECLNT(OUTCLNT)
/*
//CMDCHL DD *
DISPLAY CHANNEL(*) ALL TYPE(CLNTCONN)
/*
```

Figure 77. Sample JCL for using the MAKECLNT option of the COMMAND function

Usage notes

1. The format of commands issued from the COMMAND function is similar to the MQSeries operator command format. See the *MQSeries Command Reference* manual for information about the rules for building MQSeries commands.
2. The rules for specifying commands in the input data set are the same as for the initialization data sets:
 - The data set must have a record length of 80.
 - Only columns 1 through 72 are significant. Columns 73 through 80 are ignored.
 - Records with an asterisk (*) in column 1 are interpreted as comments and are ignored.
 - Blank records are ignored.
 - Each command must start on a new record.
 - A trailing – means continue from column 1 of the next record.
 - A trailing + means continue from the first non-blank column of the next record.
 - The maximum number of characters permitted in a command is 32 762.

With the additional rule:

- A semicolon (;) can be used to terminate a command, the remaining data in the record is ignored.

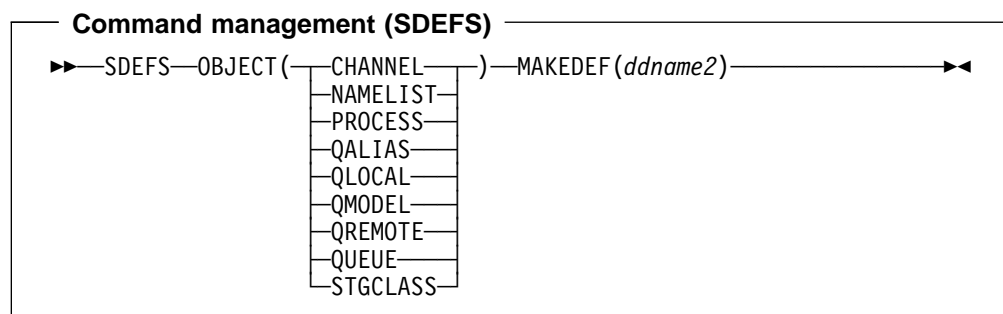
3. If you specify the MAKEDEF keyword:
 - In the input data set, the DISPLAY commands for objects must contain the ALL parameter so that the complete definition of each object is produced. See Figure 76 on page 240.
 - To obtain a complete definition, you must DISPLAY the following:
 - Queues
 - Namelist
 - Process definitions
 - Channels
 - Storage classes
 - Note:** DEFINE commands are not generated for any local queues that can be identified as dynamic, or for channels that were defined automatically.
 - Do not specify the same MAKEDEF data set for more than one COMMAND function, unless its DD statement specifies a sequential data set with DISP=MOD.
4. Whether or not the MAKEDEF or MAKECLNT keywords are used, the results of these DISPLAY commands are also sent to SYSPRINT.
5. If you specify the MAKECLNT keyword:
 - In the input data set, the display commands for channels must contain the ALL parameter so that the complete definition of each channel is produced.
 - If the DISPLAY commands return information for a given channel more than once, only the last set of information is used.
 - Do not specify the same client definition file data set for more than one COMMAND function, unless its DD statement specifies a sequential data set with DISP=MOD.
6. If you specify the FAILURE keyword, a command is considered to execute successfully or not according to the codes returned in message CSQN205I. If the return code is 00000000 and the reason code is 00000000 or 00000004, it is a success; for all other values it is a failure.
7. The COMMAND function is considered to be successful only if both:
 - All the commands in the input data set are read and issued and get a response from MQSeries, regardless of whether the response indicates successful execution of the command or not.
 - Every command issued executes successfully, if FAILURE(CONTINUE) or FAILURE(STOP) is specified.

If COMMAND fails, no further CSQUTIL functions are attempted.
8. You need the necessary authority to use the command server queues (SYSTEM.COMMAND.INPUT, SYSTEM.COMMAND.REPLY.MODEL, and SYSTEM.CSQUTIL.*) and to use the MQSC commands that you wish to issue.

Producing a list of MQSeries define commands (SDEFS)

Use the SDEFS function to produce a list of DEFINE statements describing the objects in your MQSeries subsystem, when the queue manager is not running.

To use the SDEFS function, DDname CSQP0000 must specify the data set with page set zero for the subsystem required.



Keywords and parameters

OBJECT

Specifies the type of object to be listed.

MAKEDEF(ddname2)

Specifies that define commands generated for the object are to be placed in the output data set identified by the DDname. The data set should be RECFM=FB, LRECL=80. This data set can then be used as input for a later invocation of the COMMAND function or it can be incorporated into the initialization data sets CSQINP1 and CSQINP2.

Note: DEFINE commands are not generated for any local queues that can be identified as dynamic, or for channels that were defined automatically.

Example

```

//SDEFS EXEC PGM=CSQUTIL
//STEPLIB DD DISP=SHR,DSN=thlqua1.SCSQANLE
// DD DISP=SHR,DSN=thlqua1.SCSQAUTH
//CSQP0000 DD DISP=OLD,DSN=pageset.dsname0
//OUTPUT1 DD DISP=OLD,DSN=MY.MQSERIES.COMMANDS(DEFS)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
SDEFS OBJECT(Queue) MAKEDEF(OUTPUT1)
/*

```

Figure 78. Sample JCL for the SDEFS function of CSQUTIL

Usage notes

1. You should not use SDEFS for a queue manager that is running because results will be unpredictable. You can avoid doing this accidentally by using DISP=OLD in the CSQP0000 DD statement.
2. When you use SDEFS, it is not necessary to specify a queue manager name.

Issuing commands

3. To use the SDEFS function more than once in a job, specify different DDnames and data sets for each invocation of the function, or specify a sequential data set and DISP=MOD in the DD statements.

Queue management functions

Use the CSQUTIL queue management functions to back up and restore queues and page sets or to copy queues and page sets to another MQSeries system. You can use these functions to reset your MQSeries subsystem or for migrating from one MQSeries subsystem to another.

Specifically, you can:

- Copy messages from a queue to a data set
- Delete messages from a queue
- Restore previously copied messages to their respective queues

The scope of these functions can be either:

- A *queue*, in which case the function operates on all messages in the specified queue.
- A *page set*, in which case the function operates on all the messages, in all the queues, on the specified page set.

You should use these functions only for your own queues; do not use them for system queues (those with names beginning SYSTEM).

Syncpoints

The queue management functions run within a syncpoint so that, if a function fails, its effects can be backed out. The MQSeries entity, MAXSMSGS, specifies the maximum number of messages that a task can get or put within a single unit of recovery. MAXSMSGS should be greater than:

- The number of messages in the queue – if you are working with a single queue.
- The number of messages in the longest queue in the page set – if you are working with an entire page set.

Otherwise, the utility forcibly takes syncpoints as required and issues the warning message CSQU087I. If the function subsequently fails, the already committed changes will not be backed out. Do not simply re-run the job to correct the problem or you might get duplicate messages on your queues. Instead, use the current depth of the queue to work out, from the utility output, which messages have not been backed out. Then determine the most appropriate course of action. For example, you can empty the queue and start again or you can choose to accept duplicate messages on the queues.

Use the DISPLAY QLOCAL command to find out the value of the CURDEPTH attribute, which is the current depth of the queue. To find out the value of MAXSMSGS, use the DISPLAY MAXSMSGS command. See the *MQSeries Command Reference* manual for more information.

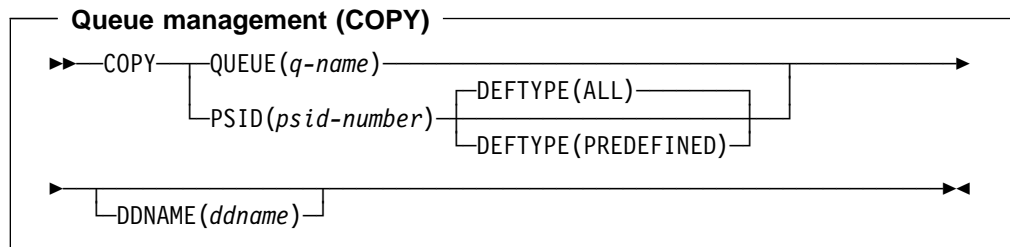
Copying queues into a data set while the queue manager is running (COPY)

Use the COPY function to copy queued messages to a sequential data set, when the queue manager is running, without destroying any messages in the original queues.

The scope of the COPY function is determined by the keyword that you specify in the first parameter. You can either copy all the messages from a named queue, or all the messages from all the queues on a named page set.

Use the complementary function, LOAD, to restore the messages to their respective queues.

Note: If you want to copy the object definitions from the named page set, use COPYPAGE. If you want to copy messages to a data set when the queue manager is not running, use SCOPY.



Keywords and parameters

QUEUE(*q-name*)

QUEUE specifies that messages in the named queue are to be copied. The keyword QUEUE can be abbreviated to Q.

q-name specifies the name of the queue to be copied. This name is case-sensitive.

PSID(*psid-number*)

PSID specifies that all the messages in all the queues in the specified page set are to be copied.

psid-number is the page set identifier, which specifies the page set to be used. This identifier is a two-digit integer (whole number) representing a single page set.

DEFTYPE

Specifies whether to copy dynamic queues:

ALL Copy all queues; this is the default.

PREDEFINED Do not include dynamic queues; this is the same set of queues that are selected by the COMMAND and SDEFS functions with the MAKEDEF parameter.

DDNAME

Specifies that the messages are to be copied to a named data set. If this keyword is omitted, the default DDname, CSQUOUT, is used. The keyword DDname can be abbreviated to DD.

ddname specifies the DDname of the destination data set, which is used to store the messages. The record format of this data set must be variable block spanned (VBS).

Example

```
//COPY EXEC PGM=CSQUTIL,PARM='CSQ1'
//STEPLIB DD DISP=SHR,DSN=thlqua1.SCSQANLE
// DD DISP=SHR,DSN=thlqua1.SCSQAUTH
//OUTPUTA DD DSN=SAMPLE.UTILITY.COPYA,DISP=(NEW,CATLG),
// SPACE=(CYL,(5,1),RLSE),UNIT=SYSDA,
// DCB=(RECFM=VBS,BLKSIZE=23200)
//CSQUOUT DD DSN=SAMPLE.UTILITY.COPY3,DISP=(NEW,CATLG),
// SPACE=(CYL,(5,1),RLSE),UNIT=SYSDA,
// DCB=(RECFM=VBS,BLKSIZE=23200)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
* COPY WHOLE PAGESET TO 'CSQUOUT'
COPY PSID(03)
* COPY ONE QUEUE TO 'OUTPUT'
COPY QUEUE(ABC123A) DDNAME(OUTPUTA)
/*
```

Figure 79. Sample JCL for the CSQUTIL COPY functions. The sample shows two instances of the COPY function—one COPY to the default DDNAME, CSQUOUT; the other to DDNAME OUTPUTA, which overrides CSQUOUT.

Usage notes

1. The queues or page set involved must not be in use when the function is invoked.
2. If you want to operate on a range of page sets, you must repeat the COPY function for each page set.
3. The function operates only on local queues.
4. A COPY PSID function is considered successful only if it successfully copies all the queues on the page set.
5. If you try to copy an empty queue (whether explicitly by COPY QUEUE or because there are one or more empty queues on a page set that you are copying), data indicating this is written to the sequential data set, and the copy is considered to be a success. However, if you attempt to copy a non-existent queue, or a page set containing no queues, the COPY function fails, and no data is written to the data set.
6. If COPY fails, no further CSQUTIL functions will be attempted.
7. To use the COPY function more than once in the job, specify different DDnames and data sets for each invocation of the function, or specify a sequential data set and DISP=MOD in the DD statements.
8. You need the necessary authority to use the command server queues (SYSTEM.COMMAND.INPUT, SYSTEM.COMMAND.REPLY.MODEL, and SYSTEM.CSQUTIL.*), to use the DISPLAY QUEUE and DISPLAY STGCLASS MQSC commands, and to use the MQSeries API to browse messages on the

Queue management functions

queues that you wish to copy. See “Syncpoints” on page 245 for information about possible syncpoint restrictions.

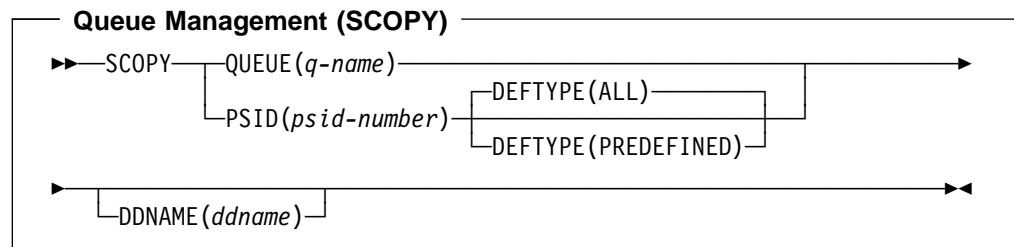
Copying queues into a data set while the queue manager is not running (SCOPY)

Use the SCOPY function to copy queued messages to a sequential data set when the queue manager is not running, without destroying any messages in the original queues.

The scope of the SCOPY function is determined by the keyword that you specify in the first parameter. You can either copy all the messages from a named queue, or all the messages from all the queues on a named page set.

Use the complementary function, LOAD, to restore the messages to their respective queues.

To use the SCOPY function, DDname CSQP0000 must specify the data set with page set zero for the subsystem required.



Keywords and parameters

QUEUE(*q-name*)

QUEUE specifies that messages in the named queue are to be copied. The keyword QUEUE can be abbreviated to Q.

q-name specifies the name of the queue to be copied. This name is case-sensitive.

DDname CSQP00*nn* must specify the data set with page set *nn* for the subsystem required, where *nn* is the number of the page set where the queue resides.

PSID(*psid-number*)

PSID specifies that all the messages in all the queues in the specified page set are to be copied.

psid-number is the page set identifier, which specifies the page set to be used. This identifier is a two-digit integer (whole number) representing a single page set.

DDname CSQP00*psid-number* must specify the data set with the required page set for the subsystem required.

DEFTYPE

Specifies whether to copy dynamic queues:

ALL Copy all queues; this is the default.

PREDEFINED Do not include dynamic queues; this is the same set of queues that are selected by the COMMAND and SDEFS functions with the MAKEDEF parameter.

This parameter is only valid if you specify PSID.

DDNAME

Specifies that the messages are to be copied to a named data set. If this keyword is omitted, the default DDname, CSQUOUT, is used. The keyword DDname can be abbreviated to DD.

ddname specifies the DDname of the destination data set, which is used to store the messages. The record format of this data set must be variable block spanned (VBS).

Do not specify the same DDname on more than one SCOPY statement, unless its DD statement specifies a sequential data set with DISP=MOD.

Example

```
//SCOPY EXEC PGM=CSQUTIL
//STEPLIB DD DISP=SHR,DSN=thlqua1.SCSQANLE
// DD DISP=SHR,DSN=thlqua1.SCSQAUTH
//OUTPUTA DD DSN=SAMPLE.UTILITY.COPYA,DISP=(NEW,CATLG),
// SPACE=(CYL,(5,1),RLSE),UNIT=SYSDA,
// DCB=(RECFM=VBS,BLKSIZE=23200)
//CSQUOUT DD DSN=SAMPLE.UTILITY.COPY3,DISP=(NEW,CATLG),
// SPACE=(CYL,(5,1),RLSE),UNIT=SYSDA,
// DCB=(RECFM=VBS,BLKSIZE=23200)
//CSQP0000 DD DISP=OLD,DSN=pageset.dsname0
//CSQP0003 DD DISP=OLD,DSN=pageset.dsname3
//CSQP0006 DD DISP=OLD,DSN=pageset.dsname6
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
* COPY WHOLE PAGE SET TO 'CSQUOUT'
SCOPY PSID(03)
* COPY ONE QUEUE TO 'OUTPUT' - QUEUE IS ON PAGE SET 6
SCOPY QUEUE(ABC123A) DDNAME(OUTPUTA)
/*
```

Figure 80. Sample JCL for the CSQUTIL SCOPY functions. The sample shows two instances of the SCOPY function—one SCOPY to the default DDNAME, CSQUOUT; the other to DDNAME OUTPUTA, which overrides CSQUOUT.

Usage notes

1. You should not use SCOPY for a queue manager that is running because results will be unpredictable. You can avoid doing this accidentally by using DISP=OLD in the page set DD statement.
2. When you use SCOPY, it is not necessary to specify a queue manager name.
3. If you want to operate on a range of page sets, you must repeat the SCOPY function for each page set.
4. The function operates only on local queues and only for persistent messages.
5. A SCOPY PSID function is considered successful only if it successfully copies all the queues on the page set that have messages; empty queues are ignored. If the page set has no queues with messages, the SCOPY function fails, and no data is written to the data set.
6. If you try to copy an empty queue explicitly by SCOPY QUEUE, data indicating this is written to the sequential data set, and the copy is considered to be a

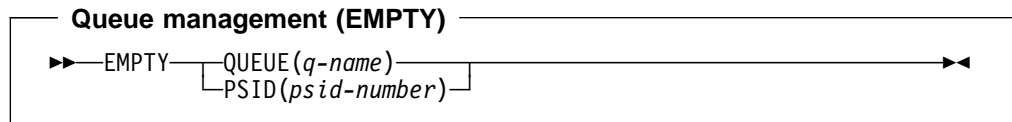
success. However, if you attempt to copy a non-existent queue, the SCOPY function fails, and no data is written to the data set.

7. If the SCOPY function fails, no further CSQUTIL functions are attempted.
8. To use the SCOPY function more than once in the job, specify different DDnames and data sets for each invocation of the function, or specify a sequential data set and DISP=MOD in the DD statements.

Emptying a queue of all messages (EMPTY)

Use the EMPTY function to delete all messages from a named queue or all the queues on a page set. The queue manager must be running. The scope of the function is determined by the keyword that you specify in the first parameter.

Use this function with care. You should only delete messages of which copies have already been made.



Keywords and parameters

You must specify the scope of the EMPTY function. Choose one of these:

QUEUE(*q-name*)

QUEUE specifies that messages are to be deleted from a named queue. This keyword can be abbreviated to Q.

q-name specifies the name of the queue from which messages are to be deleted. This name is case-sensitive.

PSID(*psid-number*)

PSID specifies that all the messages are to be deleted from all queues in the named page set.

psid-number specifies the page-set identifier. This identifier is a two-digit integer (whole number) representing a single page set.

Example

```
//EMPTY EXEC PGM=CSQUTIL,PARM=('CSQ1')
//STEPLIB DD DISP=SHR,DSN=thlqua1.SCSQANLE
// DD DISP=SHR,DSN=thlqua1.SCSQAUTH
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
EMPTY QUEUE(SPARE)
EMPTY PSID(66)
/*
```

Figure 81. Sample JCL for the CSQUTIL EMPTY function

Usage notes

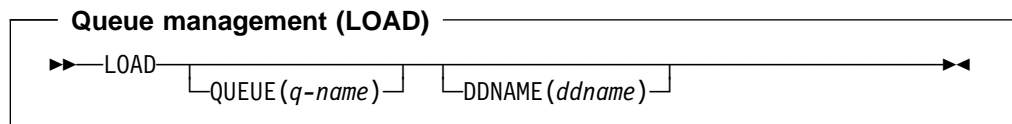
1. The queues or page sets involved must not be in use when the function is invoked.
2. This function operates only on local queues.
3. If you want to operate on a range of page sets, you must repeat the EMPTY function for each page set.
4. You cannot empty the system-command input queue (SYSTEM.COMMAND.INPUT).

5. An EMPTY PSID function is considered successful only if it successfully empties all the queues on the page set.
6. If you empty a queue that is already empty (whether explicitly by EMPTY QUEUE or because there are one or more empty queues on a page set that you are emptying), the EMPTY function is considered to be a success. However, if you attempt to empty a non-existent queue, or a page set containing no queues, the EMPTY function fails.
7. If EMPTY fails or is forced to take a syncpoint, no further CSQUTIL functions will be attempted.
8. You need the necessary authority to use the command server queues (SYSTEM.COMMAND.INPUT, SYSTEM.COMMAND.REPLY.MODEL, and SYSTEM.CSQUTIL.*), to use the DISPLAY QUEUE and DISPLAY STGCLASS MQSC commands, and to use the MQSeries API to get messages from the queues that you wish to empty. See “Syncpoints” on page 245 for information about possible syncpoint restrictions.

Restoring messages from a data set to a queue (LOAD)

The LOAD function of CSQUTIL is complementary to the COPY or SCOPY function. LOAD restores messages from the destination data set of an earlier COPY or SCOPY operation. The queue manager must be running.

The data set can contain messages from one queue only if it was created by COPY or SCOPY QUEUE, or from a number of queues if it was created by COPY PSID or several successive COPY or SCOPY QUEUE operations. Messages are restored to queues with the same name as those from which they were copied. You can specify that the first or only queue is loaded to a queue with a different name. (This would normally be used with a data set created with a single COPY queue operation to restore the messages to a queue with a different name.)



Keywords and parameters

QUEUE(*q-name*)

QUEUE specifies that the messages from the first or only queue on the destination data set of a prior COPY or SCOPY operation are to be loaded to a named queue. Messages from any subsequent queues are loaded to queues with the same names as those they came from. The keyword QUEUE can be abbreviated to Q.

q-name specifies the name of the queue to which the messages are to be loaded. This name is case-sensitive. It must not be a model queue.

DDNAME(*ddname*)

DDNAME specifies that messages are to be loaded from a named data set. This keyword can be abbreviated to DD.

ddname specifies the DDname that identifies the destination data set of a prior COPY or SCOPY operation—from which the messages are to be loaded. This name is not case-sensitive, and can be up to eight characters long.

If you omit DDname(*ddname*) the default DDname, CSQUINP, is used.

Example

```
//LOAD EXEC PGM=CSQUTIL,PARM=('CSQ1')
//STEPLIB DD DISP=SHR,DSN=thlqual.SCSQANLE
// DD DISP=SHR,DSN=thlqual.SCSQAUTH
//OUTPUTA DD DSN=MY.UTILITY.OUTPUTA,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
LOAD QUEUE(ABC123) DDNAME(OUTPUTA)
/*
```

Figure 82. Sample JCL for the CSQUTIL LOAD function

Usage notes

1. To use the LOAD function, the queues or page sets involved must not be in use when the function is invoked.
2. If the data set contains multiple queues, the LOAD function is considered successful only if it successfully loads all the queues on the data set.
3. If LOAD fails, or is forced to take a syncpoint, no further CSQUTIL functions will be attempted.
4. You need the necessary authority to use the MQSeries API to put messages on the queues that you wish to load. See “Syncpoints” on page 245 for information about possible syncpoint restrictions.

The change log inventory utility (CSQJU003)

The MQSeries change log inventory utility runs as an OS/390 batch job to change the bootstrap data set (BSDS). If you need more information about the MQSeries logs and the BSDS, see Chapter 17, “Understanding the log and the bootstrap data set” on page 299.

Through this utility, you can invoke these functions:

Function name	Purpose
NEWLOG	Add active or archive log data sets.
DELETE	Delete active or archive log data sets.
ARCHIVE	Supply passwords for archive logs.
CRESTART	Control the next restart of MQSeries.
CHECKPT	Set checkpoint records.
HIGHRBA	Update the highest written log RBA.

This utility should be run only when MQSeries is not running. This is because the active log data sets named in the BSDS are dynamically added for exclusive use to MQSeries and remain allocated exclusively to MQSeries until it terminates.

Invoking the CSQJU003 utility

The utility runs as an OS/390 batch program. Figure 83 gives an example of the JCL required.

```
//JU003 EXEC PGM=CSQJU003
//STEPLIB DD DISP=SHR,DSN=thlqua1.SCSQANLE
// DD DISP=SHR,DSN=thlqua1.SCSQAUTH
//SYSPRINT DD SYSOUT=*,DCB=BLKSIZE=629
//SYSUT1 DD DISP=SHR,DSN=bsds.dsname
//SYSIN DD *
NEWLOG DSNAME=CSQREPAL.A0001187,COPY1VOL=CSQV04,UNIT=SYSDA,
STARTRBA=3A190000,ENDRBA=3A1F0000,CATALOG=YES,PASSWORD=SYSNKZX
/*
```

Figure 83. Sample JCL to invoke the CSQJU003 utility

Data definition (DD) statements

CSQJU003 requires DD statements with these DDnames:

SYSUT1

This statement is required; it names the BSDS.

SYSUT2

This statement is required if you use dual BSDSs; it names the second copy of the BSDS.

Dual BSDSs and CSQJU003

Each time you run the CSQJU003 utility, the BSDS time stamp field is updated with the current system time. If you run CSQJU003 separately for each copy of a dual copy BSDS, the time stamp fields are not synchronized so that MQSeries fails at startup, issuing error message CSQJ120E. Therefore, if CSQJU003 is used to update dual copy BSDSs, both BSDSs must be updated within a single run of CSQJU003.

SYSPRINT

This statement is required; it names a data set for print output. The logical record length (LRECL) is 125. The block size (BLKSIZE) must be 629.

SYSIN

This statement is required; it names the input data set for statements that specify what the utility is to do. The logical record length (LRECL) is 80.

You can use more than one statement of each type. In each statement, separate the operation name (NEWLOG, DELETE, ARCHIVE, CRESTART) from the first parameter by one or more blanks. You can use parameters in any order; separate them by commas with no blanks. Do not split a parameter description across two SYSIN records.

A statement containing an asterisk in column 1 is considered to be a comment, and is ignored. However, it appears in the output listing. To include a comment or sequence number in a SYSIN record, separate it from the last comma by a blank. When a blank follows a comma, the rest of the record is ignored.

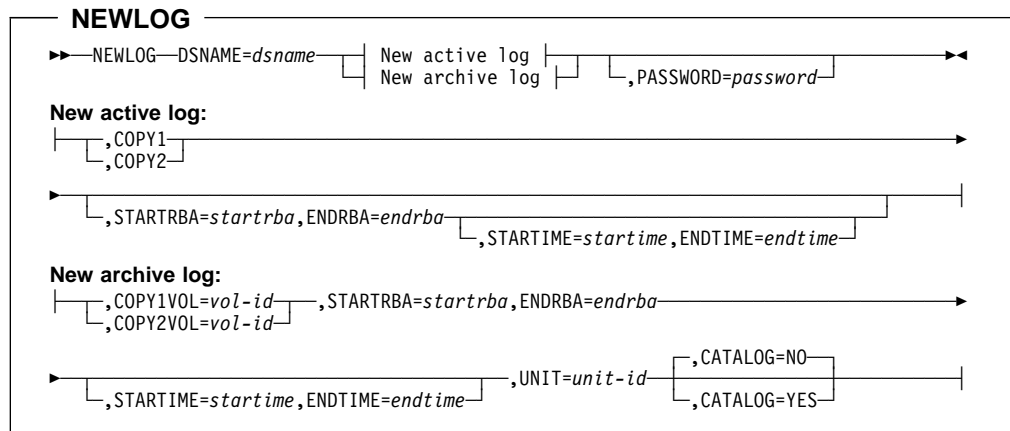
Multiple statement operation

When running CSQJU003, a significant error in any statement causes the control statements for the statement in error and all following statements to be skipped. Therefore, BSDS updates cannot occur for any operation specified in the statement in error, or any following statements. However, all the remaining statements are checked for syntax errors.

Adding information about a data set to the BSDS (NEWLOG)

The NEWLOG function declares one of these data sets:

- A VSAM data set that is available for use as an active log data set.
Use the keywords DSNAME=, COPY1, COPY2, and PASSWORD=.
- An active log data set that is replacing one that encountered an I/O error.
Use the keywords DSNAME=, COPY1, COPY2, STRTRBA=, ENDRBA=, and PASSWORD=.
- An archive log data set volume.
Use the keywords DSNAME=, COPY1VOL=, COPY2VOL=, STARTRBA=, ENDRBA=, UNIT=, CATALOG=, and PASSWORD=.



Keywords and parameters

DSNAME=*dsname*

Names a log data set. *dsname* can be up to 44 characters long.

PASSWORD=*password*

Assigns a password to the data set. It is stored in the BSDS and subsequently used in any access to the active or archive log data sets.

The password is a data set password, and should follow standard VSAM convention: 1 through 8 alphanumeric characters (A through Z, 0 through 9) or special characters (& * + - . ; ' /).

We recommend that you use an ESM such as RACF to provide your data set security requirements.

COPY1

Makes the data set an active log copy-1 data set.

COPY2

Makes the data set an active log copy-2 data set.

STARTRBA=*startrba*

Gives the log RBA (relative byte address within the log) of the beginning of the replacement active log data set or the archive log data set volume specified by DSNAME. *startrba* is a hexadecimal number of up to 12 characters. If you use fewer than 12 characters, leading zeros are added. The RBA can be obtained from messages or by printing the log map.

ENDRBA=*endrba*

Gives the log RBA (relative byte address within the log) of the end of the replacement active log data set or the archive log data set volume specified by DSNAME. *endrba* is a hexadecimal number of up to 12 characters. If you use fewer than 12 characters, leading zeros are added.

STARTIME=*starttime*

Lets you record the start time of the RBA in the BSDS. This is an optional field. The time stamp format (with valid values in parentheses) is as follows:

yyyymmddhhmmsst, where:

yyyy Indicates the year (1993 through 2099)
 ddd Indicates the day of the year (0 through 365; 366 in leap years)
 hh Indicates the hour (0 through 23)
 mm Indicates the minutes (0 through 59)
 ss Indicates the seconds (0 through 59)
 t Indicates tenths of a second

If fewer than 14 digits are specified for the STARTIME and ENDTIME parameter, then trailing zeros will be added.

STARTRBA is required when STARTIME is specified.

ENDTIME=*endtime*

Enables you to record the end time of the RBA in the BSDS. This is an optional field. For time stamp format, see the STARTIME option. The ENDTIME value must be greater than or equal to the value of STARTIME.

COPY1VOL=*vol-id*

The volume serial of the copy-1 archive log data set named after DSNAME.

COPY2VOL=*vol-id*

The volume serial of the copy-2 archive log data set named after DSNAME.

UNIT=*unit-id*

The device type of the archive log data set named after DSNAME.

CATALOG

Tells whether the archive log data set is cataloged:

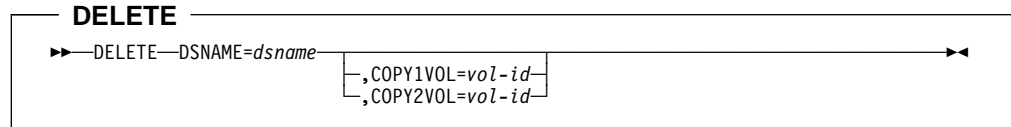
NO Indicates that the archive log data set is not cataloged. All subsequent allocations of the data set are made using the unit and volume information specified on the function. The default is NO.

YES Indicates that the archive log data set is cataloged. A flag is set in the BSDS indicating this, and all subsequent allocations of the data set are made using the catalog.

MQSeries requires that all archive log data sets on DASD be cataloged. Select CATALOG=YES if the archive log data set is on DASD.

Deleting information about a data set from the BSDS (DELETE)

Use the DELETE function to delete all information about a specified log data set or data set volume from the bootstrap data sets. For example, you can use this function to delete outdated archive log data sets.



Keywords and parameters

DSNAME=*dsname*

Specifies the name of the log data set. *dsname* can be up to 44 characters long.

COPY1VOL=*vol-id*

The volume serial number of the copy-1 archive log data set named after DSNAME.

COPY2VOL=*vol-id*

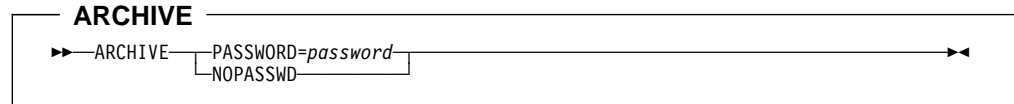
The volume serial number of the copy-2 archive log data set named after DSNAME.

Supplying a password for archive log data sets (ARCHIVE)

Use the ARCHIVE function to give a password to all archive data sets created after this operation. This password is added to the installation's OS/390 password data set each time a new archive log data set is created.

Use the NOPASSWD keyword to remove the password protection for all archives created after the archive operation.

Note: You should normally use an ESM, such as RACF, if you want to implement security on any MQSeries data sets.



Keywords and parameters

PASSWORD=*password*

PASSWORD specifies that a password is to be assigned to the archive log data sets.

password specifies the password, which is a data set password and it must follow the standard VSAM convention; that is, 1 through 8 alphanumeric characters (A through Z, 0 through 9) or special characters (& * + - . ; ' /).

NOPASSWD

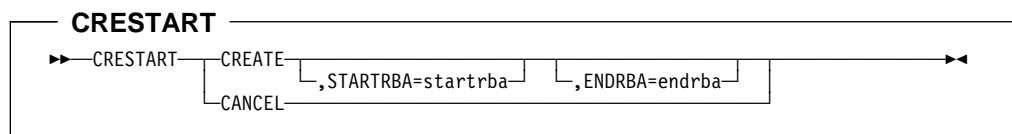
Specifies that archive password protection is not to be active for all archives created after this operation. No other keyword can be used with NOPASSWD.

Controlling the next restart (CRESTART)

Use the CRESTART function to control the next restart of MQSeries, either by creating a new conditional restart control record or by cancelling the one currently active. These records limit the scope of the log data that will be used during restart. Any existing conditional restart control record governs every restart until one of these events occurs:

- A restart operation completes
- A CRESTART CANCEL is issued
- A new conditional restart control record is created

Attention: This can override MQSeries efforts to maintain data in a consistent state. You would normally only use this function when implementing the disaster recovery process described in “Preparing for disaster recovery” on page 312, or under the guidance of IBM service.



Keywords and parameters

CREATE

Creates a new conditional restart control record. When the new record is created, the previous control record becomes inactive.

CANCEL

Makes the currently active conditional restart control record inactive. The record remains in the BSDS as historical information.

No other keyword can be used with CANCEL.

STARTRBA=*startrba*

Gives the earliest RBA of the log to be used during restart. If you omit this option, MQSeries determines the beginning of the log range.

startrba is a hexadecimal number of up to 12 digits. If you use fewer than 12 digits, leading zeros are added. The RBA can be obtained from messages or by printing the log map.

ENDRBA=*endrba*

Gives the last RBA of the log to be used during restart, and the starting RBA of the next active log to be written after restart. Any log information in the bootstrap data set and the active logs, with an RBA greater than *endrba*, is discarded. If you omit this option, MQSeries determines the end of the log range.

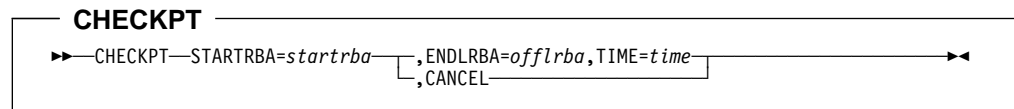
endrba is a hexadecimal number of up to 12 digits. If you use fewer than 12 digits, leading zeros are added.

The value of ENDRBA must be a multiple of 4096. (The hexadecimal value must end in 000.) Also, the value must be greater than or equal to the value of STARTRBA. If STARTRBA and ENDRBA are equal, the next restart is a 'cold start'; that is, no log records are processed during restart.

Setting checkpoint records (CHECKPT)

Use the CHECKPT function to add or delete a record in the BSDS checkpoint queue. Use the STARTRBA and ENDRBA keywords to add a record, or the STARTRBA and CANCEL keywords to delete a record.

Attention: This can override MQSeries efforts to maintain data in a consistent state. You would normally only use this function when implementing the disaster recovery process described in “Preparing for disaster recovery” on page 312, or under the guidance of IBM service.



Keywords and parameters

STARTRBA=*startrba*

Indicates the start checkpoint log record.

startrba is a hexadecimal number of up to 12 digits. If you use fewer than 12 digits, leading zeros are added. The RBA can be obtained from messages or by printing the log map.

ENDRBA=*endrba*

Indicates the end checkpoint log record corresponding to the start checkpoint record.

endrba is a hexadecimal number of up to 12 digits. If you use fewer than 12 digits, leading zeros are added. The RBA can be obtained from messages or by printing the log map.

TIME=*time*

Gives the time the start checkpoint record was written. The time stamp format (with valid values in parentheses) is as follows:

yyyydddhhmmsst, where:

- yyyy Indicates the year (1993 through 2099)
- ddd Indicates the day of the year (0 through 365; 366 in leap years)
- hh Indicates the hour (0 through 23)
- mm Indicates the minutes (0 through 59)
- ss Indicates the seconds (0 through 59)
- t Indicates tenths of a second

If fewer than 14 digits are specified for the TIME parameter, then trailing zeros are added.

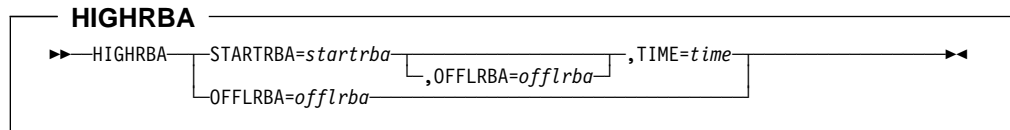
CANCEL

Deletes the checkpoint queue record containing a starting RBA that matches the RBA specified by STARTRBA.

Updating the highest written log RBA (HIGHRBA)

Use the HIGHRBA function to update the highest written log RBA recorded in the BSDS for either the active or archive log data sets. Use the STARTRBA keyword to update the active log, and the OFFLRBA keyword to update the archive log.

Attention: This can override MQSeries efforts to maintain data in a consistent state. You would normally only use this function when implementing the disaster recovery process described in “Preparing for disaster recovery” on page 312, or under the guidance of IBM service.



Keywords and parameters

STARTRBA=*startrba*

Indicates the log RBA of the highest written log record in the active log data set.

startrba is a hexadecimal number of up to 12 digits. If you use fewer than 12 digits, leading zeros are added. The RBA can be obtained from messages or by printing the log map.

TIME=*time*

Specifies when the log record with the highest RBA was written to the log. The time stamp format (with valid values in parentheses) is as follows:

yyydddhhmmsst, where:

- yyyy Indicates the year (1993 through 2099)
- ddd Indicates the day of the year (0 through 365; 366 in leap years)
- hh Indicates the hour (0 through 23)
- mm Indicates the minutes (0 through 59)
- ss Indicates the seconds (0 through 59)
- t Indicates tenths of a second

If fewer than 14 digits are specified for the TIME parameter, then trailing zeros will be added.

OFFLRBA=*offlrba*

Specifies the highest offloaded RBA in the archive log.

offlrba is a hexadecimal number of up to 12 digits. If you use fewer than 12 digits, leading zeros are added. The value must end with hexadecimal 'FFF'.

The print log map utility (CSQJU004)

The MQSeries print log map utility runs as an OS/390 batch program to list this information:

- Log data set name and log RBA association for both copies of all active and archive log data sets
- Active log data sets available for new log data
- Contents of the queue of checkpoint records in the bootstrap data set (BSDS)
- Contents of the quiesce history record
- System and utility time stamps
- Passwords for the active and archive log data sets, if provided

The CSQJU004 program can be run regardless of whether MQSeries is running. However, if MQSeries is running, consistent results from the utility can be ensured only if both the utility and the MQSeries subsystem are running under control of the same OS/390 system.

To use this utility, the user ID of the job must have the requisite security authorization, or, if the BSDS is password protected, the appropriate VSAM password for the data set.

Invoking the CSQJU004 utility

Figure 84 shows an example of the JCL used to invoke the CSQJU004 utility:

```
//JU004 EXEC PGM=CSQJU004
//STEPLIB DD DISP=SHR,DSN=thlqua1.SCSQANLE
// DD DISP=SHR,DSN=thlqua1.SCSQAUTH
//SYSPRINT DD SYSOUT=*,DCB=BLKSIZE=629
//SYSUT1 DD DISP=SHR,DSN=bsds.dsname
```

Figure 84. Sample JCL to invoke the CSQJU004 utility

Data definition statements

The CSQJU004 utility requires DD statements with the following DDnames:

SYSUT1 This statement is required to specify and allocate the bootstrap data set. If the BSDS must be shared with a concurrently executing MQSeries online subsystem, use DISP=SHR on the DD statement.

SYSPRINT

This statement is required to specify a data set or print spool class for print output. The logical record length (LRECL) is 125. The block size (BLKSIZE) must be 629.

“Finding out what the BSDS contains” on page 319 describes the output.

The log print utility (CSQ1LOGP)

You can use this utility to print information contained in the logs or the BSDS.

Invoking the CSQ1LOGP utility

You run the MQSeries log print utility as an OS/390 batch program. You can specify:

- A bootstrap data set (BSDS)
- Active logs (with no BSDS)
- Archive logs (with no BSDS)

Sample JCL to invoke the CSQ1LOGP utility is shown in figures 85, 86, and 87.

These DD statements should be provided:

SYSPRINT

All error messages, exception conditions and the detail report are written to this data set. The logical record length (LRECL) is 131.

SYSIN

Input selection criteria can be specified in this data set (see "Input control parameters" on page 267 for more information).

SYSSUMRY

If a summary report is requested, the output is written to this data set. The logical record length (LRECL) is 131.

BSDS

Name of the bootstrap data set (BSDS).

ACTIVEN

Name of an active log data set you want to print (n=number).

ARCHIVE

Name of an archive log data set you want to print.

Note: The utility will not run if MQSeries is active and you are trying to process active logs (using a BSDS or the active logs directly).

```
//PRTLOG EXEC PGM=CSQ1LOGP
//STEPLIB DD DISP=SHR,DSN=th1qua1.SCSQANLE
// DD DISP=SHR,DSN=th1qua1.SCSQLOAD
//BSDS DD DSN=bsds.dsname,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSSUMRY DD SYSOUT=*
//SYSIN DD *
insert your input control statements here
/*
```

Figure 85. Sample JCL to invoke the CSQ1LOGP utility using a BSDS

```

//PRTLOG EXEC PGM=CSQ1LOGP
//STEPLIB DD DISP=SHR,DSN=th1qua1.SCSQANLE
// DD DISP=SHR,DSN=th1qua1.SCSQLOAD
//ACTIVE1 DD DSN=bsds.logcopy1.ds01,DISP=SHR
//ACTIVE2 DD DSN=bsds.logcopy1.ds02,DISP=SHR
//ACTIVE3 DD DSN=bsds.logcopy1.ds03,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSSUMRY DD SYSOUT=*
//SYSIN DD *
    insert your input control statements here
/*

```

Figure 86. Sample JCL to invoke the CSQ1LOGP utility using active log data sets

```

//PRTLOG EXEC PGM=CSQ1LOGP
//STEPLIB DD DISP=SHR,DSN=th1qua1.SCSQANLE
// DD DISP=SHR,DSN=th1qua1.SCSQLOAD
//ARCHIVE DD DSN=bsds.archive1.ds01,DISP=SHR
// DD DSN=bsds.archive1.ds02,DISP=SHR
// DD DSN=bsds.archive1.ds03,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSSUMRY DD SYSOUT=*
//SYSIN DD *
    insert your input control statements here
/*

```

Figure 87. Sample JCL to invoke the CSQ1LOGP utility using archive logs

Input control parameters

The keywords that you can use in the SYSIN data set are described below:

RBASTART(hexadecimal-constant)

Specifies the log RBA from which to begin processing. If you are using a BSDS, this parameter must be specified.

Normally you are only interested in the most recent additions to the log. Therefore, do not specify a value of zero. If you do, you create an enormous amount of data, most of which is of no interest to you.

You can also use the forms STARTRBA or ST. Specify this keyword only once.

RBAEND(hexadecimal-constant)

Specifies the last valid log RBA that is to be processed. If this keyword is omitted, processing continues to the end of the log (FFFFFFFFFFFF).

You can also use the forms ENDRBA or EN. Specify this keyword only once.

PAGESET(decimal-integer)

Specifies a page-set identifier. The number should be in the range 00 through 99. Only log records associated with the page set you specify will be processed.

URID (hexadecimal-constant)

Specifies a hexadecimal unit of recovery identifier. Changes to data occur in the context of an MQSeries unit of recovery. A unit of recovery is identified on the log via a BEGIN UR record. The log RBA of that BEGIN UR record is the URID value you must use. If you know the URID for a given UR that you are interested in, you can limit the extraction of information from the MQSeries log to that URID.

The hexadecimal-constant can consist of 1 through 12 characters (6 bytes), and leading zeros are not required.

You can specify a maximum of 10 URID keywords in any given CSQ1LOGP job. To narrow the search, you can specify URID keywords in a job that contains other keywords.

RM (resource_manager)

Specifies a particular resource manager. Only records associated with this resource manager will be processed. Valid values for this keyword are:

- RECOVERY
- DATA
- BUFFER

SUMMARY(YES|NO|ONLY)

Specifies whether a summary report is to be produced or not:

YES Produce a summary report in addition to the detail report.

NO Do not produce a summary report.

ONLY Produce only a summary report (no detail report).

Output

The detail report begins by echoing the input selection criteria specified via SYSIN, and then prints each valid log record encountered. Definitions of keywords in the detail report are as follows:

RM	Resource manager that wrote the log record.
TYPE	Type of log record.
URID	BEGIN UR for this unit of recovery, see the description above.
LRID	Logical record identifier in the form: AAAAAAA.BBBBBBCC where: AAAAAAA Is the page set number. BBBBBB Is the relative page number in the page set. CC Is the relative record number on the page.
SUBTYPE	Subtype of the log record type.
CHANGE LENGTH	Length of the logged change.
CHANGE OFFSET	Start position of the change.
BACKWARD CHAIN	Pointer to the previous page.
FORWARD CHAIN	Pointer to the next page.
RECORD LENGTH	Length of the inserted record.

Part 6. Backup, recovery, and restart

Chapter 15. Introducing some recovery concepts	273
How changes are made to data	273
Units of recovery	273
Backing out work	274
How consistency is maintained	275
Consistency with CICS or IMS	275
How consistency is maintained after an abnormal termination	277
Chapter 16. Understanding termination and restart	279
What happens during termination	279
Normal termination	279
Abnormal termination	280
Connections and threads	280
Active threads	280
In-doubt threads	281
What happens during restart and recovery	281
Rebuilding queue indexes	282
How in-doubt units of recovery are resolved	283
How in-doubt units of recovery are resolved from CICS	283
How in-doubt units of recovery are resolved from IMS	284
How in-doubt units of recovery are resolved from RRS	285
Recovering CICS units of recovery manually	286
Recovering IMS units of recovery manually	288
Recovery procedure	288
Recovering RRS units of recovery manually	290
What happens when the CICS adapter restarts	291
What happens when the IMS adapter restarts	293
Using the OS/390 Automatic Restart Manager (ARM)	294
ARM couple data sets	294
ARM policies	295
Registering with ARM	296
Using ARM in an MQSeries network	296
Chapter 17. Understanding the log and the bootstrap data set	299
What logs are	299
Archiving	299
Dual logging	300
Log data	300
Unit-of-recovery log records	301
Checkpoint records	301
Page set control records	301
How the log is structured	301
How the logs are written	302
When the active log is written	303
When the archive log is written	303
What the bootstrap data set is for	305
Managing the logs and BSDS	306
Chapter 18. Planning for backup and recovery	307
Introduction	307

General tips for backup and recovery	307
Periodically take backup copies	308
Do not discard archive logs you might need	309
Do not change the DDname-to-page set association	309
Planning your logging environment	309
Use dual logging for your active log, archive log, and bootstrap data sets	309
Planning your archive storage	310
Archiving to tape	310
Archiving to DASD volumes	310
Using SMS with archive log data sets	311
Other recovery considerations	311
Backup and recovery with DFHSM	311
MQSeries recovery and CICS	312
MQSeries recovery and IMS	312
Using Extended Recovery Facility	312
Preparing for disaster recovery	312
Chapter 19. Managing the logs and the bootstrap data set	315
Archiving logs with the ARCHIVE LOG command	315
Discarding archive log data sets	317
Automatic archive log data set deletion	317
Manually deleting archive log data sets	318
Printing log records	319
Finding out what the BSDS contains	319
Changing the BSDS	321
Recovering logs	323
Chapter 20. Managing page sets	325
Adding a page set to a queue manager	325
When one of your page sets becomes full	325
How to expand a page set	326
How to balance loads on page sets	327
Load balancing by moving queues	328
How to reduce a page set	330
Backing up and recovering page sets	331
Creating a point of recovery	331
Recovering page sets	333
Backing up and restoring queues using CSQUTIL	335
Chapter 21. Example recovery scenarios	337
Dealing with active log problems	338
Out-of-space in active logs and delays in off-loading	338
Dual logging is lost	339
Write I/O errors on an active log data set	339
I/O errors occur while reading the active log	340
Active log stopped	342
Dealing with archive log problems	343
Allocation problems	343
Write I/O errors on the archive log during off-load	343
Read I/O errors on the archive data set while MQSeries is restarting	344
Insufficient DASD space to complete off-load processing	344
Dealing with BSDS errors	346
I/O error	346
Error occurs while opening the BSDS	347

Unequal time stamps	347
Out of synchronization	348
Log content does not agree with the BSDS information	349
BSDS recovery	350
Dealing with page set problems	353
Page set I/O errors	353
Page set full	353
Restarting if you have lost your log data sets	355
Performing a cold start of MQSeries	356
Dealing with IMS-related problems	357
IMS application problem	357
IMS is not operational	357
IMS is unable to connect to MQSeries	358
Dealing with hardware errors	359

Chapter 15. Introducing some recovery concepts

This chapter describes the background concepts of recovery and restart, which you must understand before reading the other chapters in this part of the book.

The chapter contains the following sections:

- “How changes are made to data”
- “How consistency is maintained” on page 275

How changes are made to data

MQSeries must interact with other subsystems to keep all the data consistent. This section discusses *units of recovery*; what they are and how they are used in *back outs*.

Units of recovery

A *unit of recovery* is the processing done by a single MQSeries subsystem for an application program, that changes MQSeries data from one point of consistency to another. A *point of consistency* – also called a *syncpoint* or *commit point* – is a point in time when all the recoverable data that an application program accesses is consistent.

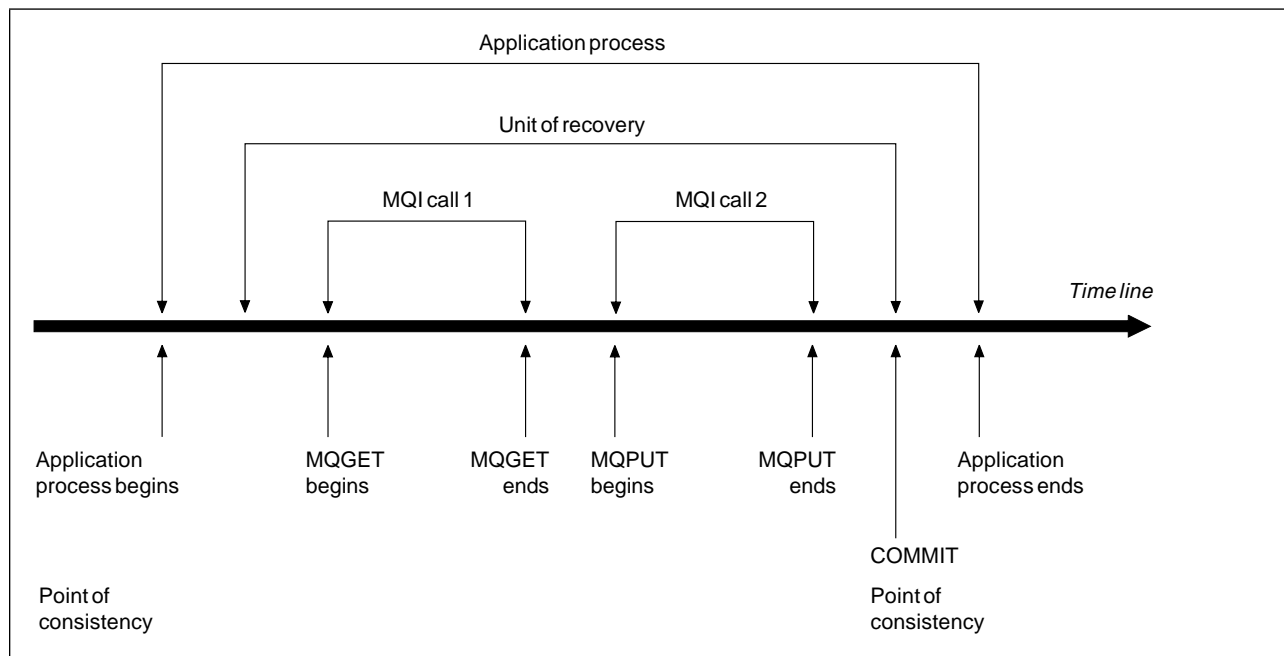


Figure 88. A unit of recovery within an application program. Typically, the unit of recovery consists of more than one MQI call. More than one unit of recovery can occur within an application program.

A unit of recovery begins with the first change to the data after the beginning of the program or following the previous point of consistency; it ends with a later point of consistency. Figure 88 shows the relationship between units of recovery, the point of consistency, and an application program. In this example, the application program makes changes to queues through MQI calls 1 and 2. The application program can include more than one unit of recovery or just one. However, any complete unit of recovery ends in a commit point.

Recovery concepts

For example, a bank transaction transfers funds from one account to another. First, the program subtracts the amount from the first account, account A. Then, it adds the amount to the second account, B. After subtracting the amount from A, the two accounts are inconsistent and MQSeries cannot commit. They become consistent when the amount is added to account B. When both steps are complete, the program can announce a point of consistency through a commit, making the changes visible to other application programs.

Normal termination of an application program automatically causes a point of consistency. Some program requests in CICS and IMS programs also cause a point of consistency, for example, EXEC CICS SYNCPOINT.

Backing out work

If an error occurs within a unit of recovery, MQSeries removes any changes to data, returning the data to its state at the start of the unit of recovery; that is, MQSeries backs out the work. The events are shown in Figure 89.

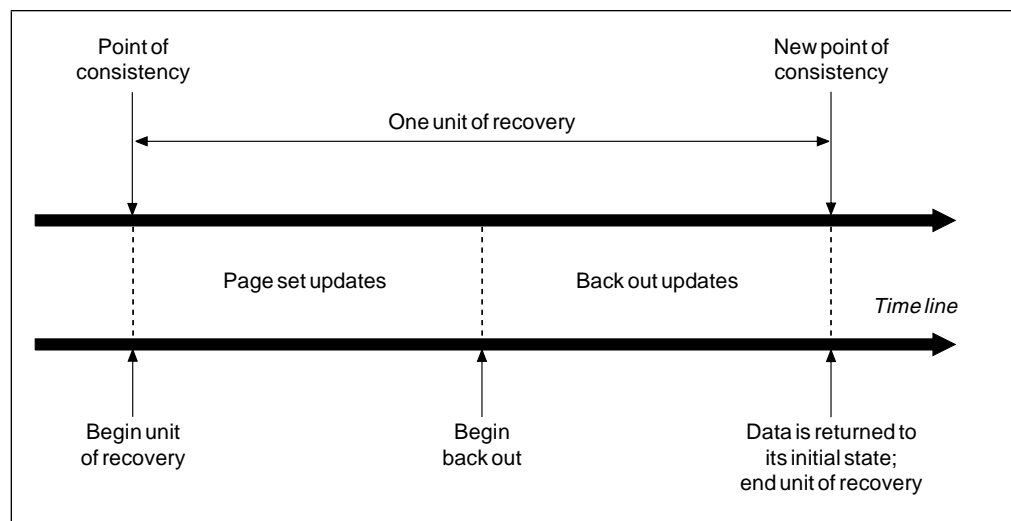


Figure 89. A unit of recovery showing back out

How consistency is maintained

If data in an MQSeries subsystem is to be consistent with batch, CICS, IMS, or TSO, any data changed in one must be matched by a change in the other. Before one system commits the changed data, it must know that the other system can make the corresponding change. So, the systems must communicate.

During a *two-phase commit* (for example under CICS), one subsystem coordinates the process. That subsystem is called the *coordinator*, the other is the *participant*. CICS or IMS is always the coordinator in interactions with MQSeries, and MQSeries is always the participant. In the batch or TSO environment, MQSeries can participate in two-phase commit protocols coordinated by OS/390 RRS.

During a *single-phase commit* (for example under TSO or batch), MQSeries is always the coordinator in the interactions and completely controls the commit process.

Consistency with CICS or IMS

The CICS–MQSeries connection supports the following syncpoint protocols:

- Two-phase commit – for transactions that update resources owned by more than one resource manager.

This is the standard distributed syncpoint protocol. It involves more logging and message flows than a single-phase commit.

- Single-phase commit – for transactions that update resources owned by a single resource manager.

This protocol is optimized for logging and message flows.

- Bypass of syncpoint – for transactions that involve MQSeries but which do nothing in the queue manager that requires a syncpoint.

In each case, CICS acts as the syncpoint manager.

The stages of the two-phase commit that MQSeries uses to communicate with CICS or IMS are:

1. In phase 1 each system determines independently whether it has recorded enough recovery information in its log, and can commit its work.

At the end of the phase, the systems communicate. If they agree, each begins the next phase.

2. In phase 2, the changes are made permanent. If one of the systems abends during phase 2, the operation is completed by the recovery process during restart.

Illustration of the two-phase commit process

Figure 90 illustrates the two-phase commit process. Events in the CICS or IMS coordinator are shown on the upper line, events in MQSeries on the lower line.

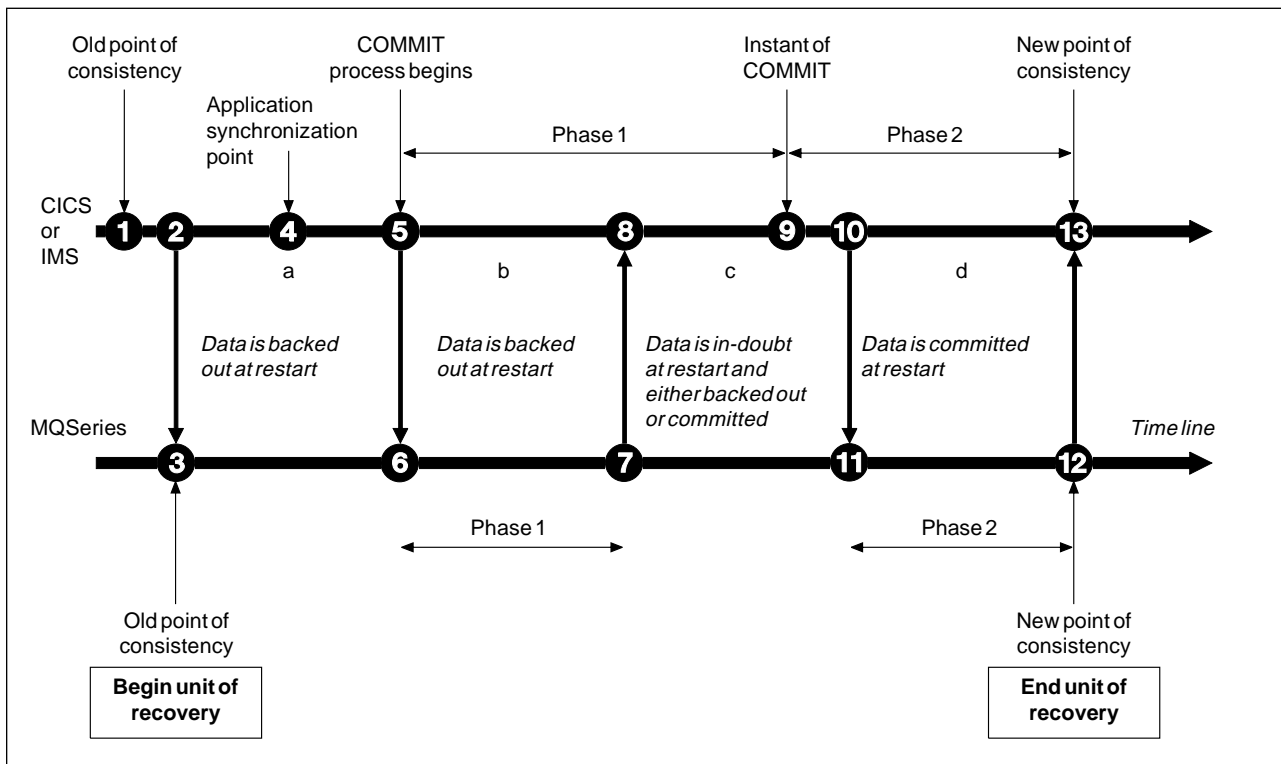


Figure 90. The two-phase commit process

The numbers in the following discussion are linked to those in the figure.

1. The data in the coordinator is at a point of consistency.
2. An application program in the coordinator calls MQSeries to update a queue by adding a message.
3. This starts a unit of recovery in MQSeries.
4. Processing continues in the coordinator until an application synchronization point is reached.
5. The coordinator then starts commit processing. CICS programs use a SYNCPOINT command or a normal application termination to start the commit. IMS programs can start the commit by using a CHKP call, a SYNC call, a GET UNIQUE call to the IOPCB, or a normal application termination. Phase 1 of commit processing begins.
6. As the coordinator begins phase 1 processing, so does MQSeries.
7. MQSeries successfully completes phase 1, writes this fact in its log, and notifies the coordinator.
8. The coordinator receives the notification.

9. The coordinator successfully completes its phase 1 processing. Now both subsystems agree to commit the data changes, because both have completed phase 1 and could recover from any errors. The coordinator records in its log the instant of commit – the irrevocable decision of the two subsystems to make the changes.

The coordinator now begins phase 2 of the processing – the actual commitment.

10. The coordinator notifies MQSeries to begin its phase 2.
11. MQSeries logs the start of phase 2.
12. Phase 2 is successfully completed, and this is now a new point of consistency for MQSeries. MQSeries then notifies the coordinator that it has finished its phase 2 processing.
13. The coordinator finishes its phase 2 processing. The data controlled by both subsystems is now consistent and available to other applications.

There are occasions when CICS or IMS invokes MQSeries when no MQSeries resource has been altered since the completion of the last commit process. This can happen, for example, when a SYNCPOINT request is issued after a series of commands have been processed, or when end-of-task is reached immediately after a SYNCPOINT request has been issued. When this occurs, the MQSeries subsystem performs both phases of the two-phase commit during the first commit phase, and records that the user or job is read-only in relation to its MQSeries processing.

How consistency is maintained after an abnormal termination

When MQSeries is restarted after an abnormal termination, it must determine whether to commit or to back out units of recovery that were active at the time of termination. For certain units of recovery, MQSeries has enough information to make the decision. For others, it does not, and must get information from the coordinator when the connection is reestablished.

Figure 90 shows four periods within the two phases: a, b, c, and d. The status of a unit of recovery depends on the period in which termination happened. The status can be:

In-flight	MQSeries ended before finishing phase 1 (period a or b); during restart, MQSeries backs out the updates.
In-doubt	MQSeries ended after finishing phase 1 and before starting phase 2 (period c); only the coordinator knows whether the error happened before or after the commit (point 9). If it happened before, MQSeries must back out its changes; if it happened after, MQSeries must make its changes and commit them. At restart, MQSeries waits for information from the coordinator before processing this unit of recovery.
In-commit	MQSeries ended after it began its own phase 2 processing (period d); it makes committed changes.
In-backout	MQSeries ended after a unit of recovery began to be backed out but before the process was complete (not shown in the figure); during restart, MQSeries continues to back out the changes.

Maintaining consistency

Chapter 16. Understanding termination and restart

This chapter describes what happens when MQSeries terminates abnormally, and how to start it again. It contains the following sections:

- “What happens during termination”
- “Connections and threads” on page 280
- “What happens during restart and recovery” on page 281
- “How in-doubt units of recovery are resolved” on page 283
- “Recovering CICS units of recovery manually” on page 286
- “Recovering IMS units of recovery manually” on page 288
- “Recovering RRS units of recovery manually” on page 290
- “What happens when the CICS adapter restarts” on page 291
- “What happens when the IMS adapter restarts” on page 293
- “Using the OS/390 Automatic Restart Manager (ARM)” on page 294

More information about restarting MQSeries is available in:

- “Restarting if you have lost your log data sets” on page 355
- “Performing a cold start of MQSeries” on page 356

What happens during termination

MQSeries terminates normally in response to the command STOP QMGR. If MQSeries stops for any other reason, the termination is considered to be abnormal.

Normal termination

In a normal termination, MQSeries stops all activity in an orderly way. You can use either STOP QMGR MODE(QUIESCE), STOP QMGR MODE(FORCE), or STOP QMGR MODE(RESTART). The effects are given in Table 18:

Thread type	QUIESCE	FORCE	RESTART
Active threads	Run to completion	Back out	Back out
New threads	Can start	Not permitted	Not permitted
New connections	Not permitted	Not permitted	Not permitted
Deregister from ARM¹	Yes	Yes	No
Note:			
1. See “Using the OS/390 Automatic Restart Manager (ARM)” on page 294 for more information.			

Batch applications are notified if a termination occurs while the application is still connected.

With CICS, a current thread runs only to the end of the unit of recovery. With CICS, STOP QMGR MODE(QUIESCE) stops the CICS adapter, and so if an active task contains more than one unit of recovery, the task does not necessarily run to completion.

Connections and threads

When you issue the STOP QMGR MODE(FORCE) or MODE(RESTART) command, no new threads are allocated, and work on connected threads is rolled back. Using these modes can create in-doubt units of recovery for threads that are between commit processing phases. They are resolved when MQSeries is reconnected with the controlling CICS, IMS, or RRS subsystem.

When you stop MQSeries, in any mode, the steps are:

- Connections are ended.
- MQSeries ceases to accept commands.
- MQSeries ensures that any outstanding updates to the page sets are completed.
- The DISPLAY USAGE command is issued internally by MQSeries so that the restart RBA is recorded on the OS/390 console log.
- The shutdown checkpoint is taken and the BSDS is updated.

Terminations that specify MODE (QUIESCE) do not affect in-doubt units of recovery. Any unit that is in doubt remains in doubt.

Abnormal termination

An abnormal termination can leave data in an inconsistent state, for example:

- A unit of recovery has been interrupted before reaching a point of consistency.
- Committed data has not been written to page sets.
- Uncommitted data has been written to page sets.
- An application program has been interrupted between phase 1 and phase 2 of the commit process, leaving the unit of recovery in doubt.

Note: MQSeries resolves any data inconsistencies arising from abnormal termination during restart and recovery.

Connections and threads

You can use the DISPLAY THREAD command (described in the *MQSeries Command Reference* manual) to get information about connections to MQSeries and their associated threads. You can display active threads to see what is currently happening, or to see what needs to be terminated in order to allow MQSeries to shut down. You can display in-doubt threads to help with recovery. Messages CSQV401I through CSQV406I describe the information returned, these are documented in the *MQSeries for OS/390 Messages and Codes* manual.

Active threads

Each current connection (**MQCONN**) to MQSeries is represented by one active thread, but certain connections (such as those by the CICS adapter or the mover) might have additional threads associated with them. Note that the CTHREAD system parameter in CSQ6SYSP (described in “Using CSQ6SYSP” on page 68) controls the number of *connections* (**MQCONNs**), not the number of threads.

In addition to the connection name, the display includes the associated user ID (if known), the number of MQSeries requests made by a thread, and the thread cross-reference identifier. The number of MQSeries requests is generally 0 for

associated threads. The thread cross-reference identifier is shown in character form if possible, but otherwise in hexadecimal; its format depends on the type of connection:

CICS Contains the CICS thread number, transaction name, and task number – see Table 30 on page 389.

IMS Contains the IMS PST region identifier and PSB name – see Table 31 on page 390.

Batch, TSO, and RRS

Contains nulls or blanks.

Mover

Blank for connections. Associated threads contain 'T' (X'E3') or 'XX**' (X'E7E75C5C') at character position 5.

In-doubt threads

An in-doubt thread is one that is in the second pass of the two-phase commit operation. Resources are held in MQSeries on its behalf. External intervention is needed to resolve the status of in-doubt threads. This might only involve starting the recovery coordinator (CICS, IMS, or RRS) or might involve more, as described in the following sections. They might have been in doubt at the last restart, or they might have become in doubt since the last restart.

The display includes the thread cross-reference identifier, which might be needed if manual recovery is necessary.

What happens during restart and recovery

MQSeries uses its recovery log and the bootstrap data set (BSDS) to determine what to recover when it restarts. The BSDS identifies the active and archive log data sets, and the location of the most recent MQSeries checkpoint in the log.

After MQSeries has been initialized, the restart process takes place as follows:

- Log initialization
- Current status rebuild
- Forward log recovery
- Backward log recovery
- Queue index rebuilding

When recovery has been completed:

- Committed changes are reflected in the data.
- In-doubt activity is reflected in the data. However, the data is locked and cannot be used until MQSeries recognizes and acts on the in-doubt decision. (For more information about in-doubt units of recovery, see “How consistency is maintained after an abnormal termination” on page 277.)
- Interrupted in-flight and in-abort changes have been removed from the queues. The messages are consistent and can be used.
- A new checkpoint has been taken.
- New indexes have been built for indexed queues containing persistent messages (see “Rebuilding queue indexes” on page 282).

Restart and recovery

For an example of the messages that are written to the MQSeries console during restart processing, see “User messages on start-up” on page 192.

Batch applications are not notified when restart occurs *after* the application has requested a connection.

If dual BSDSs are in use, MQSeries checks the consistency of the time stamps in the BSDS:

- If both copies of the BSDS are current, MQSeries tests whether the two time stamps are equal. If they are not, MQSeries issues message CSQJ120E and terminates. This can happen when the two copies of the BSDS are maintained on separate DASD volumes and one of the volumes was restored while MQSeries was stopped. MQSeries detects the situation at restart.

To recover, copy the BSDS with the latest time stamp to the BSDS on the restored volume. Also recover any active log data sets on the restored volume, by copying the dual copy of the active log data sets onto the restored volume. For more detailed instructions, see “Dealing with BSDS errors” on page 346.

- If one copy of the BSDS was deallocated, and logging continued with a single BSDS, a problem could arise. If both copies of the BSDS are maintained on a single volume, and the volume was restored, or if both BSDS copies were restored separately, MQSeries might not detect the restoration. In that case, log records not noted in the BSDS would be unknown to the system.

Rebuilding queue indexes

To increase the speed of **MQGET** operations on a queue where messages are not retrieved sequentially, you can use the **INDXTYPE** queue attribute. This attribute causes MQSeries to maintain an index of the message or correlation identifiers for all the messages on that queue (as described in the *MQSeries Application Programming Guide*).

When MQSeries is restarted, these indexes are rebuilt for each queue. This only applies to persistent messages; nonpersistent messages are deleted at restart. If your indexed queues contain large numbers of persistent messages, this will increase the time taken to restart MQSeries.

No messages are sent to the OS/390 console while these indexes are being rebuilt.

How in-doubt units of recovery are resolved

If MQSeries loses its connection to CICS, IMS, or RRS, it normally attempts to recover all inconsistent objects at restart. The information needed to resolve in-doubt units of recovery must come from the coordinating system. The next section describes the process of resolution.

How in-doubt units of recovery are resolved from CICS

The resolution of in-doubt units has no effect on CICS resources. CICS is in control of recovery coordination and, when it restarts, automatically commits or backs out each unit, depending on whether there was a log record marking the beginning of the commit. The existence of in-doubt objects does not lock CICS resources while MQSeries is being reconnected.

One of the functions of the CICS adapter is to keep data synchronized between CICS and MQSeries. If MQSeries abends while connected to CICS, it is possible for CICS to commit or back out work without MQSeries being aware of it. When MQSeries restarts, that work is termed *in doubt*.

MQSeries cannot resolve these in-doubt units of recovery (that is, commit or back out the changes made to MQSeries resources) until the connection to CICS is restarted or reconnected.

A process to resolve in-doubt units of recovery is initiated during startup of the CICS adapter. The process starts when the adapter requests a list of in-doubt units of recovery. Then:

- The adapter receives a list of in-doubt units of recovery for this connection ID from MQSeries, and passes them to CICS for resolution.
- CICS compares entries from this list with entries in its own. CICS determines from its own list what action it took for each in-doubt unit of recovery.

Under some circumstances, CICS cannot run the MQSeries process to resolve in-doubt units of recovery. When this happens, MQSeries sends one of these messages:

- CSQC404E
- CSQC405E
- CSQC406E
- CSQC407E

followed by the message CSQC408I.

For details of what these messages mean, see the *MQSeries for OS/390 Messages and Codes* manual.

For all resolved units, MQSeries updates the queues as necessary and releases the corresponding locks. Unresolved units can remain after restart. Resolve them by the methods described in “Recovering CICS units of recovery manually” on page 286.

How in-doubt units of recovery are resolved from IMS

Resolving in-doubt units of recovery in IMS has no effect on DL/I resources. IMS is in control of recovery coordination and, when it restarts, automatically commits or backs out incomplete DL/I work. The decision to commit or back out for online regions (non-fast-path) is on the presence or absence of IMS log record types X'3730' and X'3801' respectively. The existence of in-doubt units of recovery does not imply that DL/I records are locked until MQSeries connects.

During restart, MQSeries makes a list of in-doubt units of recovery. IMS builds its own list of residual recovery entries (RREs). The RREs are logged at IMS checkpoints until all entries are resolved.

When in-doubt units are resolved:

1. If MQSeries recognizes that it has marked an entry for commit and IMS has marked it to be backed out, MQSeries issues message CSQQ010E. MQSeries issues this message for all inconsistencies of this type between MQSeries and IMS.
2. If MQSeries has any remaining in-doubt units, the adapter issues message CSQQ008I.

For all resolved units, MQSeries updates queues as necessary and releases the corresponding locks.

MQSeries maintains locks on in-doubt work that was not resolved. This can cause a backlog in the system if important locks are being held. The connection remains active so you can resolve the IMS RREs. Recover the in-doubt threads by the methods described in "Controlling IMS connections" on page 169.

All in-doubt work should be resolved unless there are software or operating problems, such as with an IMS cold start. In-doubt resolution by the IMS control region takes place in two circumstances:

1. At the start of the connection to MQSeries, during which resolution is done synchronously.
2. When a program abends, during which the resolution is done asynchronously.

How in-doubt units of recovery are resolved from RRS

One of the functions of the RRS adapter is to keep data synchronized between MQSeries and other RRS-participating resource managers. If a failure occurs when MQSeries has completed phase one of the commit and is waiting for a decision from RRS (the commit coordinator), the unit of recovery enters the in-doubt state.

When communication is reestablished between RRS and MQSeries, RRS automatically commits or backs out each unit of recovery, depending on whether there was a log record marking the beginning of the commit. MQSeries cannot resolve these in-doubt units of recovery (that is, commit or back out the changes made to MQSeries resources) until the connection to RRS is reestablished.

Under some circumstances, RRS cannot resolve in-doubt units of recovery. When this happens, MQSeries sends one of the following messages to the OS/390 console:

- CSQ3011I
- CSQ3013I
- CSQ3014I
- CSQ3016I

For details of what these messages mean, see the *MQSeries for OS/390 Messages and Codes* manual.

For all resolved units of recovery, MQSeries updates the queues as necessary and releases the corresponding locks. Unresolved units of recovery can remain after restart. Resolve them by the method described in “Recovering RRS units of recovery manually” on page 290.

Recovering CICS units of recovery manually

If the adapter abends, CICS and MQSeries build in-doubt lists either dynamically or during restart, depending on which subsystem caused the abend.

Note: If you use the DFH£INDB sample program to show units of work, you might find that it does not always show MQSeries ones correctly.

When CICS connects to MQSeries, there might be one or more units of recovery, that have not been resolved.

One of the following messages is sent to the console:

- CSQC404E
- CSQC405E
- CSQC406E
- CSQC407E
- CSQC408I

For details of what these messages mean, see the *MQSeries for OS/390 Messages and Codes* manual.

CICS retains details of units of recovery that were not resolved during connection startup. An entry is purged when it no longer appears on the list presented by MQSeries.

Any units of recovery that CICS cannot resolve must be resolved manually using MQSeries commands. This manual procedure is rarely used within an installation, because it is required only where operational errors or software problems have prevented automatic resolution. *Any inconsistencies found during in-doubt resolution must be investigated.*

To recover the units of recovery:

1. Obtain a list of the units of recovery from MQSeries by issuing the following command:

```
+cpf DISPLAY THREAD(*) TYPE(INDOUBT)
```

You receive the following messages:

```
CSQV401I +cpf DISPLAY THREAD REPORT FOLLOWS -
CSQV406I +cpf INDOUBT THREADS
NAME      THREAD-XREF  URID  NID
VICIC3   xref                VICIC3.A75E483235A90900
DISPLAY  THREAD REPORT COMPLETE
CSQ9022I +cpf CSQVDT ' DISPLAY THREAD' NORMAL COMPLETION
```

For CICS connections, the NID consists of the CICS applid and a unique number provided by CICS at the time the syncpoint log entries are written. This unique number is stored in records written to both the CICS system log and the MQSeries log at syncpoint processing time. This value is referred to in CICS as the *recovery token*.

2. Scan the CICS log for entries related to a particular unit of recovery.

Look for a PREPARE record, for the task-related installation where the recovery token field (JCSRMTKN) equals the value obtained from the network ID. The network ID is supplied by MQSeries in the DISPLAY THREAD command output.

The PREPARE record in the CICS log for the units of recovery provides the CICS task number. All other entries on the log for this CICS task can be located using this number.

You can use the CICS journal print utility DFHJUP when scanning the log. For details of using this program, see the *CICS Operations Guide*.

3. Scan the MQSeries log for entries related to a particular unit of recovery.

To do this, scan the MQSeries log to locate the record with the NID required. Then use the URID from this record to obtain the rest of the log records for this unit of recovery.

When scanning the MQSeries log, note that the MQSeries startup message CSQJ001I provides the start RBA for this session.

The print log records program (CSQ1LOGP) can be used for that purpose.

4. If you need to, do in-doubt resolution in MQSeries.

MQSeries can be directed to take the recovery action for a unit of recovery using an MQSeries RESOLVE INDOUBT command.

For information about RESOLVE INDOUBT, see the *MQSeries Command Reference* manual.

To recover all threads associated with *connection-name*, use the NID(*) option.

The command produces one of the following messages showing whether the thread is committed or backed out:

```
CSQV414I +cpf THREAD network-id COMMIT SCHEDULED
CSQV415I +cpf THREAD network-id ABORT SCHEDULED
```

When performing in-doubt resolution, CICS and the adapter are not aware of the commands to MQSeries to commit or back out units of recovery, because only MQSeries resources are affected. However, CICS keeps details about the in-doubt threads that could not be resolved by MQSeries. This information is purged either when the list presented is empty, or when the list does not include a unit of recovery of which CICS has details.

Recovering IMS units of recovery manually

When IMS connects to MQSeries, MQSeries might have one or more in-doubt units of recovery that have not been resolved.

If MQSeries has in-doubt units of recovery that IMS did not resolve, the following message is issued at the IMS master terminal:

```
CSQQ008I  nn units of recovery are still in doubt in queue manager
```

When this message is issued, IMS was either cold-started or it was started with an incomplete log tape. This message can also be issued if MQSeries or IMS terminates abnormally because of a software error or other subsystem failure.

After receiving the CSQQ008I message:

- The connection remains active.
- IMS applications can still access MQSeries resources.
- Some MQSeries resources remain locked out.

If the in-doubt thread is not resolved, IMS message queues can start to build up. If the IMS queues fill to capacity, IMS terminates. Therefore, users must be aware of this potential difficulty and must monitor IMS until the in-doubt units of recovery are fully resolved.

Recovery procedure

Use the following procedure to recover the IMS units of work:

1. Force the IMS log closed, using /SWI OLDS, and then archive the IMS log. Use the utility, DFSERA10, to print the records from the previous IMS log tape. Type X'3730' log records indicate a phase-2 commit request and type X'38' log records indicate an abort request. Record the requested action for the last transaction in each dependent region.
2. Run the DL/I batch job to back out each PSB involved that has not reached a commit point. The process might take some time because transactions are still being processed. It might also lock up a number of records, which could impact the rest of the processing and the rest of the message queues.
3. Produce a list of the in-doubt units of recovery from MQSeries by issuing the following command:

```
DISPLAY THREAD(*) TYPE(INDOUBT)
```

You receive the following messages:

```
CSQV401I +cpf DISPLAY THREAD REPORT FOLLOWS -
CSQV406I +cpf INDOUBT THREADS -
NAME      THREAD-XREF      URID  NID
name      xref                network-id
name      xref                network-id
DISPLAY THREAD REPORT COMPLETE
CSQ9022I +cpf CSQVDT ' DISPLAY THREAD' NORMAL COMPLETION
```

For IMS, the NID consists of the IMS connection name and a unique number provided by IMS. The value is referred to in IMS as the *recovery token*. For more information, see the *IMS Customization Guide*.

4. Compare the NIDs (IMSID plus OASN in hexadecimal) displayed in the DISPLAY THREAD messages with the OASNs (4 bytes decimal) shown in the DFSERA10 output. Decide whether to commit or back out.
5. Perform in-doubt resolution in MQSeries with the RESOLVE INDOUBT command, as follows:

```
RESOLVE INDOUBT(connection-name)  
ACTION(COMMIT|BACKOUT)  
NID(network-id)
```

For information about RESOLVE INDOUBT, see the *MQSeries Command Reference* manual.

To recover all threads associated with *connection-name*, use the NID(*) option.

The command results in one of the following messages to indicate whether the thread is committed or backed out:

```
CSQV414I  THREAD network-id COMMIT SCHEDULED  
CSQV415I  THREAD network-id BACKOUT SCHEDULED
```

When performing in-doubt resolution, IMS and the adapter are not aware of the commands to MQSeries to commit or back out in-doubt units of recovery because only MQSeries resources are affected.

Recovering RRS units of recovery manually

When RRS connects to MQSeries, MQSeries may have one or more in-doubt units of recovery that have not been resolved. If MQSeries has in-doubt units of recovery that RRS did not resolve, one of the following messages is issued at the OS/390 console:

- CSQ3011I
- CSQ3013I
- CSQ3014I
- CSQ3016I

Both MQSeries and RRS provide tools to display information about in-doubt units of recovery, and techniques for manually resolving them.

In MQSeries, use the `DISPLAY THREAD` command to display information about in-doubt MQSeries threads. The output from the command includes RRS unit of recovery IDs for those MQSeries threads that have RRS as a coordinator. This can be used to determine the outcome of the unit of recovery.

Use the MQSeries `RESOLVE INDOUBT` command to resolve the MQSeries in-doubt thread manually. This command can be used to either commit or back out the unit of recovery after you have determined what the correct decision is.

What happens when the CICS adapter restarts

For background information, see Chapter 6, “The MQSeries-CICS adapter” on page 109.

Whenever a connection is broken, the adapter has to go through a *restart phase* during the *reconnect process*. The restart phase resynchronizes resources. Resynchronization between CICS and MQSeries enables *in-doubt units of work* to be identified and resolved.

Resynchronization can be caused by:

- An explicit request from the distributed queuing component
- An implicit request when a connection is made to MQSeries

If the resynchronization is caused by connecting to MQSeries, the sequence of events is:

1. The connection process gets a list of unit of work (UOW) IDs that MQSeries thinks are in doubt.
2. The UOW IDs are displayed on the console in CSQC313I messages.
3. The UOW IDs are passed to CICS.
4. CICS initiates a resynchronization task (CRSY) for each in-doubt UOW ID.
5. The result of the task for each in-doubt UOW is displayed on the console.

You need to check the messages that are displayed during the connect process:

CSQC313I Shows that a UOW is in doubt.

CSQC400I Identifies the UOW and is followed by one of these messages:

- CSQC402I and CSQC403I show that the UOW was resolved successfully (committed or backed out).
- CSQC404E, CSQC405E, CSQC406E, and CSQC407E show that the UOW was not resolved.

CSQC409I Shows that all UOWs were resolved successfully.

CSQC408I Shows that not all UOWs were resolved successfully.

CSQC314I Warns that UOW IDs highlighted with a “*” will not be resolved automatically. These UOWs must be resolved explicitly by the distributed queuing component when it is restarted.

Figure 91 on page 292 shows an example set of restart messages displayed on the OS/390 console.

```

CSQ9022I +cpf CSQYASCP ' START QMGR' NORMAL COMPLETION
+CSQC323I VICIC1 CSQCQCON CONNECT received from TERMID=PB62 TRANID=CKCN
+CSQC303I VICIC1 CSQCQCON CSQCSERV loaded. Entry point is 850E8918.
+CSQC313I VICIC1 CSQCQCON UOWID=VICIC1.A6E5A6F0E2178D25 is in doubt
+CSQC313I VICIC1 CSQCQCON UOWID=VICIC1.A6E5A6F055B2AC25 is in doubt
+CSQC313I VICIC1 CSQCQCON UOWID=VICIC1.A6E5A6EFFD60D425 is in doubt
+CSQC313I VICIC1 CSQCQCON UOWID=VICIC1.A6E5A6F07AB56D22 is in doubt
+CSQC307I VICIC1 CSQCQCON Successful connection to subsystem VC2
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008BAD18) connect
successful.
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008BAA10) connect
successful.
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008BA708) connect
successful.
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008CAE88) connect
successful.
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008CAB80) connect
successful.
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008CA878) connect
successful.
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008CA570) connect
successful.
+CSQC472I VICIC1 CSQCSERV Server subtask (TCB address=008CA268) connect
successful.
+CSQC403I VICIC1 CSQCTRUE Resolved BACKOUT for
+CSQC400I VICIC1 CSQCTRUE UOWID=VICIC1.A6E5A6F0E2178D25
+CSQC403I VICIC1 CSQCTRUE Resolved BACKOUT for
+CSQC400I VICIC1 CSQCTRUE UOWID=VICIC1.A6E5A6F055B2AC25
+CSQC403I VICIC1 CSQCTRUE Resolved BACKOUT for
+CSQC400I VICIC1 CSQCTRUE UOWID=VICIC1.A6E5A6F07AB56D22
+CSQC403I VICIC1 CSQCTRUE Resolved BACKOUT for
+CSQC400I VICIC1 CSQCTRUE UOWID=VICIC1.A6E5A6EFFD60D425
+CSQC409I VICIC1 CSQCTRUE Resynchronization completed successfully

```

Figure 91. Example restart messages

The total number of CSQC313I messages should equal the total number of CSQC402I plus CSQC403I messages. If the totals are not equal, there are UOWs that the connection process cannot resolve. Those UOWs that cannot be resolved are caused by problems with CICS (for example, a cold start) or with MQSeries, or by distributing queuing. When these problems have been fixed, you can initiate another resynchronization by disconnecting and then reconnecting.

Alternatively, you can resolve each outstanding UOW yourself using the MQSeries RESOLVE INDOUBT command and the UOW ID shown in message CSQC400I. You must then initiate a disconnect and a connect to clean up the *unit of recovery descriptors* in CICS. You need to know the correct outcome of the UOW to resolve UOWs manually. See also “Recovering CICS units of recovery manually” on page 286.

All messages that are associated with unresolved UOWs are locked by MQSeries and no Batch, TSO, or CICS task can access them.

Note: If CICS fails and an emergency restart is necessary, *do not* vary the GENERIC APPLID of the CICS system. If you do and then reconnect to MQSeries, data integrity with MQSeries cannot be guaranteed. This is because MQSeries treats the new instance of CICS as a different CICS (because the APPLID is different). In-doubt resolution is then based on the wrong CICS log.

Similarly, if MQSeries fails, do not change the subsystem ID of the MQSeries system.

What happens when the IMS adapter restarts

For background information, see Chapter 9, “The MQSeries-IMS adapter” on page 159.

Whenever the connection to MQSeries is restarted, either following an MQSeries restart, or an IMS /START SUBSYS command, IMS initiates the following resynchronization process:

1. IMS presents the list of unit of work (UOW) IDs that it believes are in doubt to the MQSeries IMS adapter one at a time with a resolution parameter of Commit or Backout.
2. The IMS adapter passes the resolution request to MQSeries and reports the result back to IMS.
3. Having processed all the IMS resolution requests, the IMS adapter gets from MQSeries a list of all UOWs that MQSeries still holds in doubt that were initiated by the IMS system. These are reported to the IMS master terminal in message CSQQ008I.

See “Recovering IMS units of recovery manually” on page 288 for information about recovering these UOWs.

Note: While a UOW is in doubt, any associated MQSeries message is locked by MQSeries and is not available to any application.

Using the OS/390 Automatic Restart Manager (ARM)

The OS/390 Automatic Restart Manager (ARM) is an OS/390 recovery function that can improve the availability of your MQSeries subsystems. When a job or task fails, or the system on which it is running fails, ARM can restart the job or task without operator intervention.

If a queue manager or a channel initiator has failed, ARM restarts it on the same OS/390 image. If OS/390, and hence a whole group of related subsystems and applications have failed, ARM can restart all the failed systems automatically, in a predefined order, on another OS/390 image within the sysplex. This is called a cross-system restart.

The channel initiator should be restarted by ARM only in exceptional circumstances. If the queue manager is restarted by ARM, the channel initiator should be restarted from the CSQINP2 initialization data set (see “Using ARM in an MQSeries network” on page 296).

You can use ARM to restart an MQSeries subsystem that uses LU 6.2 communication protocols on a different OS/390 image within the sysplex in the event of OS/390 failure. (You cannot do this if you use TCP/IP communication protocols.) The network implications of MQSeries ARM restart on a different OS/390 image are discussed in “Using ARM in an MQSeries network” on page 296.

To enable automatic restart:

- You must set up an ARM couple data set.
- You must define the automatic restart actions that you want OS/390 to perform in an *ARM policy*.
- You must start the ARM policy.

Also, MQSeries must register with ARM at startup (this happens automatically).

Note: If you want to restart queue managers in different OS/390 images automatically, every queue manager must be defined as a subsystem in each OS/390 image on which that queue manager might be restarted, with a sysplex wide unique 4-character subsystem name.

ARM couple data sets

You must ensure that you define the couple data sets required for ARM, and that they are online and active before you start any MQSeries subsystem for which you want ARM support. MQSeries automatic ARM registration fails if the couple data sets are not available at MQSeries startup. In this situation, MQSeries assumes that the absence of the couple data set means that you do not want ARM support, and initialization continues.

See the *OS/390 MVS Setting up a Sysplex* manual for information about ARM couple data sets.

ARM policies

ARM functions are controlled by a user-defined *ARM policy*. Each OS/390 image running a queue manager instance that is to be restarted by ARM must be connected to an ARM couple data set with an active ARM policy.

IBM provides a default ARM policy. You can define new policies, or override the policy defaults by using the *administrative data utility* (IXCMIAPU) provided with OS/390. The *OS/390 MVS Setting up a Sysplex* manual describes this utility, and includes full details of how to define an ARM policy.

Figure 92 shows an example of an ARM policy. This sample policy will restart any MQSeries queue manager within a sysplex, in the event that either the queue manager failed, or a whole system failed.

```
//IXCMIAPU EXEC PGM=IXCMIAPU,REGION=2M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DATA TYPE(ARM)
DEFINE POLICY NAME(ARMPOL1) REPLACE(YES)
RESTART_GROUP(DEFAULT)
ELEMENT(*)
    RESTART_ATTEMPTS(0) /* Jobs not to be restarted by ARM */
RESTART_GROUP(GROUP1)
ELEMENT(SYSMQMGRMQ*) /* These jobs to be restarted by ARM */
/*
```

Figure 92. Sample ARM policy

Defining an ARM policy

We recommend that you set up your ARM policy as follows:

- Define RESTART_GROUPS for each queue manager instance which also contain any CICS or IMS subsystems that connect to that queue manager instance. If you use a subsystem naming convention, you might be able to use the '?' and '*' wild-card characters in your element names to achieve the above with minimum definition effort.
- Specify TERMTYPE(ELEMTerm) for your channel initiators to indicate that they will be restarted only if the channel initiator has failed and the OS/390 image has not failed.
- Specify TERMTYPE(ALLTERM) for your queue managers to indicate that they will be restarted if either the queue manager has failed or the OS/390 image has failed.
- Specify RESTART_METHOD(BOTH, PERSIST) for both queue managers and channel initiators. This tells ARM to restart using the JCL it saved (after resolution of system symbols) during the last startup. It tells ARM to do this irrespective of whether the individual element failed, or the OS/390 image failed.
- Accept the default values for all the other ARM policy options.

Activating an ARM policy

To start your automatic restart management policy, issue the following OS/390 command:

```
SETXCF START,POLICY,TYPE=ARM,POLNAME=mypol
```

When the policy is started, all systems connected to the ARM couple data set use the same active policy.

Use the SETXCF STOP command to disable automatic restarts.

Registering with ARM

MQSeries registers automatically as an *ARM element* during its startup phase (subject to ARM availability). It deregisters during its shutdown phase, unless requested not to.

At startup, the queue manager determines whether ARM is available. If it is, MQSeries registers using the name SYSMQMGR*ssid*, where *ssid* is the 4-character queue-manager name, and SYSMQMGR is the element type.

The STOP QMGR MODE(QUIESCE) and STOP QMGR MODE(FORCE) MQSeries commands deregister MQSeries from ARM (if it was registered with ARM at startup). This prevents ARM restarting this queue manager. The STOP QMGR MODE(RESTART) command does not deregister the queue manager from ARM. It is thus eligible for immediate automatic restart.

Each MQSeries channel initiator address space determines whether ARM is available, and if so will register with the element name SYSMQCH*ssid*, where *ssid* is the queue manager name, and SYSMQCH is the element type.

The channel initiator is always deregistered from ARM when it stops normally, and remains registered only if it ends abnormally. The channel initiator is always deregistered if the queue manager fails.

Using ARM in an MQSeries network

You should set up your MQSeries system so that the channel initiators and associated listeners are started automatically when MQSeries is restarted. To ensure fully automatic MQSeries restart on the same OS/390 image for both LU 6.2 and TCP/IP communication protocols:

- Start your channel initiator automatically by adding the appropriate START CHINIT command to the CSQINP2 data set.
- Start your listeners automatically by adding the appropriate START LISTENER command to the CSQINPX data set.

See “Task 10: Customize the initialization input data sets” on page 52 for information about the CSQINP2 data set and “Initialization commands for distributed queuing” on page 53 for information about the CSQINPX data set.

Restarting on a different OS/390 image

If you use only LU 6.2 communication protocols you should also do the following to enable network reconnect after automatic restart of MQSeries on a different OS/390 image within the sysplex:

- Define each MQSeries queue manager within the sysplex with a unique subsystem name.
- Define each channel initiator within the sysplex with a unique LUNAME. This is specified in both the channel initiator parameter module and in the START LISTENER command.

Note: The LUNAME names an entry in the APPC side table, which in turn maps this to the actual LUNAME.

- Set up a shared APPC side table, which is referenced by each OS/390 image within the sysplex. This should contain an entry for each channel initiator's LUNAME. See the *MVS Planning: APPC/MVS Management* manual for information about this.
- Set up an APPCPMxx member of SYS1.PARMLIB for each channel initiator within the sysplex to contain an LUADD to activate the APPC side table entry for that channel initiator. These members should be shared by each OS/390 image. The appropriate SYS1.PARMLIB member is activated by a SET APPC=xx OS/390 command which is issued automatically during ARM restart of MQSeries (and its channel initiator) on a different OS/390 image, as described below.
- Use the LU62ARM keyword of the CSQ6CHIP macro to specify the xx suffix of this SYS1.PARMLIB member for each channel initiator in the channel initiator parameter module. This will cause the channel initiator to issue the required SET APPC=xx OS/390 command to activate its LUNAME.

You should define your ARM policy to restart the channel initiator only if it fails while its OS/390 image stays up. You should not restart the channel initiator automatically if its OS/390 image also fails, but use the CSQINP2 and CSQINPX data sets to start the channel initiator and listeners.

TCP/IP does not currently support moving of an IP address from one OS/390 image to another OS/390 image within the sysplex. If your MQSeries subsystem uses TCP/IP communication protocols, you should not define your ARM policy to restart MQSeries in a different OS/390 image within the sysplex following OS/390 failure.

Chapter 17. Understanding the log and the bootstrap data set

MQSeries maintains *logs* of data changes and significant events as they occur. The *bootstrap data set* (BSDS) stores information about the data sets that contain the logs.

Note: You must set up the log data sets for MQSeries, see “Task 11: Create the bootstrap and log data sets” on page 61.

This chapter serves as a general introduction and reference to the logging mechanism. Day-to-day procedures for managing the log and the BSDS are not described here; you can find them in Chapter 19, “Managing the logs and the bootstrap data set” on page 315.

This chapter contains the following sections:

- “What logs are”
- “How the log is structured” on page 301
- “How the logs are written” on page 302
- “What the bootstrap data set is for” on page 305
- “Managing the logs and BSDS” on page 306

What logs are

MQSeries records all significant events in an *active log* as they occur. The log contains the information needed to recover:

- Persistent messages
- MQSeries objects, such as queues
- The MQSeries subsystem

Note: The log does not contain information for statistics, traces, or performance evaluation.

The active log comprises a series of data sets that are used cyclically. See “How the logs are written” on page 302 for more information.

Archiving

Because the active log is finite, MQSeries copies the contents of each log data set periodically to an *archive log*, which is normally a data set on a direct access storage device (DASD) or a magnetic tape. If there is a subsystem or transaction failure, MQSeries uses the active log and, if necessary, the archive log for recovery.

The archive log can contain up to 1000 sequential data sets. Each data set can be cataloged using the integrated catalog facility (ICF).

Important note about archiving

Archiving is an essential component of MQSeries recovery. If a unit of recovery is a long-running one, it is possible that log records within that unit of recovery will be found in the archive log. In this case, recovery will require data from the archive log. However, if archiving is switched off, the active log records will wrap, overwriting earlier log records. This means that MQSeries will be unable to back out the unit of recovery and messages will be lost. MQSeries will then terminate abnormally with a reason code 00D94012.

Therefore, in a production environment, **you must never switch archiving off**. If you do, you run the risk of losing data after a system or transaction failure. Only if you are running in a test environment should you consider switching archiving off. If you need to do this, use the CSQ6LOGP macro (see “Using CSQ6LOGP” on page 74).

Dual logging

You can configure MQSeries to run with either *single logging* or *dual logging*. With single logging, log records are written once to an active log data set. With dual logging, each log record is written to two different active log data sets. Dual logging minimizes the likelihood of data loss problems during restart. If possible, the two log data sets should be on separate volumes. This reduces the risk of them both being lost if one of the volumes is corrupted or destroyed. If both copies of the log are lost, the probability of data loss is high.

Note: You should always use dual logging and dual BSDSs rather than dual writing to DASD.

Single logging gives you 2 through 53 active log data sets, whereas dual logging gives you 4 through 106. Each active log data set is a single-volume, single-extent VSAM linear data set (LDS).

Although the minimum number of log data sets required is two, in practice you should have at least three, and on a busy system you might need more. This is to allow time for each log data set to be copied to archive before it is reused in the active log cycle.

Log data

The log can contain up to 280 million million (2.8×10^{14}) bytes. Each byte can be addressed by its offset from the beginning of the log, and that offset is known as its *relative byte address* (RBA).

The log is made up of *log records*, each of which is a set of log data treated as a single unit. A log record is identified by the RBA of the first byte of its header; that RBA is called the relative byte address of the record. The RBA uniquely identifies a record that starts at a particular point in the log.

Each log record has a header that gives its type, the MQSeries sub-component that made the record, and, for unit of recovery records, a unit of recovery identifier.

There are three types of log record, described under these headings:

- “Unit-of-recovery log records” on page 301

- “Checkpoint records” on page 301
- “Page set control records”

Unit-of-recovery log records

Most of the log records describe changes to MQSeries queues. All such changes are made within units of recovery.

MQSeries uses special logging techniques involving *undo/redo* and *compensating log records* to reduce restart times and improve system availability.

One effect of this is that the restart time is bounded. If a failure occurs during a restart so that MQSeries has to be restarted a second time, all the recovery activity that completed to the point of failure in the first restart does not need to be re-applied during a second restart. This means that successive restarts do not take progressively longer times to complete.

Checkpoint records

To reduce restart time, MQSeries takes periodic checkpoints during normal operation:

- When a predefined number of log records has been written.

This number is defined by the checkpoint frequency operand called LOGLOAD of the installation macro CSQ6SYSP described in “Using CSQ6SYSP” on page 68.

- At the end of a successful restart.
- At normal termination.

At the time a checkpoint is taken, MQSeries issues the DISPLAY THREAD command internally so that a list of threads currently in doubt is written to the OS/390 console log.

Page set control records

These records register the page sets known to the MQSeries subsystem at each checkpoint.

How the log is structured

Each active log data set must be a VSAM linear data set (LDS). The physical output unit written to the active log data set is a 4 KB control interval (CI). Each CI contains one VSAM record.

Physical and logical log records

One VSAM CI is a *physical* record. The information to be logged at a particular time forms a *logical* record, whose length varies independently of the space available in the CI. So one physical record can contain:

- Several logical records
- One or more logical records and part of another logical record
- Part of one logical record only

The term “log record” refers to the *logical* record, regardless of how many *physical* records are needed to store it.

How the logs are written

MQSeries writes each log record to a DASD data set called the *active log*. When the active log is full, MQSeries copies its contents to a DASD or tape data set called the *archive log*. This process is called *off-loading*.

Figure 93 illustrates the process of logging. Log records typically go through the following cycle:

1. MQSeries notes changes to data and significant events in recovery log records.
2. MQSeries processes recovery log records and breaks them into segments, if necessary.
3. Log records are placed sequentially in *output log buffers*, which are formatted as VSAM CIs. Each log record is identified by a relative byte address in the range 0 through $2^{48}-1$.
4. The CIs are written to a set of predefined DASD active log data sets, which are used sequentially and recycled.
5. If archiving is active, as each active log data set becomes full, its contents are automatically off-loaded to a new archive log data set.

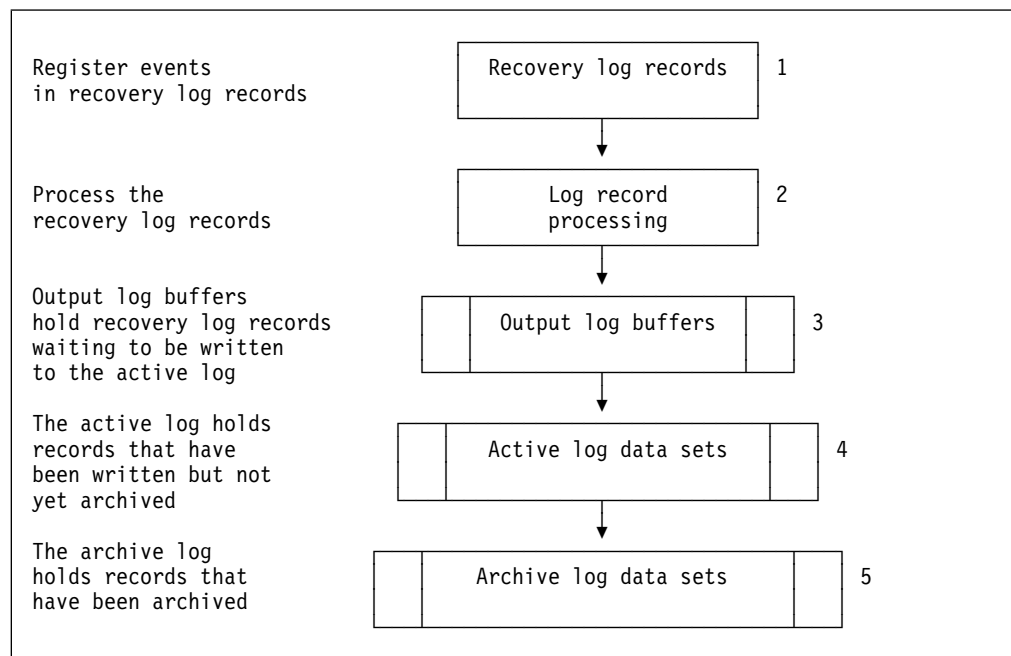


Figure 93. The logging process

When the active log is written

The in-storage log buffers are written to an active log data set whenever any of the following occur:

- The log buffers become full.
- The write threshold is reached (as specified in the CSQ6LOGP macro).
- Certain significant events occur, such as, a commit point.

When MQSeries is initialized, the active log data sets named in the BSDS are dynamically allocated *for exclusive use by MQSeries* and remain allocated *exclusively to MQSeries* until MQSeries terminates. To add or replace active log data sets, you must terminate and restart MQSeries (see “The change log inventory utility (CSQJU003)” on page 256). For details about setting log parameters, see “Using CSQ6LOGP” on page 74.

When the archive log is written

The process of copying active logs to archive logs is called *off-loading*. The relation of off-loading to other logging events is shown schematically in Figure 94.

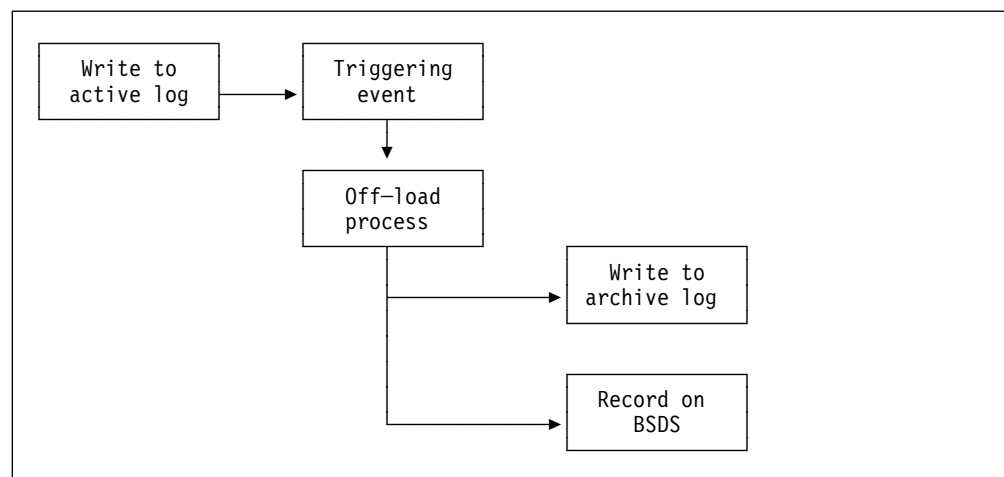


Figure 94. The off-loading process

Triggering an off-load

The off-load of an active log to an archive log can be triggered by several events. For example:

- Filling an active log data set.

Message CSQJ110E is issued when the last available active log is 75% full and at 5% increments thereafter, stating the percentage of the log's capacity in use. If all the active logs become full, MQSeries stops processing until off-loading occurs and issues this message:

```
CSQJ111A +cpf OUT OF SPACE IN ACTIVE LOG DATA SETS
```

- Using the MQSeries command ARCHIVE LOG.
- An error occurring while writing to an active log data set.

The data set is truncated before the point of failure, and the record that failed to be written becomes the first record of the new data set. Off-load is triggered for the

truncated data set as it would be for a normal full log data set. If there are dual active logs, both copies are truncated so that the two copies remain synchronized.

The off-load process

When all the active logs become full, the MQSeries subsystem runs an off-load and halts processing until the off-load has been completed. If the off-load processing fails when the active logs are full, then MQSeries abends. For more information, see “Dealing with active log problems” on page 338.

When an active log is ready to be off-loaded, a request is sent to the OS/390 console operator to mount a tape or prepare a DASD unit. The value of the ARCWTOR option, discussed in “Using CSQ6ARVP” on page 76, determines whether the request is received. If you are using tape for off-loading, specify ARCWTOR=YES. If the value is YES, the request is preceded by a WTOR (message number CSQJ008E) telling the operator to prepare an archive log data set to be allocated.

The operator need not respond to this message immediately. However, delaying the response delays the off-load process. It does not affect MQSeries performance unless the operator delays the response for so long that MQSeries runs out of active logs.

The operator can respond by canceling the off-load. In which case, if the allocation is for the first copy of dual archive data sets, the off-load is merely delayed until the next active log data set becomes full. If the allocation is for the second copy, the archive process switches to single copy mode, but for this data set only.

Archive log data sets

Details of how to plan your archive storage are given in “Planning your archive storage” on page 310.

Day-to-day procedures for managing the archive log are given in Chapter 19, “Managing the logs and the bootstrap data set” on page 315.

Interruptions and errors while off-loading

A +cpf STOP QMGR command does not take effect until off-loading has finished. If MQSeries fails while off-loading is in progress, off-load begins again when MQSeries is restarted. Off-load handling of read I/O errors on the active log is discussed under “Dealing with active log problems” on page 338, and of write I/O errors on the archive log, under “Dealing with archive log problems” on page 343.

Messages during off-load

Off-load messages are sent to the OS/390 console by MQSeries and the off-load process. With the exception of the CSQJ139I message, these messages can be used to find the RBA ranges in the various log data sets. For an explanation of the off-load messages, see the *MQSeries for OS/390 Messages and Codes* manual.

What the bootstrap data set is for

The *bootstrap data set* (BSDS) is a VSAM key-sequenced data set (KSDS) that holds information needed by MQSeries. It contains:

- An inventory of all active and archived log data sets known to MQSeries. MQSeries uses this inventory to:
 - Track the active and archived log data sets
 - Locate log records so that it can satisfy log read requests during normal processing
 - Locate log records so that it can handle restart processing

MQSeries stores information in the inventory each time an archive log data set is defined or an active log data set is reused. For active logs, the inventory shows which are full and which are available for reuse. The inventory holds the relative byte address (RBA) of each log data set. There can be more than one RBA if the log data set spans more than one volume.

- A *wrap-around* inventory of all recent MQSeries activity. This is needed if MQSeries has to be restarted.

The BSDS is required if the subsystem has an error and has to be restarted. MQSeries must have a BSDS; it is not optional. To minimize the likelihood of problems during a restart, MQSeries can be configured with dual BSDSs, each recording the same information. This is known as running in *dual mode*. If possible, the copies should be on separate volumes. This reduces the risk of them both being lost if the volume is corrupted or destroyed. You should use dual BSDSs rather than dual write to DASD.

The BSDS is set up when MQSeries is customized and the inventory can be managed using the change log inventory utility (CSQJU003). For further information, see “Changing the BSDS” on page 321. It is referenced by a DD statement in the MQSeries startup procedure.

Normally, MQSeries keeps duplicate copies of the BSDS. If an I/O error occurs, it deallocates the failing copy and continues with a single BSDS. (For details of how to restore dual mode, see “Dealing with BSDS errors” on page 346.)

The active logs are first registered in the BSDS when MQSeries is installed. They cannot be replaced, nor can new ones be added, without terminating and restarting MQSeries.

Archive log data sets are dynamically allocated. When one is allocated, the data set name is registered in the BSDS. The list of archive log data sets expands as archives are added, and wraps when a user-determined number of entries has been reached. The maximum number of entries is 1000 for single archive logging and 2000 for dual logging.

You can use a tape management system to delete the archive log data sets. MQSeries does not have an automated method. Therefore, the information about an archive log data set can be in the BSDS long after the archive log data set has been deleted by the system administrator.

Conversely, the maximum number of archive log data sets could have been exceeded, and the data from the BSDS dropped long before the data set has

reached its expiry date. For additional information, see “Automatic archive log data set deletion” on page 317.

If CSQZPARM specifies that archive log data sets are cataloged when allocated, the BSDS points to the interactive catalog facility (ICF) catalog for the information needed for later allocations. Otherwise, the BSDS entries for each volume register the volume serial number and unit information that is needed for later allocations.

Archive log data sets and BSDS copies

Each time a new archive log data set is created, a copy of the BSDS is also created. If the archive log is on tape, the BSDS is the first data set on the first output volume. If the archive log is on DASD, the BSDS is a separate data set.

The data set names of the archive log and the BSDS copy are the same, except that the lowest-level qualifier of the archive log name begins with A and the BSDS copy begins with B, for example:

Archive log name	CSQ.ARCHLOG1.D93022.T2336229.A0000001
BSDS copy name	CSQ.ARCHLOG1.D93022.T2336229.B0000001

If there is a read error while copying the BSDS, the copy is not created, message CSQJ125E is issued, and the off-load to the new archive log data set continues without the BSDS copy.

Managing the logs and BSDS

Day-to-day procedures for managing the BSDS are described in Chapter 19, “Managing the logs and the bootstrap data set” on page 315.

In these procedures and the other specialized procedures that are described, you might need to copy log or BSDS data sets. Because they are VSAM data sets, use the Access Method Services REPRO function (or any equivalent) to do this.

For more information on the REPRO statement, see the *DFSMS/MVS Access Method Services for VSAM* or the *DFSMS/MVS Access Method Services for the Integrated Catalog Facility* manuals.

Chapter 18. Planning for backup and recovery

This chapter describes what you can do now to minimize problems following any future failure. Chapter 21, “Example recovery scenarios” on page 337 describes the procedures you should follow when particular failures occur.

This chapter contains these sections:

- “Introduction”
- “General tips for backup and recovery”
- “Planning your logging environment” on page 309
- “Planning your archive storage” on page 310
- “Other recovery considerations” on page 311

Introduction

Developing backup and recovery procedures at your site is vital to avoid costly and time-consuming losses of data. MQSeries provides means for recovering both queues and messages to their current state after a system failure. You should develop the following procedures for MQSeries:

- Creating a point of recovery
- Backing up page sets
- Recovering page sets
- Recovering from out-of-space conditions (MQSeries logs and page sets)

See Chapter 20, “Managing page sets” on page 325 for information about procedures for page sets and creating a point of recovery, and “Dealing with active log problems” on page 338 for information about dealing with problems with the log.

You should also be familiar with the procedures used at your site for the following:

- Recovering from a hardware or power failure
- Recovering from an OS/390 component failure
- Recovering from a site interruption, using off-site recovery

General tips for backup and recovery

The MQSeries restart process recovers your data to a consistent state by applying log information to the page sets. If your page sets are damaged or unavailable, you can resolve the problem using your backup copies of your page sets (provided that all the logs are available). If your log data sets are damaged or unavailable, it might not be possible to recover completely. This section introduces some backup and recovery tasks.

Periodically take backup copies

A *point of recovery* is the term used to describe a set of backup copies of MQSeries page sets and the corresponding log data sets required to recover these page sets. These backup copies provide a potential restart point in the event of page set loss (for example, page set I/O error). If MQSeries were to be restarted using these backup copies, the data in MQSeries will be consistent up to the point that these copies were taken. Providing that all logs are available from this point, MQSeries can be recovered to the point of failure. See “Creating a point of recovery” on page 331 for more information about points of recovery.

The more recent your backup copies, the quicker MQSeries can recover the data in the page sets. The recovery of the page sets are dependent on all the necessary log data sets being available.

In planning for recovery, you need to determine how often to take backup copies and how many complete backup cycles to keep. These values tell you how long you must keep your log data sets and backup copies of page sets for MQSeries recovery.

In deciding how often to take backup copies, consider the time needed to recover a page set. It is determined by:

- The amount of log to traverse
- The time it takes an operator to mount and remove archive tape volumes
- The time it takes to read the part of the log needed for recovery
- The time needed to reprocess changed pages
- The storage medium used for the backup copies
- The method used to make and restore backup copies

In general, the more often you make backup copies, the less time recovery takes, but, the more time is spent making copies.

For each queue manager, you should take backup copies of:

- The archive log data sets
- The BSDS copies created at the time of the archive (see “Archive log data sets and BSDS copies” on page 306)
- The page sets
- Your object definitions

To reduce the risk of your backup copies being lost or damaged, you should consider:

- Storing the backup copies on different storage volumes to the original copies.
- Storing the backup copies at a different site to the original copies.
- Making at least two copies of each backup of your page sets and, if you are using single logging or a single BSDS, two copies of your archive logs and BSDS. If you are using dual logging or BSDS, a single copy of both archive logs or BSDS will suffice.

Before moving MQSeries to a production environment you should have tested and documented your backup procedures.

Backing up your object definitions

You should also back up copies of your object definitions. To do this, use the MAKEDEF feature of the CSQUTIL COMMAND function (described in “MQSeries command management functions” on page 238).

You should do this whenever you take backup copies of your queue manager data sets, and keep the most current version.

Do not discard archive logs you might need

MQSeries might need to use archive logs during restart. You must keep sufficient archive logs so the system can be fully restored. MQSeries might use an archive log to recover a page set from a restored backup copy. If you have discarded that archive log, MQSeries is not able to restore the page set to its current state. To find out when and how you should discard archive logs, see “Discarding archive log data sets” on page 317.

Do not change the DDname-to-page set association

MQSeries associates page set number 00 with DDname CSQP0000, page set number 01 with DDname CSQP0001, and so on up to CSQP0099. MQSeries writes recovery log records for a page set based on the DDname that the page set is associated with. For this reason, you must not move page sets that have already been associated with a PSID DDname.

Planning your logging environment

The MQSeries logging environment is established using the installation macros to specify options, such as whether to have single or dual active logs, what media to use for the archive log volumes, and how many log buffers to have. For details, see “Task 13: Tailor your system parameter module” on page 67.

Use dual logging for your active log, archive log, and bootstrap data sets

Dual logging minimizes the risk associated with DASD failure.

Important

In a production MQSeries subsystem, it is important to establish both of the following:

1. Archiving
2. Dual logging for:
 - Active logs
 - Archive logs
 - Bootstrap data sets

This minimizes the risk of losing your data, for example, because of DASD failures.

Planning your archive storage

This section describes the different ways of maintaining your archive log data sets.

Archive log data sets can be placed on standard-label tapes, or DASD, and can be managed by data facility hierarchical storage manager (DFHSM). Each OS/390 logical record in an archive log data set is a VSAM control interval from the active log data set. The block size is a multiple of 4 KB. For more information, see “Using CSQ6LOGP” on page 74.

Archive log data sets are dynamically allocated, with names chosen by MQSeries. The data set name prefix, block size, unit name, and DASD sizes needed for such allocations are specified in the CSQZPARM module. You can also choose, at installation time, to have MQSeries add a date and time to the archive log data set name. For more information, see “Using CSQ6ARVP” on page 76.

It is not possible to choose specific volumes for new archive logs. If allocation errors occur, off-loading is postponed until the next time off-loading is triggered.

If you specify dual archive logs at installation time, each log control interval retrieved from the active log is written to two archive log data sets. The log records that are contained in the pair of archive log data sets are identical, but the end-of-volume points are not synchronized for multi-volume data sets.

Archiving to tape

If the unit name reflects a tape device, MQSeries can extend to a maximum of twenty volumes.

If you choose to off-load to tape, you should consider adjusting the size of your active log data sets so that each nearly fills a tape volume. This minimizes tape handling and volume mounts, and maximizes the use of tape resources. However, such an adjustment is not essential.

If you are considering changing the size of the active log data set so that the set fits on one tape volume, you must bear in mind that a copy of the BSDS is placed on the same tape volume as the copy of the active log data set. Adjust the size of the active log data set downward to offset the space required for the BSDS on the tape volume.

Archiving to DASD volumes

MQSeries requires that all archive log data sets allocated on non-tape devices (DASD) be cataloged. If you choose to archive to DASD, the CATALOG DATA parameter of the CSQ6ARVP macro must be YES. (For details, see “Using CSQ6ARVP” on page 76.) If this parameter is NO, and you decide to place archive log data sets on DASD, you receive message CSQJ072E each time an archive log data set is allocated, although the MQSeries subsystem still catalogs the data set.

If the unit name shows that the archive log data set is held on DASD, the archive log data sets cannot extend to another volume.

If you choose to use DASD, make sure that the primary space allocation (both quantity and block size) is large enough to contain all of the data coming from the

active log data set, plus that from the corresponding BSDS. This minimizes the possibility of unwanted OS/390 B37 or E37 abends during the off-load process. The primary space allocation is set with the PRIQTY (primary quantity) parameter of the CSQ6ARVP macro discussed in “Using CSQ6ARVP” on page 76.

Using SMS with archive log data sets

If you have MVS/DFP storage management subsystem (DFSMS™) installed, you can write an assembly control system (ACS) user-exit filter for your archive log data sets, which helps you convert them for the SMS environment. Such a filter, for example, can route your output to a DASD data set, which in turn can be managed by DFSMS. You must exercise caution if you use an ACS filter in this manner. Because SMS requires DASD data sets to be cataloged, you must make sure the CATALOG DATA field of the CSQ6ARVP macro contains YES. If it does not, message CSQJ072E is returned; however, the data set is still cataloged by MQSeries. For details about this macro, see “Using CSQ6ARVP” on page 76.

For more information about ACS filters, see the *DFP Storage Administration Reference* manual, and the *SMS Migration Planning Guide*.

Other recovery considerations

This section describes some other things that you should take into account when you are planning your backup and recovery procedures.

Backup and recovery with DFHSM

The data facility hierarchical storage manager (DFHSM) does automatic space- and data-availability management among storage devices in your system. If you use it, you need to know that it moves data to and from the MQSeries storage automatically.

DFHSM manages your DASD space efficiently by moving data sets that have not been used recently to alternate storage. It also makes your data available for recovery by automatically copying new or changed data sets to tape or DASD backup volumes. It can delete data sets, or move them to another device. Its operations occur daily, at a specified time, and allow for keeping a data set for a predetermined period before deleting or moving it.

All DFHSM operations can also be performed manually. The *Data Facility Hierarchical Storage Manager User's Guide* explains how to use the DFHSM commands.

If you use DFHSM with MQSeries, note that DFHSM:

- Uses cataloged data sets
- Operates on page sets and logs
- Supports VSAM data sets

MQSeries recovery and CICS

The recovery of CICS resources is not affected by the presence of MQSeries. CICS recognizes MQSeries as a non-CICS resource (or external resource manager), and includes MQSeries as a participant in any syncpoint coordination requests using the CICS resource manager interface (RMI). For more information about CICS recovery, see the *CICS Recovery and Restart Guide*. For information about the CICS resource manager interface, see the *CICS Customization Guide*.

MQSeries recovery and IMS

IMS recognizes MQSeries as an external subsystem and as a participant in syncpoint coordination. IMS recovery for external subsystem resources is described in the *IMS Customization Guide*.

Using Extended Recovery Facility

MQSeries can be used in an extended recovery facility (XRF) environment. All MQSeries-owned data sets (executable code, BSDSs, logs, and page sets) must be on DASD shared between the active and alternate XRF processors. If you use XRF for recovery, you must stop MQSeries on the active processor and start it on the alternate. For CICS, this can be done using the command list table (CLT) provided by CICS, or manually by the system operator. For IMS, this is a manual operation and must be done after the coordinating IMS system has completed the processor switch. MQSeries utilities must be completed or terminated before MQSeries can be switched to the alternate processor. Consider the effect of this potential interruption carefully when planning your XRF recovery plans.

Take care to prevent MQSeries starting on the alternate processor before the MQSeries system on the active processor terminates. A premature start can cause severe integrity problems in data, the catalog, and the log. Using global resource serialization (GRS) helps avoid the integrity problems by preventing simultaneous use of MQSeries on the two systems. The BSDS must be included as a protected resource, and the active and alternate XRF processors must be included in the GRS ring.

Preparing for disaster recovery

In the case of a total loss of an MQSeries computing center, you can recover on another MQSeries system at a recovery site. To be able to do this, you must regularly back up the page sets and the logs. As with all data recovery operations, the objectives of disaster recovery are to lose as little data, workload processing (updates), and time as possible.

At the recovery site:

- The recovery MQSeries queue manager **must** have the same name as the lost queue manager.
- The system parameter module (CSQZPARM) used on the recovery queue manager should contain the same parameters as the lost queue manager.

The following process can be used to perform disaster recovery at the recovery site. It assumes that all that is available are:

- Copies of the archive logs and BSDSs created by normal running at the primary site (the active logs will have been lost along with the queue manager at the primary site).

- Copies of the page sets from the queue manager at the primary site that are the same age or older than the most recent archive log copies available.

If required, dual active and archive logs should be considered, and the BSDS updates applied to both copies:

1. Define new page set data sets and load them with the data in the copies of the page sets from the primary site.
2. Define new active log data sets.
3. Define a new BSDS data set and use Access Method Services REPRO to copy the **most recent** archived BSDS into it.
4. Use the print log map utility CSQJU004 to print information from this most recent BSDS. At the time this BSDS was archived, the most recent archived log you have would have just been truncated as an active log, and will not appear as an archived log. Record the STARTRBA and ENDRBA of this log.
5. Use Access Method Services REPRO to copy the most recent archived log into one of the active logs.
6. Use the change log inventory utility CSQJU003 to remove all active log information from the BSDS.
7. Use CSQJU003 to add active logs to the BSDS, including the RBA range of the logs used in Step 5 as found in Step 4.
8. Use CSQJU003 to add a restart control record to the BSDS. Specify:

```
CRESTART CREATE,ENDRBA=highrba
```

Where `highrba` is the high RBA of the most recent archive log available (found in Step 4), plus 1.

The BSDS now describes one active log with an RBA range, all other active logs as being empty, all the archived logs you have available, and no checkpoints beyond the end of your logs.

9. Restart MQSeries with the usual START QMGR command. During initialization, an operator reply message such as the following will be issued:

```
CSQJ245D +cpf RESTART CONTROL INDICATES TRUNCATION AT RBA highrba.  

      REPLY Y TO CONTINUE, N TO CANCEL
```

Reply Y to start MQSeries. MQSeries will start, and will recover data up to ENDRBA specified in the CRESTART statement.

See “The change log inventory utility (CSQJU003)” on page 256 for information about using CSQJU003 and “The print log map utility (CSQJU004)” on page 265 for information about using CSQJU004.

Figure 95 on page 314 shows sample input statements for CSQJU003 for steps 6, 7, and 8.

The things you need to consider for restarting the channel initiator at the recovery site are similar to those faced when using ARM to restart the channel initiator on a different OS/390 image. See “Restarting on a different OS/390 image” on page 297 for more information.

Your recovery strategy should also cover recovery of the MQSeries product libraries and the application programming environments that use MQSeries (CICS, for example).

Recovery considerations

```
* Step 6
  DELETE DSNAME=MQM2.LOGCOPY1.DS01
  DELETE DSNAME=MQM2.LOGCOPY1.DS02
  DELETE DSNAME=MQM2.LOGCOPY1.DS03
  DELETE DSNAME=MQM2.LOGCOPY2.DS01
  DELETE DSNAME=MQM2.LOGCOPY2.DS02
  DELETE DSNAME=MQM2.LOGCOPY2.DS03

* Step 7
  NEWLOG DSNAME=MQM2.LOGCOPY1.DS01,COPY1
    STARTRBA=05C000,ENDRBA=000000062FFF
  NEWLOG DSNAME=MQM2.LOGCOPY1.DS02,COPY1
  NEWLOG DSNAME=MQM2.LOGCOPY1.DS03,COPY1
  NEWLOG DSNAME=MQM2.LOGCOPY2.DS01,COPY2
    STARTRBA=05C000,ENDRBA=000000062FFF
  NEWLOG DSNAME=MQM2.LOGCOPY2.DS02,COPY2
  NEWLOG DSNAME=MQM2.LOGCOPY2.DS03,COPY2

* Step 8
  CRESTART CREATE,ENDRBA=063000
```

Figure 95. Sample input statements for CSQJU003

Other functions of the change log inventory utility (CSQJU003) can also be used in disaster recovery scenarios. The HIGHRBA function allows the update of the highest RBA written and highest RBA offloaded values within the bootstrap data set. The CHECKPT function allows the addition of new checkpoint queue records or the deletion of existing checkpoint queue records in the BSDS. These functions might affect the integrity of the MQSeries system and should only be used in disaster recovery scenarios under the guidance of IBM service personnel.

Chapter 19. Managing the logs and the bootstrap data set

This chapter describes the tasks involved in managing the logs and the bootstrap data set. It contains these sections:

- “Archiving logs with the ARCHIVE LOG command”
- “Discarding archive log data sets” on page 317
- “Printing log records” on page 319
- “Finding out what the BSDS contains” on page 319
- “Changing the BSDS” on page 321
- “Recovering logs” on page 323

Archiving logs with the ARCHIVE LOG command

An authorized operator can archive the current MQSeries active log data sets whenever required using the ARCHIVE LOG command.

When you issue the ARCHIVE LOG command, MQSeries truncates the current active log data sets, then runs an asynchronous off-load, and updates the BSDS with a record of the off-load.

The ARCHIVE LOG command has a MODE(QUIESCE) option. With this option, MQSeries users are quiesced after a commit point, and the resulting point of consistency is captured in the current active log before it is off-loaded.

Consider using the MODE(QUIESCE) option when planning a backup strategy for off site recovery. It creates a system-wide point of consistency, which minimizes the number of data inconsistencies when the archive log is used with the most current backup page set copy during recovery. For example:

```
ARCHIVE LOG MODE(QUIESCE)
```

If the ARCHIVE LOG command is issued without specifying a TIME parameter, the quiesce time period defaults to the value of the QUIESCE parameter of the CSQ6ARVP macro. If the time required for the ARCHIVE LOG MODE(QUIESCE) to complete is less than the time specified, the command completes successfully; otherwise, the command fails when the time period expires. You can specify the time period explicitly by using the TIME option, for example:

```
ARCHIVE LOG MODE(QUIESCE) TIME(60)
```

This command specifies a quiesce period of up to 60 seconds before ARCHIVE LOG processing occurs.

Attention: Using this option when time is critical can cause a significant disruption in MQSeries availability for all jobs and users that use MQSeries resources.

By default, the command is processed asynchronously from the time you submit the command. (To process the command synchronously with other MQSeries commands use the WAIT(YES) option QUIESCE, but be aware that the OS/390 console is locked from MQSeries command input for the entire QUIESCE period.)

During the quiesce period:

- Jobs and users on MQSeries are allowed to go through commit processing, but are suspended if they try to update any MQSeries resource after the commit.
- Jobs and users that only read data can be affected, since they can be waiting for locks held by jobs or users that were suspended.
- New tasks can start, but they are not allowed to update data.

The DISPLAY THREAD output uses the message CSQV400I to indicate that a quiesce is in effect. For example:

```
CSQV401I +cpf DISPLAY THREAD REPORT FOLLOWS -
CSQV400I +cpf ARCHIVE LOG QUIESCE CURRENTLY ACTIVE
CSQV402I +cpf ACTIVE THREADS -
NAME      ST A  REQ THREAD-XREF  USERID      ASID  URID
BATCH    T   14                CON0327     0016  000000000000
DISPLAY ACTIVE REPORT COMPLETE
CSQ9022I +cpf CSQVDT ' DISPLAY THREAD' NORMAL COMPLETION
```

When all updates are quiesced, the quiesce history record in the BSDS is updated with the date and time that the active log data sets were truncated, and with the last-written RBA in the current active log data sets. MQSeries truncates the current active log data sets, switches to the next available active log data sets, and issues message CSQJ311E stating that off-load started.

If updates cannot be quiesced before the quiesce period expires, MQSeries issues message CSQJ317I, and ARCHIVE LOG processing terminates. The current active log data sets are not truncated and not switched to the next available log data sets, and off-load is not started.

Whether the quiesce was successful or not, all suspended users and jobs are then resumed, and MQSeries issues message CSQJ312I, stating that the quiesce is ended and update activity is resumed.

If ARCHIVE LOG is issued when the current active log is the last available active log data set, the command is not processed, and MQSeries issues this message:

```
CSQJ319I - csect-name CURRENT ACTIVE LOG DATA SET IS THE LAST
          AVAILABLE ACTIVE LOG DATA SET.  ARCHIVE LOG PROCESSING
          WILL BE TERMINATED.
```

If ARCHIVE LOG is issued when another ARCHIVE LOG command is already in progress, the new command is not processed, and MQSeries issues this message:

```
CSQJ318I - ARCHIVE LOG COMMAND ALREADY IN PROGRESS.
```

For information about the syntax of the ARCHIVE LOG command, see the *MQSeries Command Reference* manual. For information about the messages issued during archiving, see the *MQSeries for OS/390 Messages and Codes* manual.

Discarding archive log data sets

You must keep enough log records to recover units of recovery or perform media recovery if a page set is lost. Do not discard archive log data sets that might be required for recovery; if you discard these archive log data sets you might not be able to recover using your page set backups.

However, if you have confirmed that your archive log data sets can be discarded, you can do this in either of the following ways:

- Automatic archive deletion (see “Automatic archive log data set deletion”)
- Manual archive deletion (see “Manually deleting archive log data sets” on page 318)

Automatic archive log data set deletion

You can use a DASD or tape management system to delete archive log data sets automatically. The retention period for MQSeries archive log data sets is specified by the retention period field ARCRETN in the CSQ6ARVP installation macro. See “Using CSQ6ARVP” on page 76 for more information. This value is passed to the management system in the JCL parameter RETPD.

The default for the retention period specifies that archive logs are to be kept for 9999 days (the maximum possible). **You can change the retention period but you must ensure that you can accommodate the number of backup cycles that you have planned for.**

MQSeries uses the value as the value for the JCL parameter RETPD when archive log data sets are created.

The retention period set by MVS/DFP’s storage management subsystem (SMS) can be overridden by this MQSeries parameter. Typically, the retention period is set to the smaller value specified by either MQSeries or SMS. The storage administrator and MQSeries administrator must agree on a retention period value that is appropriate for MQSeries.

Note: Because some tape management systems provide external manual overrides of retention periods, MQSeries does not have an automated method to delete information about archive log data sets from the BSDS. Therefore, information about an archive log data set can still be in the BSDS long after the data-set retention period has expired and the data set has been scratched by the tape management system. Conversely, the maximum number of archive log data sets might have been exceeded and the data from the BSDS might have been dropped before the data set has reached its expiration date.

If archive log data sets are deleted automatically, remember that the operation does not update the list of archive logs in the BSDS. You can update the BSDS with the change log inventory utility, as described in “Changing the BSDS” on page 321. The update is not essential. Recording old archive logs wastes space in the BSDS, but does no other harm.

Manually deleting archive log data sets

You must keep all the log records as far back as the lowest RBA identified in messages CSQI024I and CSQI025I. This RBA is obtained using the DISPLAY USAGE command as issued when creating a point of recovery using “Method 1: Full backup” on page 331. **You should read “Creating a point of recovery” on page 331 before discarding any logs.**

Locate and discard archive log data sets

Having established the minimum log RBA required for recovery from your page set backup cycles, you can find archive log data sets that contain only earlier log records by performing the following procedure:

1. Use the print log map utility to print the contents of the BSDS. For an example of the output, see “The print log map utility (CSQJU004)” on page 265.
2. Find the sections of the output titled “ARCHIVE LOG COPY n DATA SETS”. If you use dual logging, there are two sections. The columns labeled STARTRBA and ENDRBA show the range of RBAs contained in each volume. Find the volumes whose ranges include the minimum RBA you found with messages CSQI024I and CSQI025I. These are the earliest volumes you need to keep. If you are using dual-logging, there are two such volumes.

If no volumes have an appropriate range, one of these cases applies:

- The minimum RBA has not yet been archived, and you can discard all archive log volumes.
- The list of archive log volumes in the BSDS wrapped around when the number of volumes exceeded the number allowed by the MAXARCH parameter of the CSQ6LOGP macro. If the BSDS does not register an archive log volume, that volume cannot be used for recovery. Therefore, you should consider adding information about existing volumes to the BSDS. For instructions, see “Changes for archive logs” on page 322.

You should also consider increasing the value of MAXARCH. For information, see “Using CSQ6LOGP” on page 74.

3. Delete any archive log data set or volume whose ENDRBA value is less than the STARTRBA value of the earliest volume you want to keep. If you are using dual logging, delete both such copies.

Because BSDS entries wrap around, the first few entries in the BSDS archive log section might be more recent than the entries at the bottom. Look at the combination of date and time and compare their ages. Do not assume that you can discard all entries *above* the entry for the archive log containing the minimum LOGRBA.

Delete the data sets. If the archives are on tape, erase the tapes. If they are on DASD, run an OS/390 utility to delete each data set. Then, if you want the BSDS to list only existing archive volumes, use the change log inventory utility (CSQJU003) to delete entries for the discarded volumes. See “Changes for archive logs” on page 322 for an example.

Printing log records

You can extract and print log records using the CSQ1LOGP utility. For instructions, see “The log print utility (CSQ1LOGP)” on page 266.

Finding out what the BSDS contains

The print log map utility (CSQJU004) is a batch utility that lists the information stored in the BSDS. For instructions on running it, see “The print log map utility (CSQJU004)” on page 265.

Time stamps in the BSDS

The output of the print log map utility shows the time stamps, which are used to record the date and time of various system events, that are stored in the BSDS.

The following time stamps are included in the header section of the report:

SYSTEM TIMESTAMP

Reflects the date and time the BSDS was last updated. The BSDS time stamp can be updated when:

- MQSeries starts.
- The write threshold is reached during log write activities. Depending on the number of output buffers you have specified and the system activity rate, the BSDS can be updated several times a second, or could not be updated for several seconds, minutes, or even hours. For details of the write threshold, see the WRTHRSH parameter of the CSQ6LOGP macro in the section “Using CSQ6LOGP” on page 74.
- MQSeries drops into a single BSDS mode from its normal dual BSDS mode due to an error. This can occur when a request to get, insert, point to, update, or delete a BSDS record is unsuccessful. When this error occurs, MQSeries updates the time stamp in the remaining BSDS to force a time stamp mismatch with the disabled BSDS.

UTILITY TIMESTAMP

The date and time the contents of the BSDS were altered by the change log inventory utility (CSQJU003).

The following time stamps are included in the active and archive log data sets portion of the report:

Active log date

The date the active log entry was created in the BSDS, that is, when the CSQJU003 NEWLOG was done.

Active log time

The time the active log entry was created in the BSDS, that is, when the CSQJU003 NEWLOG was done.

Archive log date

The date the archive log entry was created in the BSDS, that is, when the CSQJU003 NEWLOG was done or the archive itself was done.

Finding what the BSDS contains

Archive log time

The time the archive log entry was created in the BSDS, that is, when the CSQJU003 NEWLOG was done or the archive itself was done.

Active log data set status

The BSDS records the status of an active log data set as one of the following:

- NEW** The data set has been defined but never used by MQSeries, or the log was truncated to a point before the data set was first used. In either case, the data set starting and ending RBA values are reset to zero.
- REUSABLE** Either the data set has been defined but never used by MQSeries, or the data set has been off-loaded. In the print log map output, the start RBA value for the last REUSABLE data set is equal to the start RBA value of the last archive log data set.
- NOT REUSABLE** The data set contains records that have not been off-loaded.
- STOPPED** The off-load processor encountered an error while reading a record, and that record could not be obtained from the other copy of the active log.
- TRUNCATED** Either:
- An I/O error occurred, and MQSeries has stopped writing to this data set. The active log data set is off-loaded, beginning with the starting RBA and continuing up to the last valid record segment in the truncated active log data set. The RBA of the last valid record segment is lower than the ending RBA of the active log data set. Logging is switched to the next available active log data set, and continues uninterrupted.
- or
- An ARCHIVE LOG function has been called, which has truncated the active log.

The status appears in the output from the print log map utility.

Changing the BSDS

You do not have to take special steps to keep the BSDS updated with records of logging events because MQSeries does that automatically. However, you might want to change the BSDS if you do any of the following:

- Add more active log data sets.
- Copy active log data sets to newly allocated data sets, for example, when providing larger active log allocations.
- Move log data sets to other devices.
- Recover a damaged BSDS.
- Discard outdated archive log data sets.

You can change the BSDS by running the change log inventory utility (CSQJU003). This utility can be run whether MQSeries is active or inactive. However, you are recommended **not** to run it when MQSeries is active, or you might get inconsistent results. The action of the utility is controlled by statements in the SYSIN data set. This section shows several examples. For complete instructions, see “The change log inventory utility (CSQJU003)” on page 256.

You can copy an active log data set only when MQSeries is inactive because MQSeries allocates the active log data sets as exclusive (DISP=OLD) at MQSeries startup.

Changes for active logs

You can add to, delete from, and record entries in the BSDS for active logs using the change log utility. Examples only are shown here; replace the data set names shown with the ones you want to use. For more details of the utility, see “The change log inventory utility (CSQJU003)” on page 256.

Adding record entries to the BSDS: If an active log has been flagged as “stopped”, it is not reused for logging; however, it continues to be used for reading. Use the access method services to define new active log data sets, then use the change log inventory utility to register the new data sets in the BSDS. For example, use:

```
NEWLOG DSNAME=MQM111.LOGCOPY1.DS10,COPY1
NEWLOG DSNAME=MQM111.LOGCOPY2.DS10,COPY2
```

If you are copying the contents of an old active log data set to the new one, you can also give the RBA range and the starting and ending time stamps on the NEWLOG function.

Deleting information about the active log data set from the BSDS: To delete information about an active log data set from the BSDS, you could use:

```
DELETE DSNNAME=MQM111.LOGCOPY1.DS99
DELETE DSNNAME=MQM111.LOGCOPY2.DS99
```

Recording information about the log data set in the BSDS: To record information about an existing active log data set in the BSDS, use:

```
NEWLOG DSNNAME=MQM111.LOGCOPY1.DS10,COPY2,STARTIME=19930212205198,
        ENDTIME=19930412205200,STARTRBA=6400,ENDRBA=94FF
```

Inserting a record containing this type of information in the BSDS might be necessary because:

- The entry for the data set has been deleted, but is needed again.
- You are copying the contents of one active log data set to another data set.
- You are recovering the BSDS from a backup copy.

Enlarging the active log: This procedure must only be used when MQSeries is inactive:

1. Stop MQSeries. This step is required because MQSeries allocates all active log data sets for its exclusive use when it is active.
2. Use Access Method Services ALTER with the NEWNAME option to rename your active log data sets.
3. Use Access Method Services DEFINE to define larger active log data sets.

By reusing the old data set names, you do not have to run the change log inventory utility to establish new names in the BSDSs. The old data set names and the correct RBA ranges are already in the BSDSs.

4. Use Access Method Services REPRO to copy the old (renamed) data sets into their respective new data sets.
5. Start MQSeries.

Although it is not necessary for all log data sets to be the same size, it is operationally more consistent and efficient. If the log data sets are not the same size, it is more difficult to track your system's logs, and so space can be wasted.

Changes for archive logs

You can add to, delete from, and change the password of entries in the BSDS for archive logs. Examples only are shown here; you must replace the data set names shown with the ones you want to use. For more details of the utility, see "The change log inventory utility (CSQJU003)" on page 256.

Adding an archive log: When the recovery of an object depends on reading an existing archive log data set, the BSDS must contain information about that data set, so that MQSeries can find it. To register information about an existing archive log data set in the BSDS, use:

```
NEWLOG DSNAME=CSQARC1.ARCHLOG1.D92021.T2205197.A0000015,COPY1VOL=CSQV04,
UNIT=TAPE,STARTRBA=3A190000,ENDRBA=3A1F0FFF,CATALOG=NO
```

Deleting an archive log: To delete an entire archive log data set on one or more volumes, use:

```
DELETE DSNAME=CSQARC1.ARCHLOG1.D92021.T2205197.A0000015,COPY1VOL=CSQV04
```

Changing the password of an archive log: If you change the password of an existing archive log data set, you must also change the information in the BSDS.

1. List the BSDS, using the print log map utility.
2. Delete the entry for the archive log data set with the changed password, using the DELETE function of the CSQJU003 utility (see page 256).
3. Name the same data set as a new archive log data set. Use the NEWLOG function of the CSQJU003 utility (see page 256), and give the new password, the starting and ending RBAs, and the volume serial numbers (which can be found in the print log map utility output, see page 265).

To change the password for new archive log data sets, use:

```
ARCHIVE PASSWORD=password
```

To stop placing passwords on new archive log data sets, use:

```
ARCHIVE NOPASSWD
```

Note: You should only use the ARCHIVE utility function if you do not have an external security manager.

Recovering logs

Normally, you do not need to back up and restore the MQSeries logs, especially if you are using dual logging. However, in rare circumstances, such as an I/O error on a log, you might need to recover the logs. Use Access Method Services to delete and redefine the data set, and then copy the corresponding dual log into it.

Recovering logs

Chapter 20. Managing page sets

This chapter describes how to create, copy, and generally manage the page sets associated with a queue manager. It contains these sections:

- “Adding a page set to a queue manager”
- “When one of your page sets becomes full”
- “How to balance loads on page sets” on page 327
- “How to reduce a page set” on page 330
- “Backing up and recovering page sets” on page 331
- “Backing up and restoring queues using CSQUTIL” on page 335

See “An overview of storage management” on page 15 for a description of page sets, storage classes, buffers, and buffer pools, and some of the performance considerations that apply.

Adding a page set to a queue manager

This description assumes that you have an MQSeries subsystem that is already running. You might need to add a page set if, for example, your MQSeries subsystem has to cope with new applications using new queues.

To add a new page set, use the following procedure:

1. Stop the queue manager by issuing a STOP QMGR command.
2. Define and format the new page set. You can use the sample JCL in thlqual.SCSQPROC(CSQ4PAGE) as a basis. For more information, see “Formatting page sets (FORMAT)” on page 231.

Take care not to format any page sets that are in use, unless this is what you intend. If so, use the FORCE option of the FORMAT utility function.
3. Add the new page set to the startup procedure for your MQSeries subsystem.
4. Add a definition for the new page set to your CSQINP1 initialization data set. Use the DEFINE PSID command to associate the page set with a buffer pool.
5. Add the appropriate storage class definitions for your page set to your CSQINP2 initialization data set concatenation. This step is optional but recommended, see “Task 10: Customize the initialization input data sets” on page 52.
6. Restart the queue manager.

When one of your page sets becomes full

You can find out about the utilization of page sets by using the MQSeries command DISPLAY USAGE. For example, the command:

```
DISPLAY USAGE PSID(03)
```

displays the current state of the page set 03. This tells you how many free pages this page set has.

If you have defined secondary extents for your page sets, they will be dynamically expanded each time they fill up. Eventually, all secondary extents will be used, or no further disk space is available, in which case, and application will receive a return code MQRC_PAGESET_FULL.

If an application receives a return code of MQRC_PAGE_SET_FULL from an MQI call, this is a clear indication that this page set is over-utilized, that is, there is not enough space left on the page set. If the problem persists or is likely to reoccur, you must do something to solve it.

You can approach this problem in two ways:

1. Expand the page set.
2. Balance the load between page sets by moving queues from one page set to another.

How to expand a page set

You expand a page set by creating a new, larger page set and copying the messages from the old page set to the new one. You then have to ensure that the new page set is used when you restart the queue manager.

Note

This technique involves stopping and restarting the queue manager. This will result in any nonpersistent messages being deleted at restart time. If you have nonpersistent messages that you do not want to be deleted, consider load balancing (see “How to balance loads on page sets” on page 327).

Note: In this description, the page set that you want to expand is referred to as the *source* page set; the new, larger page set is referred to as the *destination* page set.

Follow these steps:

1. Stop the queue manager.
2. Define the destination page set, ensuring that it is larger than the source page set, with a larger secondary extent value.
3. Use the FORMAT function of CSQUTIL to format the destination page set. See “Formatting page sets (FORMAT)” on page 231 for more details.
4. Use the COPYPAGE function of CSQUTIL to copy all the messages from the source page set to the destination page set. See “Expanding a page set (COPYPAGE)” on page 233 for more details.
5. Restart the queue manager using the destination page set by doing one of the following:
 - Change the MQSeries startup procedure to reference the destination page set. See “Task 12: Define your page sets” on page 62 for more details.
 - Use Access Method Services to delete the source page set and then rename the destination page set, giving it the same name as that of the source page set.

Attention: Before you delete any MQSeries page set, be sure that you have made the required backup copies.

How to balance loads on page sets

Load balancing on page sets means moving the messages associated with one or more queues from one page set to another, less utilized page set.

Note

You should use this technique if it is not practical to expand the page set. It is possible that messages can be lost when uncommitted messages on the queue are committed or backed out during the load balancing operation.

Step 5 on page 328 uses the COPY function of CSQUTIL to off-load the messages to a data set. This only copies committed messages. Step 7 on page 328 deletes the queue that you are moving with the PURGE option, and only succeeds if there are no uncommitted messages on the queue. In the time between execution of these steps:

- **MQPUT** calls that are committed in this window are deleted with the queue
- **MQGET** calls that are backed out in this window are deleted with the queue

When identifying which queues are most suitable for moving you should therefore consider the following points:

- Do applications issue **MQPUT** or **MQGET** calls within syncpoint against the queue?
If not, there is no potential for losing messages between the execution of steps 5 and 7 on page 328 because no commit or backout activity will occur during load balancing. These queues are therefore the most suitable for moving, and the considerations listed below do not apply.
- Do these applications ever issue **MQCLOSE** calls against the queue before issuing a syncpoint?
If not, there is no potential for losing messages during the execution of steps 5 and 7 on page 328 because all commit or backout activity will have completed before the queue is closed. These queues are the next most suitable for moving and the considerations listed below do not apply.
- If applications **do** issue **MQCLOSE** calls against the queue before issuing a syncpoint, these queues are most likely to lose messages during the load balancing operation. This is because there might be uncommitted messages on the queue even though there are no open handles. These queues are therefore the least suitable for moving.

You should also consider the following points:

- There is no method for establishing whether there are uncommitted **MQPUT** or **MQGET** calls outstanding against the queue. In particular, the CURDEPTH attribute of a queue reflects both committed and uncommitted messages, and so is not a reliable indicator of whether all messages on the queue are committed.
- Using MQSeries security to restrict access to queues during the load balancing operation will not prevent the potential loss of messages between the execution of steps 5 and 7 on page 328 because it has no effect on the uncommitted activity on the queue.

Balancing loads

To identify which queues are using a page set, use the appropriate MQSeries commands. For example, to find out which queues are mapped to page set 02, first, find out which storage classes map to page set 02, by using this command:

```
DISPLAY STGCLASS(*) PSID(02)
```

Then use this command:

```
DISPLAY QUEUE(*) TYPE(QLOCAL) STGCLASS
```

to find out which queues use which storage class.

Load balancing by moving queues

To move queues and their messages from one page set to another, use the COPY or SCOPY, LOAD, and COMMAND functions of the MQSeries utility program, CSQUTIL. These are described in Chapter 14, “Using the MQSeries utilities” on page 227.

When you have identified the queue or queues that you want to move to a new page set, follow this procedure for each of these queues:

1. Ensure that the queue to be moved is not in use by any applications, that is, the queue attributes IPPROCS and OPPROCS are zero.
2. Change the following queue attributes:
 - PUT(ENABLED)
 - GET(ENABLED)
3. Define a temporary queue with the same attributes as the queue that is being moved:

```
DEFINE QL(TEMPQ) LIKE(Q_BEING_MOVED)
```

Note: If this temporary queue already exists from a previous run, delete it before doing the define.
4. Prevent applications from putting messages on the queue being moved by altering the queue definition to disable **MQPUTs**. Change this queue attribute:
 - PUT(DISABLED)
5. Run the COPY (or SCOPY) function to off-load the messages to a data set. See “Copying queues into a data set while the queue manager is running (COPY)” on page 246.

Note: Only committed messages will be copied.
6. Prevent applications from getting messages from the queue by altering the queue definition to disable **MQGETs**. Change this queue attribute:
 - GET(DISABLED)
7. Delete the queue you are moving, using the purge option to delete all messages associated with the queue.

Note: The delete purge will fail if the queue is open for inquire, set, or browse, or if there are uncommitted messages on the queue.

8. Define a new storage class which maps to the required page set, for example:

```
DEFINE STGCLASS(NEW) PSID(nn)
```

Add the new storage class definition to CSQINP2 ready for the next MQSeries subsystem restart.

9. Redefine the queue that is being moved, changing the storage class attribute. When the queue is redefined, it is based on the temporary queue created in step 3 on page 328:

```
DEFINE QL(Q_BEING_MOVED) LIKE(TEMPQ) STGCLASS(NEW)
```

10. The queue created in step 3 on page 328 is no longer required. Use the following command to delete it:

```
DELETE QL(TEMPQ)
```

11. If the queue being moved was defined in the CSQINP2 concatenation, change the STGCLASS attribute of the appropriate DEFINE QLOCAL command in the CSQINP2 concatenation. Add the REPLACE keyword so that the existing queue definition is replaced.

12. Use the LOAD function to restore the messages onto the queue from the data set generated by the COPY or SCOPY function in step 5 on page 328.

Figure 96 shows an extract from a load balancing job.

```
//UTILITY EXEC PGM=CSQUTIL,PARM=('CSQ1')
//STEPLIB DD DSN=th1qua1.SCSQANLE,DISP=SHR
// DD DSN=th1qua1.SCSQAUTH,DISP=SHR
//OUTPUT DD DSN=&&COPYMSG,
// SPACE=(CYL,(5,1),RLSE),UNIT=SYSDA,
// DCB=(RECFM=VBS,BLKSIZE=23200)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
COMMAND DDNAME(PUTDIS)
COPY QUEUE(Queue_TO_MOVE) DDNAME(OUTPUT)
COMMAND DDNAME(DEFTEMP)
LOAD QUEUE(Queue_TO_MOVE) DDNAME(OUTPUT)
/*
//PUTDIS DD *
ALTER QL(Queue_TO_MOVE) GET(ENABLED) PUT(ENABLED)
DELETE QL(TEMP_QUEUE) PURGE
DEFINE QL(TEMP_QUEUE) LIKE(Queue_TO_MOVE)
ALTER QL(Queue_TO_MOVE) PUT(DISABLED)
/*
//DEFTEMP DD *
ALTER QL(Queue_TO_MOVE) GET(DISABLED)
DELETE QL(Queue_TO_MOVE) PURGE
DEFINE STGCLASS(NEW) PSID(2)
DEFINE QL(Queue_TO_MOVE) LIKE(TEMP_QUEUE) STGCLASS(NEW)
DELETE QL(TEMP_QUEUE)
/*
```

Figure 96. Extract from a load balancing job

How to reduce a page set

If you have a large page set that is mostly empty (as shown by the DISPLAY USAGE command), you might want to reduce its size. The procedure to do this involves using the COPY, EMPTY, RESETPAGE, FORMAT, LOAD, and COMMAND functions of CSQUTIL (see “MQSeries utility program (CSQUTIL)” on page 229). This procedure will not work for page set zero; it is not practical to reduce the size of this page set.

1. Prevent all users, other than the MQSeries administrator, from using the queue manager. This could be done by changing the access security settings for example.
2. Wait until all queue manager use has ended; you might have to stop and restart the queue manager to achieve this. (Remember that if you do this, you will lose all your nonpersistent messages.)
3. Run the COPY function of CSQUTIL with the PSID option to copy all message data from the large page set and save them in a sequential data set.
4. Run the EMPTY function of CSQUTIL with the PSID option to remove all messages from the page set. Use the DISPLAY USAGE PSID(n) command to verify that the page set is totally empty.
5. Use the DISPLAY STGCLASS(*) PSID(n) command to identify all storage classes that relate to the page set that is to be reduced in size.
6. Use the DISPLAY QUEUE(*) TYPE(QLOCAL) STGCLASS command to identify all queues that use any of the storage classes identified in step 5.
7. Alter each queue that you have identified to use a different storage class that maps to a different page set. This does not have to be a permanent change to the queue, but is essential for the queue manager to be able to restart. If you do not do this, you could get 00C91B01 abends when the queue manager attempts to start.

Use the ALTER QLOCAL(q-name) STGCLASS(stgc1-name) command to alter the storage class attribute of each queue.
8. Use the STOP QMGR command with the QUIESCE or FORCE attribute to stop the queue manager.
9. Run the RESETPAGE function of CSQUTIL against all page sets other than the page set that is to be reduced in size. (You can choose to reset the page set in place, or you can copy and reset the page set.)
10. Define a new smaller page set data set to replace the large page set. Run the FORMAT function of CSQUTIL against it.
11. Define new log data sets (BSDS and active logs) with new data set names.
12. Restart the queue manager using the page sets created in steps 9 and 10 and the new BSDS and log data sets created in step 11.
13. Use the ALTER QLOCAL(q-name) STGCLASS(stgc1-name) command to reset the storage class attribute to the previous value for each queue altered in step 7.
14. Run the LOAD function of CSQUTIL to load back all the messages saved during step 3.
15. Allow all users access to the queue manager.

16. You can now delete the old large page set and the old BSDS and log data sets.

Backing up and recovering page sets

This section describes:

- “Creating a point of recovery”
- “Recovering page sets” on page 333

Creating a point of recovery

MQSeries can recover objects and persistent messages to their current state only if there is:

1. A copy of all page sets from an earlier point.
2. All the MQSeries logs since that point.

These represent a point of recovery.

Both objects and messages are held on page sets. Multiple objects and messages from different queues can exist on the same page set. Therefore, for recovery purposes, objects and messages cannot be backed up in isolation so that a page set must be backed up as a whole to ensure the proper recovery of the data.

The MQSeries recovery log contains a record of all persistent messages and changes made to objects. If MQSeries fails (for example, due to an I/O error on a page set), you can recover the page set by restoring the backup copy and restarting MQSeries. MQSeries applies the log changes to the page set from the point of the backup copy.

There are two ways of creating a point of recovery. The first involves stopping the queue manager thereby forcing all updates on to the page sets. The second involves taking ‘fuzzy’ backup copies of the page sets without stopping the queue manager.

The first will allow you to restart from the point of recovery, using only the backed up page set data sets and the logs from that point on.

If you use the second method, and your associated logs subsequently become damaged or lost you will not be able to use the fuzzy page set backup copies to recover. This is because the fuzzy page set backup copies contain an inconsistent view of the state of MQSeries and are dependent on the logs being available. If the logs are not available, you will have to return to the last set of backup page set copies taken while the subsystem was inactive (Method 1 below) and accept the loss of data from that time.

Method 1: Full backup

This method involves shutting MQSeries down. This forces all updates on to the page sets so that the page sets are in a consistent state.

1. Stop all MQSeries applications using the queue manager (allowing them to complete first). This could be done by changing the access security or queue settings, for example.
2. When all activity has completed, display and resolve any in-doubt units of recovery in the subsystem. (Use the MQSeries DISPLAY THREAD and

RESOLVE INDOUBT commands as described in the *MQSeries Command Reference* manual.)

This will bring the page sets to a consistent state; if you do not do this, or can't, your page sets might not be consistent, and you are effectively doing a "fuzzy" backup.

3. Issue the MQSeries command ARCHIVE LOG to ensure that the latest log data is written out to the log data sets, and a new log data set is started at the RBA value to be recorded in the next step. In order to restart from the point of recovery, you do not need to keep any log data sets up to and including that created by the ARCHIVE LOG command.
4. Issue the MQSeries command STOP QMGR MODE(QUIESCE). Record the lowest RBA value in the CSQI024I or CSQI025I messages (see the *MQSeries for OS/390 Messages and Codes* manual for information about these messages).
5. Take backup copies of the page sets (see "Backing up page sets").

Method 2: Fuzzy backup

This method does not involve shutting MQSeries down. Therefore, updates might be in virtual storage buffers during the backup process. This means that the page sets are not in a consistent state, and can only be used for recovery in conjunction with the logs.

1. Issue the MQSeries command DISPLAY USAGE and record the RBA value in the CSQI024I or CSQI025I message (see the *MQSeries for OS/390 Messages and Codes* manual for information about these messages).
2. Take backup copies of the page sets (see "Backing up page sets").
3. Issue the MQSeries command ARCHIVE LOG to ensure that the latest log data is written out to the log data sets. In order to restart from the point of recovery, you must keep copies of all the log data sets from that containing the recorded RBA value to that created by the ARCHIVE LOG command.

Backing up page sets

You can take a backup of your page sets in two ways:

- "Using Access Method Services"
- "Using volume dump and restore" on page 333.

To recover a page set, MQSeries needs to know how far back in the log to go. MQSeries maintains a log RBA number in page 0 of each page set, called the recovery log sequence number (LSN). This number is the starting RBA in the log from which MQSeries can recover the page set. When you back up a page set, this number is also copied.

If the copy is later used to recover the page set, MQSeries must have access to all of the log records from this RBA value to the current RBA. That means you must keep enough of the log records to enable MQSeries to recover from the oldest backup copy of a page set you intend to keep.

Using Access Method Services: You can use Access Method Services REPRO function (or any equivalent) to make copies of your page sets. You can do this whether or not MQSeries is running. If you want to do it while MQSeries is running, you must DEFINE the page sets with SHAREOPTIONS(2,3).

If you copy the page set while MQSeries is running you *must* use a copy utility that copies page 0 of the page set first – if you do not do this you could corrupt the data in your page set. Access Method Services REPRO meets this requirement.

If the process of dynamically expanding a page set is interrupted, for example by power to the system being lost, you can still use Access Method Services REPRO to take a backup of a page set.

If you perform an Access Method Services IDCAMS LISTCAT ENT('page set data set name') ALLOC, you will see that the HI-ALLOC-RBA will be higher than the HI-USED-RBA. Access Method Services REPRO will just copy the page set up to the high used RBA, and not give an error.

The next time this page set fills up it will be extended again, if possible, and the pages between the high used RBA and the highest allocated RBA will be used, along with another new extent.

For more information on the REPRO statement, see the *DFSMS/MVS Access Method Services for VSAM* or the *DFSMS/MVS Access Method Services for the Integrated Catalog Facility* manual.

Using volume dump and restore: Volume dump services dumps all the data sets on the volume. Likewise, restore volume services can (depending on the type of service) restore all the data sets.

Backing up your object definitions

You should also back up copies of your object definitions. To do this, use the MAKEDEF feature of the CSQUTIL COMMAND function (described in “MQSeries command management functions” on page 238).

You should do this whenever you take a backup copy of your queue manager, and keep the most current version.

Recovering page sets

If MQSeries has suffered a failure that has caused it to terminate, MQSeries can normally be restarted with all recovery being performed during restart. However, such recovery is not possible if any of your page sets or log data sets are not available. The extent to which you can now recover depends on the availability of backup copies of page sets and log data sets.

To restart from a point of recovery you must have:

- A backup copy of the page set that is to be recovered.
- If you used “fuzzy” backup, the log data set that included the recorded RBA value, the log data set that was made by the ARCHIVE LOG command, and all the log data sets between these.
- If you used full backup, but you do not have the log data set following that made by the ARCHIVE LOG command, you will need to use the RESETPAGE function of the CSQUTIL utility. The RBA identified using either method in “Creating a point of recovery” on page 331 is the restart point for the backed-up page sets.

To recover a page set to its current state, you must also have all the log data sets and records since the ARCHIVE LOG command.

There are two methods for recovering a page set. To use either method, the queue manager must not be running.

Simple recovery

This is the simpler method, and is appropriate for most recovery situations.

1. Delete and redefine the page set you want to restore from backup.
2. Use Access Method Services REPRO to copy the backup copy of the page set into the new page set. You should define your new page set with a secondary extent value so that it can be expanded dynamically.

Alternatively, you can rename your backup copy to the original name, or change the CSQP00xx DD statement in your queue manager procedure to point to your backup page set. However, if you then lose or corrupt the page set, you will no longer have a backup copy to restore from.

3. Restart the queue manager.
4. When the queue manager has restarted successfully, you can restart your applications
5. Reinstate your normal backup procedures for the restored page.

Advanced recovery

This method provides performance advantages if you have a large page set to recover, or if there has been a lot of activity on the page set since the last backup copy was taken. However, it requires more manual intervention than the simple method, which might increase the risk of error and the time taken to perform the recovery.

1. Delete and redefine the page set you want to restore from backup.
2. Use Access Method Services REPRO to copy the backup copy of the page set into the new page set. You should define your new page set with a secondary extent value so that it can be expanded dynamically.

Alternatively, you can rename your backup copy to the original name, or change the CSQP00xx DD statement in your queue manager procedure to point to your backup page set. However, if you then lose or corrupt the page set, you will no longer have a backup copy to restore from.

3. Change the CSQINP1 definitions for your queue manager to make the buffer pool associated with the page set being recovered as large as possible. By making the buffer pool this large, you might be able to keep all of the changed pages resident in the buffer pool and reduce the amount of I/O to the page set.
4. Restart the queue manager.
5. When the queue manager has restarted successfully, stop it (using quiesce) and then restart it using the normal buffer pool definition for that page set. After this second restart completes successfully, you can restart your applications
6. Reinstate your normal backup procedures for the restored page.

What happens when MQSeries is restarted

When MQSeries is restarted, it applies all changes made to the page set that are registered in the log, beginning at the restart point for the page set. MQSeries can recover multiple page sets in this way. The page set will be dynamically expanded, if required, during media recovery.

During restart MQSeries determines the log RBA to start from by taking the lowest value from the following:

- Recovery LSN from the checkpoint log record for each page set.
- Recovery LSN from page 0 in each page set.
- The RBA of the oldest incomplete unit of recovery in the system at the time the backup was taken.

All object definitions are stored on page set 0. Messages can be stored on any available page set.

Note: MQSeries cannot restart if page set 0 is not available.

Backing up and restoring queues using CSQUTIL

You can use the CSQUTIL utility functions for backing up and restoring queues. To back up a queue, use the COPY or SCOPY function to copy the messages from a queue onto a data set. To restore the queue, use the complementary function LOAD. For more information, see “MQSeries utility program (CSQUTIL)” on page 229.

Chapter 21. Example recovery scenarios

This chapter describes procedures for recovering MQSeries in the following circumstances:

- “Dealing with active log problems” on page 338
- “Dealing with archive log problems” on page 343
- “Dealing with BSDS errors” on page 346
- “BSDS recovery” on page 350
- “Dealing with page set problems” on page 353
- “Restarting if you have lost your log data sets” on page 355
- “Performing a cold start of MQSeries” on page 356
- “Dealing with IMS-related problems” on page 357
- “Dealing with hardware errors” on page 359

Day-to-day logging tasks are described in Chapter 19, “Managing the logs and the bootstrap data set” on page 315.

Note: In connection with some recovery procedures, it might be necessary to reset the BSDS. This will cause the current archive log data set sequence number to be reset to 1.

This means that you might subsequently get duplicate data set name errors when an archive log copy is created, because of previously created archive log data sets. You can avoid this by:

- Using data set names including a time stamp, by setting TSTAMP=YES or TSTAMP=EXT in CSQ6ARVP.
- Using a different data set name prefix, by setting ARCPFX1 and ARCPFX2 in CSQ6ARVP.

See “Using CSQ6ARVP” on page 76 for more information.

Dealing with active log problems

This section covers some of the more likely active log problems:

- “Out-of-space in active logs and delays in off-loading”
- “Dual logging is lost” on page 339
- “Write I/O errors on an active log data set” on page 339
- “I/O errors occur while reading the active log” on page 340
- “Active log stopped” on page 342

Out-of-space in active logs and delays in off-loading

The active log can fill up for several reasons, for example, delays in off-loading and excessive logging.

Symptoms

An out-of-space condition on the active log has serious consequences. When the active log becomes full, the MQSeries subsystem halts processing until an off-load has been completed. If the off-load processing stops when the active log is full, the MQSeries subsystem can abend. Corrective action is required before MQSeries can be restarted.

Because of the serious implications of this event, the MQSeries subsystem issues the following warning message when the last available active log data set is 75% full:

```
CSQJ110E +cpf LAST COPYn ACTIVE LOG DATA SET IS nnn PERCENT FULL
```

and reissues the message after each additional 5% of the data set space is filled. Each time the message is issued, the off-load process is started.

If the active log fills to capacity, MQSeries issues the following message:

```
CSQJ111A +cpf OUT OF SPACE IN ACTIVE LOG DATA SETS
```

and an off-load is started. The MQSeries subsystem then halts processing until an off-load has been completed.

System action

MQSeries waits for an available active log data set before resuming normal MQSeries processing. Normal shutdown, with either QUIESCE or FORCE, is not possible because the shutdown sequence requires log space to record system events related to shutdown (for example, checkpoint records). If the off-load processing stops when the active log is full, MQSeries forces itself to stop using an X'6C6' abend; restart in this case requires special attention. For more details, see the *MQSeries for OS/390 Problem Determination Guide*.

System programmer action

Additional active log data sets can be provided as required before restarting MQSeries. This permits MQSeries to continue its normal operation while the error causing the off-load problems is corrected. To add new active log data sets, use the change log inventory utility (CSQJU003) when MQSeries is not active. For more details about adding new active log data sets, see “Changing the BSDS” on page 321.

You should also consider increasing the number of logs in the CSQZPARM member by:

1. Making sure MQSeries is stopped, then using the Access Method Services DEFINE command to define a new active log data set.
2. Defining the new active log data set in the BSDS using the change log inventory utility (CSQJU003).

Restarting MQSeries: off-load starts automatically during startup, and work in progress when MQSeries was forced to stop is recovered.

To optimize the number of off-loads taken per day in your installation, you should consider adjusting the size of the active log data sets. Use Access Method Services to define new active log data sets, and use the change log inventory utility CSQJU003 to add the new data sets to the log inventory in the BSDS.

Operator action

Check whether the off-load process is waiting for a tape drive. If it is, mount a tape. If you cannot mount a tape, force MQSeries to stop by using OS/390 CANCEL.

Dual logging is lost

Symptoms

MQSeries issues the following message:

```
CSQJ004I +cpf ACTIVE LOG COPY n INACTIVE, LOG IN SINGLE MODE,
        ENDRBA=...
```

Having completed one active log data set, MQSeries found that the subsequent (COPY n) data sets were not off-loaded or were marked stopped.

System action

MQSeries continues in single mode until off-loading has been completed, then returns to dual mode.

System programmer action

None.

Operator action

Check that the off-load is proceeding and is not waiting for a tape mount. It might be necessary to run the print log map utility to determine the state of all data sets. It might also be necessary to define additional data sets.

Write I/O errors on an active log data set

Symptoms

MQSeries issues the following message:

```
CSQJ105E +cpf csect-name LOG WRITE ERROR DSNAME=...,
        LOGRBA=..., ERROR STATUS=ccccffss
```

System action

MQSeries carries out these steps:

1. Marks the log data set that has the error as TRUNCATED in the BSDS.
2. Goes on to the next available data set.

Active log problems

3. If dual active logging is used, truncates the other copy at the same point.

The data in the truncated data set is off-loaded later, as usual.

The data set is not *stopped* and is reused on the next cycle.

System programmer action

None.

Operator action

If errors on this data set still exist, take MQSeries down after the next off-load. Then use access method services (AMS) and the change log inventory utility to add a replacement. (For instructions, see "Changing the BSDS" on page 321.)

I/O errors occur while reading the active log

Symptoms

MQSeries issues the following message:

```
CSQJ106E +cpf LOG READ ERROR DSNAME=..., LOGRBA=...,  
        ERROR STATUS=ccccffss
```

System action

This depends on when the error occurred:

- If the error occurs during the off-load process, the process tries to read the RBA range from a second copy.
 - If no second copy exists, the active log data set is stopped.
 - If the second copy also has an error, only the original data set that triggered the off-load is stopped. The archive log data set is then terminated, leaving a gap in the archived log RBA range.
 - This message is issued:

```
CSQJ124E +cpf OFFLOAD OF ACTIVE LOG SUSPENDED FROM  
        RBA xxxxxx TO RBA xxxxxx DUE TO I/O ERROR
```
 - If the second copy is satisfactory, the first copy is not stopped.
- If the error occurs during recovery, MQSeries provides data from specific log RBAs requested from another copy or archive. If this is unsuccessful, recovery does not succeed, and MQSeries terminates abnormally.
- If the error occurs during restart, MQSeries terminates. All the copies of the active log data sets must be available to MQSeries.

System programmer action

Look for system messages, such as IEC prefixed messages, and try to resolve the problem using the recommended actions for these messages.

(You could write a program to archive as much of the stopped active log data set as possible. Then run the change log inventory utility to notify the BSDS of the new archive log and its log RBA range.) Repairing the active log does not solve the problem, because off-load does not go back to unload it.

If the active log data set has been stopped, it is not used for logging. The data set is not deallocated; it is still used for reading. Even if the

data set is not stopped, an active log data set that gives persistent errors should nevertheless be replaced.

Operator action

None. You are not told explicitly whether the data set has been stopped.

Replacing the data set

How you replace the data set depends on whether you are using single or dual active logging.

If you are using dual active logging:

1. Ensure that the data has been saved.

The data is saved on the other active log and this can be copied to a replacement active log.

2. Stop MQSeries and delete the data set in error using Access Method Services.
3. Redefine a new log data set using Access Method Services DEFINE so that you can write to it. Use the Access Method Services REPRO function to copy the good log into the redefined data set so that you have two consistent, correct logs again.
4. Use the change log inventory utility, CSQJU003, to update the information in the BSDS about the corrupt data set as follows:
 - a. Use the DELETE function to remove information about the corrupt data set.
 - b. Use the NEWLOG function to name the new data set as the new active log data set and give it the RBA range that was successfully copied.

The DELETE and NEWLOG functions can be run in the same job step. Put the DELETE statement before NEWLOG statement in the SYSIN input data set.

5. Restart MQSeries.

If you are using single active logging:

1. Ensure that the data has been saved.
2. Stop MQSeries.
3. Determine whether the data set with the error has been off-loaded:
 - a. Use the CSQJU003 utility to list information about the archive log data sets from the BSDS.
 - b. Search the list for a data set whose RBA range includes the RBA of the corrupt data set.
4. If the corrupt data set has been off-loaded, copy its backup in the archive log to a new data set. Then, skip to step 6.
5. If an active log data set is stopped, an RBA is not off-loaded. Use Access Method Services REPRO to copy the data from the corrupt data set to a new data set.

If further I/O errors prevent you from copying the entire data set, a gap occurs in the log.

Note: MQSeries restart will not be successful if a gap in the log is detected.

Active log problems

6. Use the change log inventory utility, CSQJU003, to update the information in the BSDS about the corrupt data set as follows:
 - a. Use the DELETE function to remove information about the corrupt data set.
 - b. Use the NEWLOG function to name the new data set as the new active log data set and to give it the RBA range that was successfully copied.

The DELETE and NEWLOG functions can be run in the same job step. Put the DELETE statement before NEWLOG statement in the SYSIN input data set.

7. Restart MQSeries.

Active log stopped

Symptoms

MQSeries issues the following message :

```
CSQJ030E +cpf RBA RANGE startrba TO endrba NOT AVAILABLE IN ACTIVE  
LOG DATA SETS
```

System action

The active log data sets that contain the RBA range reported in message CSQJ030E are unavailable to MQSeries. The status of these logs is STOPPED in the BSDS. MQSeries will terminate with a dump.

System programmer action

This problem must be resolved before restarting MQSeries. The log RBA range must be available for MQSeries to be recoverable. An active log that is marked as STOPPED in the BSDS will never be reused or archived and this will create a hole in the log.

Look for messages that indicate why the log data set has stopped, and follow the instructions for those messages.

The BSDS active log inventory needs to be modified to reset the STOPPED status. To do this, follow this procedure after MQSeries has terminated:

1. Use the print log utility (CSQJU004) to obtain a copy of the BSDS log inventory. This shows the status of the log data sets.
2. Use the DELETE function of the change log inventory utility (CSQJU003) to delete the active log data sets that are marked as STOPPED.
3. Use the NEWLOG function of CSQJU003 to add the active logs back into the BSDS inventory. The starting and ending RBA for each active log data set must be specified on the NEWLOG statement. (The correct values to use can be found from the print log utility report obtained in Step 1.)
4. Rerun CSQJU004. The active log data sets that were marked as STOPPED will now be shown as NEW and NOT REUSABLE. These active logs will be archived in due course.
5. Restart MQSeries.

Note: If your MQSeries subsystem is running in dual BSDS mode, you must update both BSDS inventories.

Dealing with archive log problems

This section covers the most likely archive log problems:

- “Allocation problems”
- “Write I/O errors on the archive log during off-load”
- “Read I/O errors on the archive data set while MQSeries is restarting” on page 344
- “Insufficient DASD space to complete off-load processing” on page 344

Allocation problems

Symptoms

MQSeries issues the following message:

```
CSQJ103E +cpf LOG ALLOCATION ERROR DSNAME=dsname,
        ERROR STATUS=eeeeiii
```

OS/390 dynamic allocation provides the ERROR STATUS. If the allocation was for off-load processing, the following message is also displayed:

```
CSQJ115E +cpf OFFLOAD FAILED, COULD NOT ALLOCATE AN ARCHIVE
        DATA SET
```

System action

The following actions take place:

- If the input is needed for recovery, recovery is not successful, and MQSeries will abend.
- If the active log had become full and an off-load was scheduled, off-load tries again the next time it is triggered. The active log does not wrap around.

System programmer action

None.

Operator action

Check the allocation error code for the cause of the problem, and correct it. Ensure that drives are available, and either restart or wait for the off-load to be retried. Be careful if a DFP/DFSMS ACS user-exit filter has been written for an archive log data set, because this can cause a device allocation error when the MQSeries subsystem tries to read the archive log data set. For details of how to solve this problem, see “Archiving to DASD volumes” on page 310.

Write I/O errors on the archive log during off-load

Symptoms

No specific MQSeries message is issued for write I/O errors.

Only an OS/390 error recovery program message appears. If you get MQSeries message CSQJ128E, the off-load task has terminated abnormally and you should consult the *MQSeries for OS/390 Messages and Codes* manual.

Archive log problems

System action

The following actions take place:

- Off-load abandons the output data set; no entry is made in the BSDS.
- Off-load dynamically allocates a new archive and restarts off-loading from the point at which it was previously triggered. If there is dual archiving, the second copy waits.
- If an error occurs on the new data set, then:
 - In dual archive mode, this message is generated and the off-load processing changes to single mode:

```
CSQJ114I +cpf ERROR ON ARCHIVE DATA SET, OFFLOAD  
CONTINUING WITH ONLY ONE ARCHIVE DATA SET BEING  
GENERATED
```
 - In single archive mode, the output data set is abandoned. Another attempt to off-load this RBA range is made the next time off-load is triggered.
 - The active log does not wrap around; if there are no more active logs, data is not lost.

System programmer action

None.

Operator action

Ensure that off-load is allocated on a reliable drive and control unit.

Read I/O errors on the archive data set while MQSeries is restarting

Symptoms

No specific MQSeries message is issued; only the OS/390 error recovery program message appears.

System action

This depends on whether a second copy exists:

- If a second copy exists, it is allocated and used.
- If a second copy does not exist, restart is not successful.

System programmer action

None.

Operator action

Try to restart, using a different drive.

Insufficient DASD space to complete off-load processing

Symptoms

While off-loading the active log data sets to DASD, the process terminates unexpectedly. MQSeries issues the following message:

```
CSQJ128E +cpf LOG OFF-LOAD TASK FAILED FOR ACTIVE LOG nnnnn
```

The error is preceded by OS/390 messages IEC030I, IEC031I, or IEC032I.

System action

MQSeries de-allocates the data set on which the error occurred. If MQSeries is running in dual archive mode, MQSeries changes to single

archive mode and continues the off-load. If the off-load cannot be completed in single archive mode, the active log data sets cannot be off-loaded, and the state of the active log data sets remains NOT REUSABLE. Another attempt to off-load the RBA range of the abandoned active log data sets is made the next time the off-load process is triggered.

System programmer action

None.

Operator action

Quiesce the MQSeries subsystem (using `+cpf STOP QMGR MODE(QUIESCE)`) to restrict logging activity until the OS/390 abend is resolved.

The most likely causes of these symptoms are:

- The size of the archive log data set is too small to contain the data from the active log data sets during off-load processing. All the secondary space allocations have been used. This condition is normally accompanied by OS/390 message IEC030I.

To solve the problem, either increase the primary or secondary allocations (or both) for the archive log data set (in CSQZPARM), or reduce the size of the active log data set. If the data to be off-loaded is particularly large, you can mount another online storage volume or make one available to MQSeries.

- All available space on the DASD volumes to which the archive data set is being written has been exhausted. This condition is normally accompanied by OS/390 message IEC032I.

To solve the problem, make more space available on the DASD volumes, or make available another online storage volume for MQSeries.

- The primary space allocation for the archive log data set (as specified in CSQZPARM) is too large to allocate to any available online DASD device. This condition is normally accompanied by OS/390 message IEC032I.

To solve the problem, make more space available on the DASD volumes, or make available another online storage volume for MQSeries. If this is not possible, you must adjust the value of PRIQTY in CSQ6ARVP to reduce the primary allocation. (For details, see “Using CSQ6ARVP” on page 76.)

Note: If the primary allocation is reduced, the size of the secondary space allocation might have to be increased to avoid future abends.

Dealing with BSDS errors

For background information about the bootstrap data set (BSDS), see “What the bootstrap data set is for” on page 305.

This section describes the more likely BSDS problems:

- “I/O error”
- “Error occurs while opening the BSDS” on page 347
- “Unequal time stamps” on page 347
- “Out of synchronization” on page 348
- “Log content does not agree with the BSDS information” on page 349

Normally, there are two copies of the BSDS, but if one is damaged, MQSeries immediately changes to single BSDS mode. However, the damaged copy of the BSDS must be recovered before restart. If you are in single mode and damage the only copy of the BSDS, or if you are in dual mode and damage both copies, see “BSDS recovery” on page 350.

This section covers some of the BSDS problems that can occur at startup. Problems *not* covered here include:

- +cpf RECOVER BSDS command errors (messages CSQJ301E - CSQJ307I)
- Change log inventory utility errors (message CSQJ123E)
- Errors in the BSDS backup being dumped by off-load (message CSQJ125E)

For information about those problems, see the *MQSeries for OS/390 Messages and Codes* manual.

I/O error

Symptoms

MQSeries changes to single BSDS mode and issues the user message:

```
CSQJ126E +cpf BSDS ERROR FORCED SINGLE BSDS MODE
```

This is followed by one of these messages:

```
CSQJ107E +cpf READ ERROR ON BSDS  
          DSNAME=... ERROR STATUS=...
```

```
CSQJ108E +cpf WRITE ERROR ON BSDS  
          DSNAME=... ERROR STATUS=...
```

System action

The BSDS mode changes from dual to single.

System programmer action

None.

Operator action

Carry out these steps:

1. Use Access Method Services to rename or delete the damaged BSDS and to define a new BSDS with the same name as the BSDS that had the error. Control statements can be found in job CSQ4BSDS in thlqual.SCSQPROC.

2. Issue the MQSeries command `+cpf RECOVER BSDS` to make a copy of the good BSDS in the newly allocated data set and reinstate dual BSDS mode. See also “BSDS recovery” on page 350.

Error occurs while opening the BSDS

Symptoms

MQSeries issues the following message:

```
CSQJ100E +cpf ERROR OPENING BSDSn DSNAME=..., ERROR STATUS=eeii
```

where *eeii* is the VSAM return code. For information about VSAM codes, see the *DFSMS/MVS Macro Instructions for Data Sets* manual. For an explanation of this message, see the *MQSeries for OS/390 Messages and Codes* manual.

System action

During system initialization, the startup is terminated.

During a `+cpf RECOVER BSDS` command, the system continues in single BSDS mode.

System programmer action

None.

Operator action

This depends on when the error occurred:

If the error occurred at restart:

1. Use Access Method Services to delete or rename the damaged data set, to define a replacement data set, and to copy the remaining BSDS to the replacement with the Access Method Services REPRO command.
2. Use the command `+cpf START QMGR` to start the MQSeries subsystem.

If the error occurred during the RECOVER BSDS command:

1. Use Access Method Services to rename or delete the damaged BSDS and to define a new BSDS with the same name as the BSDS that had the error. Control statements can be found in job CSQ4BSDS in `thlqual.SCSQPROC`.
2. Issue the MQSeries command `+cpf RECOVER BSDS` to make a copy of the good BSDS in the newly allocated data set and reinstate dual BSDS mode. See also “BSDS recovery” on page 350.

Unequal time stamps

Symptoms

MQSeries issues the following message:

```
CSQJ120E +cpf DUAL BSDS DATA SETS HAVE UNEQUAL TIME STAMPS,  
SYSTEM BSDS1=...,BSDS2=...,  
UTILITY BSDS1=...,BSDS2=...
```

The possible causes are:

- One of the volumes containing the BSDS has been restored. All information on the restored volume is down-level. If the volume contains any active log data sets or MQSeries data, they are also down-level. The down-level volume has the lower time stamp.
- Dual logging has degraded to single logging, and you are trying to start without recovering the damaged log.
- The MQSeries subsystem terminated abnormally after updating one copy of the BSDS but before updating the second copy.

System action

MQSeries startup is terminated.

System programmer action

None.

Operator action

Carry out these steps:

1. Run the print log map utility on both copies of the BSDS, compare the lists to determine which copy is accurate or current.
2. Rename the down-level data set and define a replacement for it.
3. Copy the good data set to the replacement data set, using access method services.
4. Determine whether the volume containing the down-level BSDS has been restored. If it has been restored, all data on that volume, such as the active log data, is also down-level.

If the restored volume contains active log data and you were using dual active logs on separate volumes, you need to copy the current version of the active log to the down-level data set. "Recovering logs" on page 323 tells you how to do this.

Out of synchronization

Symptoms

MQSeries issues the following message:

```
CSQJ122E +cpf DUAL BSDS DATA SETS ARE OUT OF SYNCHRONIZATION
```

The system time stamps of the two data sets are identical. Differences can exist if operator errors occurred while the change log inventory utility was being used. (For example, the change log inventory utility was only run on one copy.) The change log inventory utility sets a private time stamp in the BSDS control record when it starts, and a close flag when it ends. MQSeries checks the change log inventory utility time stamps and, if they are different, or they are the same but one close flag is not set, MQSeries compares the copies of the BSDSs. If the copies are different, CSQJ122E is issued.

System action

MQSeries startup is terminated.

System programmer action

None.

Operator action

Carry out these steps:

1. Run the print log map utility on both copies of the BSDS, and compare the lists to determine which copy is accurate or current.
2. Rename the data set that had the problem, and define a replacement for it.
3. Copy the accurate data set to the replacement data set, using access method services.

Log content does not agree with the BSDS information

Symptoms

This message appears:

```
CSQJ102E +cpf LOG RBA CONTENT OF LOG DATA SET DSNAME=...,
          STARTRBA=..., ENDRBA=...,
          DOES NOT AGREE WITH BSDS INFORMATION
```

This message indicates that the change log inventory utility was used incorrectly or that a down-level volume is being used.

System action

MQSeries startup processing is terminated.

System programmer action

None.

Operator action

Run the print log map utility and the change log inventory utility to print and correct the contents of the BSDS.

BSDS recovery

If MQSeries is operating in dual BSDS mode and one BSDS becomes damaged, forcing MQSeries into single BSDS mode, MQSeries continues to operate without a problem (until the next restart). To return the environment to dual BSDS mode:

1. Use Access Method Services to rename or delete the damaged BSDS and to define a new BSDS with the same name as the damaged BSDS. Control statements can be found in job CSQ4BSDS in thlqual.SCSQPROC. (See “Task 11: Create the bootstrap and log data sets” on page 61.)
2. Issue the MQSeries command `+cpf RECOVER BSDS` to make a copy of the valid BSDS in the newly allocated data set and to reinstate dual BSDS mode.

If MQSeries is operating in single BSDS mode and the BSDS is damaged, or if MQSeries is operating in dual BSDS mode and both BSDSs are damaged, MQSeries stops and does not restart until the BSDS data sets are repaired. In this case:

1. Locate the BSDS associated with the most recent archive log data set. The data set name of the most recent archive log appears on the OS/390 console in the last occurrence of message CSQJ003I, which indicates that off-loading has been completed successfully. In preparation for the rest of this procedure, it is a good practice to keep a log of all successful archives noted by that message:
 - If archive logs are on DASD, the BSDS is allocated on any available DASD. The BSDS name is like the corresponding archive log data set name; change only the first letter of the last qualifier, from A to B, as in the example below:

Archive log name	CSQ.ARCHLOG1.A0000001
BSDS copy name	CSQ.ARCHLOG1.B0000001
 - If archive logs are on tape, the BSDS is the first data set of the first archive log volume. The BSDS is not repeated on later volumes.
2. If the most recent archive log data set has no copy of the BSDS (presumably because an error occurred when off-loading it), then locate an earlier copy of the BSDS from an earlier off-load.
3. Rename *damaged* BSDSs using the Access Method Services ALTER command with the NEWNAME option. If you decide to delete a damaged BSDS, use the Access Method Services DELETE command. For each damaged BSDS, use Access Method Services to define a new BSDS as a replacement data set. Job CSQ4BSDS in thlqual.SCSQPROC contains Access Method Services control statements to define a new BSDS. (See “Task 11: Create the bootstrap and log data sets” on page 61.)
4. Use Access Method Services to delete or rename the damaged data set, to define a replacement data set, and to copy the remaining BSDS to the replacement with the Access Method Services REPRO command.
5. Use the Access Method Services REPRO command to copy the BSDS from the archive log to one of the replacement BSDSs you defined in step 3. Do not copy any data to the second replacement BSDS—you do that in step 6 on page 352.
 - a. Print the contents of the replacement BSDS.

Use the print log map utility (CSQJU004) to print the contents of the replacement BSDS. This enables you to review the contents of the replacement BSDS before continuing your recovery work.

b. Update the archive log data set inventory in the replacement BSDS.

Examine the output from the print log map utility and check that the replacement BSDS does not contain a record of the archive log from which the BSDS was copied. If the replacement BSDS is an old copy, its inventory might not contain all archive log data sets that were created more recently. Therefore, the BSDS inventory of the archive log data sets must be updated to reflect the current subsystem inventory.

Use the change log inventory utility (CSQJU003) NEWLOG statement to update the replacement BSDS, adding a record of the archive log from which the BSDS was copied. If the archive log data set is password-protected, be certain to use the PASSWORD option of the NEWLOG function. Also, make certain the CATALOG option of the NEWLOG function is properly set to CATALOG=YES if the archive log data set is cataloged. Use the NEWLOG statement to add any additional archive log data sets that were created later than the BSDS copy.

c. Update passwords in the replacement BSDS.

The BSDS contains passwords for the archive log data sets and for the active log data sets. To ensure that the passwords in the replacement BSDS reflect the current passwords used by your installation, use the change log inventory ARCHIVE utility function with the PASSWORD option.

d. Update the active log data set inventory in the replacement BSDS.

In unusual circumstances, your installation could have added, deleted, or renamed active log data sets since the BSDS was copied. In this case, the replacement BSDS does not reflect the actual number or names of the active log data sets your installation has currently in use.

If you need to delete an active log data set from the replacement BSDS log inventory, use the change log inventory utility DELETE function.

If you need to add an active log data set to the replacement BSDS log inventory, use the change log inventory utility NEWLOG function. Ensure that the RBA range is specified correctly on the NEWLOG function. If the active log data set is password-protected, be sure to use the PASSWORD option.

If you need to rename an active log data set in the replacement BSDS log inventory, use the change log inventory utility DELETE function, followed by the NEWLOG function. Be sure that the RBA range is specified correctly on the NEWLOG function. If the active log data set is password-protected, be certain to use the PASSWORD option.

- e. Update the active log RBA ranges in the replacement BSDS.

Later, when MQSeries restarts, it compares the RBAs of the active log data sets listed in the BSDS with the RBAs found in the actual active log data sets. If the RBAs do not agree, MQSeries does not restart. The problem is magnified when a particularly old copy of the BSDS is used. To solve this problem, you can use the change log inventory utility (CSQJU003) to adjust the RBAs found in the BSDS using the RBAs in the actual active log data sets. This can be done by:

- Using the print log records utility (CSQ1LOGP) to print a summary report of the active log data set. This shows the starting and ending RBAs.
- Comparing the actual RBA ranges with the RBA ranges you have just printed, when the RBAs of all active log data sets are known.

If the RBA ranges are equal for all active log data sets, you can proceed to the next recovery step without any additional work.

If the RBA ranges are not equal, then the values in the BSDS must be adjusted to reflect the actual values. For each active log data set that needs to have the RBA range adjusted, use the change log inventory utility DELETE function to delete the active log data set from the inventory in the replacement BSDS. Then use the NEWLOG function to redefine the active log data set to the BSDS. If the active log data sets are password-protected, be certain to use the PASSWORD option of the NEWLOG function.

- f. If only two active log data sets are specified for each copy of the active log, MQSeries can have difficulty during restart. The problem can arise when one of the active log data sets is full and has not been off-loaded, while the second active log data set is close to filling. In this case, add a new active log data set for each copy of the active log and define each new active log data set in the replacement BSDS log inventory.

Use the Access Method Services DEFINE command to define a new active log data set for each copy of the active log. The control statements to accomplish this task can be found in job CSQ4BSDS in thlqual.SCSQPROC. (See "Task 11: Create the bootstrap and log data sets" on page 61.) Once the active log data sets are physically defined and allocated, use the change log inventory utility NEWLOG function to define the new active log data sets in the replacement BSDS. The RBA ranges need not be specified on the NEWLOG statement; however, if the active log data sets are password-protected, be certain to use the PASSWORD option of the NEWLOG function.

6. Copy the updated BSDS to the second new BSDS data set. The BSDSs are now identical.

Consider using the print log map utility (CSQJU004) to print the contents of the second replacement BSDS at this point.

7. See "Dealing with active log problems" on page 338 for information about what to do if you have lost your current active log data set.
8. Restart MQSeries, using the newly constructed BSDS. MQSeries determines the current RBA and what active logs need to be archived.

Dealing with page set problems

This section covers the problems that you might encounter with page sets:

- “Page set I/O errors” describes what happens if a page set is damaged.
- “Page set full” describes what happens if there is not enough space on the page set for any more **MQPUT** operations.

Page set I/O errors

Problem A page set has an I/O error.

Symptoms

This message is issued:

```
CSQP004I +cpf csect-name I/O ERROR STATUS ret-code
PSID psid RBA rba
```

System action

MQSeries terminates abnormally.

System programmer action

None.

Operator action

Repair the I/O error cause.

If none of the page sets are damaged, restart MQSeries. From the logs, MQSeries automatically restores the page set to a consistent state.

If one or more page sets are damaged, restore them from a backup copy and restart MQSeries. As before, from the logs MQSeries automatically applies any updates that are necessary.

You cannot restart MQSeries if page set zero is not available. However, if one of the other page sets is not available, you can comment out the page set DD statement in the MQSeries start-up JCL procedure. This lets you defer recovery of the defective page set, enabling other users to continue accessing MQSeries.

A reason code of MQRC_PAGESET_ERROR is returned to any application that tries to access a queue defined on a page set that is not available. When you have restored the defective page set, restore its associated DD statement and restart MQSeries.

The operator actions described here are only possible if all log data sets are available. If your log data sets are lost or damaged, see “Restarting if you have lost your log data sets” on page 355.

Page set full

Problem

There is not enough space on a page set for one of the following:

- **MQPUT** or **MQPUT1** calls to be completed
- Object manipulation commands to be completed (for example, DEFINE QLOCAL)
- **MQOPEN** calls for dynamic queues to be completed

Symptoms

The request fails with reason code MQRC_PAGESET_FULL. The queue manager is unable to complete the request because, there is not enough space remaining on the page set.

The cause of this problem could be due to messages accumulating on a transmission queue because they cannot be sent to another system.

System action

Further requests that use this page set are blocked until enough messages are removed or objects deleted to make room for the new incoming requests.

Operator action

Use the MQSeries command DISPLAY USAGE PSID(*) to identify which page set is full.

System programmer action

You can either enlarge the page set involved or reduce the loading on that page set by moving queues to another page set. See Chapter 20, "Managing page sets" on page 325 for more information about these tasks. If the cause of the problem is messages accumulating on the transmission queue, consider starting distributed queuing to transmit the messages.

Restarting if you have lost your log data sets

If, after stopping MQSeries (using the STOP QMGR command), both copies of the log are lost or found to be corrupt, it is possible to restart MQSeries providing you have a consistent set of page sets (produced using “Method 1: Full backup” on page 331).

Follow this procedure:

1. Define new page sets to correspond to each existing page set in your MQSeries subsystem. See “Task 12: Define your page sets” on page 62 for information about page set definition.

Ensure that each new page set is larger than the corresponding source page set.

2. Use the FORMAT function of CSQUTIL to format the destination page set. See “Formatting page sets (FORMAT)” on page 231 for more details.
3. Use the RESETPAGE function of CSQUTIL to copy the existing page sets or reset them in place, and reset the log RBA in each page. See “Copying a page set and resetting the log (RESETPAGE)” on page 235 for more information about this function.
4. Redefine your MQSeries log data sets and BSDS using CSQJU003 (see “The change log inventory utility (CSQJU003)” on page 256).
5. Restart MQSeries, using the new page sets. To do this, you do one of the following:
 - Change the MQSeries startup procedure to reference the new page sets. See “Task 12: Define your page sets” on page 62 for more details.
 - Use Access Method Services to delete the old page sets and then rename the new page sets, giving them the same names as the old page sets.

Attention: Before you delete any MQSeries page set, be sure that you have made the required backup copies.

Performing a cold start of MQSeries

If MQSeries has terminated abnormally you might not be able to restart it. This could be because your page sets or logs have been lost, truncated, or corrupted. If this has happened, you will have to perform a *cold start* to restart MQSeries.

Attention

Performing a cold start will enable you to recover your MQSeries system and your object definitions; you will **not** be able to recover your message data. This means that you should only perform a cold start if you are unable to restart MQSeries any other way.

When you have successfully restarted, all your MQSeries objects will be defined and available for use, but there will be no message data.

To cold start MQSeries, follow this procedure:

1. Prepare the object definition statements that will be used when you restart MQSeries. To do this, either:
 - If page set zero is available, use the CSQUTIL SDEFS function (see “Producing a list of MQSeries define commands (SDEFS)” on page 243)
 - If page set zero is not available, use the definitions from the last time you backed up your object definitions (see “Backing up your object definitions” on page 309)
2. Redefine your queue manager data sets (do not do this until you have completed step 1).
 - See “Task 11: Create the bootstrap and log data sets” on page 61 for information about redefining your log data sets and BSDS.
 - See “Task 12: Define your page sets” on page 62 for information about redefining your page sets.
3. Restart MQSeries using the newly defined and initialized log data sets, BSDS, and page sets. Use the object definition input statements that you created in step 1 as input in the CSQINP2 initialization input data set.

Dealing with IMS-related problems

This section includes plans for problems that you might encounter in the IMS environment:

“IMS application problem”

“IMS is not operational”

“IMS is unable to connect to MQSeries” on page 358

IMS application problem

Problem

An IMS application terminates abnormally.

Symptoms

The following message is sent to the user’s terminal:

```
DFS555I TRANSACTION tranid ABEND abcode
MSG IN PROCESS: message data:
```

where *tranid* represents any IMS transaction that is terminating abnormally and *abcode* is the abend code.

System action

IMS requests the adapter to resolve the unit of recovery. IMS remains connected to MQSeries.

System programmer action

None.

Operator action

As indicated in message DFS554A on the IMS master terminal.

IMS is not operational

Problem

IMS is not operational.

Symptoms

More than one symptom is possible:

- IMS waits or loops

Because MQSeries cannot detect a wait or loop in IMS, you must find the origin of the wait or loop. This can be IMS, IMS applications, or the IMS adapter.

- IMS terminates abnormally.
 - See the manuals *IMS/ESA Messages and Codes* and *IMS/ESA Failure Analysis Structure Tables* for more information.
 - If threads are connected to MQSeries when IMS terminates, MQSeries issues message CSQ3201E. This message indicates that MQSeries end-of-task (EOT) routines have been run to clean up and disconnect any connected threads.

System action

MQSeries detects the IMS error and:

- Backs out in-flight work.

- Saves in-doubt units of recovery to be resolved when IMS is reconnected.

System programmer action

None.

Operator action

Resolve and correct the problem that caused IMS to terminate abnormally, then carry out an emergency restart of IMS. The emergency restart:

- Backs out in-flight transactions that changed IMS resources.
- Remembers the transactions with access to MQSeries that might be in doubt.

It might be necessary to restart the connection to MQSeries with the IMS command:

```
/START SUBSYS subsysname
```

During startup, IMS requests the adapter to resolve in-doubt units of recovery.

IMS is unable to connect to MQSeries

Problem

The IMS adapter cannot connect to MQSeries.

Symptoms

IMS remains operative. The IMS adapter issues these messages for control region connect:

```
CSQQ001I  
CSQQ002E  
CSQQ003E  
CSQQ004E  
CSQQ005E  
CSQQ007E
```

For details, see the *MQSeries for OS/390 Messages and Codes* manual.

If an IMS application program tries to access MQSeries while the IMS adapter cannot connect, it can either receive a completion code and reason code or terminate abnormally. This depends on the value of the REO option in the SSM member of IMS PROCLIB.

System action

All connection errors are also reported in the IMS message DFS3611.

System programmer action

None.

Operator action

Analyze and correct the problem, then restart the connection with the IMS command:

```
/START SUBSYS subsysname
```

IMS requests the adapter to resolve in-doubt units of recovery.

Dealing with hardware errors

If a hardware error causes data to be unreadable on your subsystem, MQSeries can still be recovered by using the *media recovery* technique:

1. To recover the data, you need a backup copy of the data. Use Access Method Services REPRO regularly to make a copy of your data.
2. Reinstall the most recent backup copy.
3. Restart MQSeries.

The more recent your backup copy, the more quickly your subsystem can be made available again.

When MQSeries restarts, it uses the archive logs to reinstall changes made since the backup copy was taken. You must keep sufficient archive logs to enable MQSeries to reinstall the subsystem fully. Do not delete archive logs until there is a backup copy that includes all the changes in the log.

Part 7. Monitoring performance and resource usage

Chapter 22. Monitoring performance and resource usage	363
Getting snapshots of MQSeries	364
Using DISPLAY commands	364
Using CICS adapter statistics	364
Investigating performance problems	364
Symptoms of reduced performance	365
Investigating the overall system	365
Using System Management Facility	366
Reporting data in SMF	366
Allocating additional SMF buffers	366
Using other products with MQSeries	367
Using Resource Measurement Facility	367
Using Performance Reporter for OS/390	367
Using the CICS monitoring facility	367
Using MQSeries trace	368
Starting MQSeries trace	368
Controlling MQSeries trace	368
Specifying trace keywords	368
Effect of trace on MQSeries performance	369
Using MQSeries events	370
Chapter 23. Interpreting MQSeries performance statistics	371
SMF type 115 record layout	371
SMF type 115 record header description	371
Processing type 115 SMF records	372
Record subtypes	372
Where the information comes from	372
Self-defining sections for type 115 records	373
Message manager statistics	374
Interpretation	374
Data manager statistics	375
Interpretation	375
Buffer manager statistics	376
Interpreting buffer manager statistics	377
Buffer pool management	377
Log manager statistics	380
Interpreting log manager statistics	381
Sample SMF statistics records	382
Chapter 24. Interpreting MQSeries accounting data	385
SMF type 116 record layout	385
SMF type 116 record header description	385
Processing type 116 SMF records	386
Where the information comes from	386
Self-defining sections	387
Message manager accounting	388
Interpretation	389
Sample SMF accounting record	390

Chapter 22. Monitoring performance and resource usage

This part describes how to monitor the performance and resource usage of an MQSeries subsystem.

- It outlines some of the information that you can retrieve and briefly describes a general approach to investigating performance problems. (You can find information about dealing with performance problems in the *MQSeries for OS/390 Problem Determination Guide*.)
- It describes how you can obtain statistics related to the performance of an MQSeries subsystem by using SMF records.
- It describes how to gather accounting data to enable you to charge your customers for their use of your MQSeries subsystems.
- It describes how to use MQSeries events (alerts) to monitor your systems.

These are some of the tools you might use to monitor MQSeries; they are described in the sections that follow:

- Tools provided by MQSeries:
 - “Using DISPLAY commands” on page 364
 - “Using CICS adapter statistics” on page 364
 - “Using MQSeries events” on page 370
- OS/390 service aids:
 - “Using System Management Facility” on page 366
- Other IBM licensed programs:
 - “Using Resource Measurement Facility” on page 367
 - “Using Performance Reporter for OS/390” on page 367
 - “Using the CICS monitoring facility” on page 367

Information about interpreting the data gathered by the performance statistics trace is given in Chapter 23, “Interpreting MQSeries performance statistics” on page 371.

Information about interpreting the data gathered by the accounting trace is given in Chapter 24, “Interpreting MQSeries accounting data” on page 385.

Getting snapshots of MQSeries

You can get an idea of the current the state of your MQSeries subsystem by using the DISPLAY commands and, for the CICS adapter, the CICS adapter panels.

Using DISPLAY commands

You can use the MQSeries DISPLAY commands to obtain information about the current state of MQSeries. They provide information on the status of the command server, process definitions, queues, the queue manager, and so on. These commands are:

- DISPLAY CHANNEL
- DISPLAY CHSTATUS
- DISPLAY CMDSERV
- DISPLAY CLUSQMGR
- DISPLAY DQM
- DISPLAY PROCESS
- DISPLAY QUEUE
- DISPLAY QMGR
- DISPLAY SECURITY
- DISPLAY STGCLASS
- DISPLAY THREAD
- DISPLAY TRACE
- DISPLAY USAGE

These commands provide a snapshot of the system *only* at the moment the command was processed. If you want to examine trends in the system, you must start an MQSeries trace and analyze the results over a period of time.

Note: You must be authorized to use these commands.

For the detailed syntax of each command, see the *MQSeries Command Reference* manual. All of the functions of these commands (except DISPLAY CMDSERV and DISPLAY TRACE) are also available through the operations and control panels.

Using CICS adapter statistics

If you are an authorized CICS user, you can use the CICS adapter control panels to display CICS adapter statistics dynamically. These statistics provide a snapshot of information related to CICS thread usage and situations when all threads are busy. The display connection panel can be refreshed by pressing the Enter key. For more information, see “Displaying details of connections and CICS tasks” on page 135.

Investigating performance problems

The performance measurement tools discussed so far are useful for monitoring MQSeries, as well as for MQSeries problem determination.

This section presents an overview of problem investigation and analyzes reports generated by performance monitoring tools.

Symptoms of reduced performance

Performance can be adversely affected by:

- Buffer pools that are the wrong size
- Lack of real storage
- I/O contention for page sets
- I/O contention for the logs
- Log buffer thresholds that are set wrongly
- Wrong numbers of log buffers
- Large messages
- Units of recovery that last a long time, incorporating many messages per syncpoint
- Messages that remain on a queue for a long time
- RACF auditing
- Unnecessary security checks
- Inefficient program design

When you analyze performance data, always start by looking at the overall system before you decide that you have a specific MQSeries problem. Remember that almost all symptoms of reduced performance are magnified when there is contention. For example, if there is contention for DASD, transaction response times can increase.

Also, the more transactions there are in the system, the greater the processor overhead and greater the demand for both virtual and real storage.

In such situations, the system shows heavy use of *all* its resources. However, the system is actually experiencing normal system stress, and this might be hiding the cause of a performance reduction. To find the cause of such a loss of performance, you must consider all items that might be affecting your active tasks.

Investigating the overall system

Within MQSeries, the performance problem is either reduced response time or an unexpected and unexplained heavy use of resources. You should first check factors such as total processor usage, DASD activity, and paging. An IBM tool for this is resource management facility (RMF™). In general, you need to look at the system in some detail to see why tasks are progressing slowly, or why a given resource is being heavily used.

Start by looking at general task activity, then focus on particular activities, such as specific tasks or a specific time interval.

Another possibility is that the system has limited real storage; therefore, because of paging interrupts, the tasks progress more slowly than expected.

Using System Management Facility

System management facility (SMF) is an OS/390 service aid used to collect information from various OS/390 subsystems. This information is dumped and reported periodically, for example, hourly. You can use SMF with the MQSeries trace facility to collect data from MQSeries. In this way you can monitor *trends*, for example, in system utilization and performance, and collect accounting information about each user ID using the MQSeries subsystem.

You must decide how you are going to process the SMF records produced by MQSeries trace. For example, while MQSeries is running, you can use the OS/390 operator commands SETSMF or SS to alter the SMF parameters that you specified previously.

To record performance statistics (record type 115) to SMF specify the following in the SMFPRMxx member of SYS1.PARMLIB.

```
SETSMF SYS(TYPE(115))
```

To record accounting information (record type 116) to SMF specify the following in the SMFPRMxx member of SYS1.PARMLIB.

```
SETSMF SYS(TYPE(116))
```

Either PROMPT(ALL) or PROMPT(LIST) must be specified in the SMFPRMxx member if these commands are to work.

SMF must be running before you can send data to it. For more information about SMF, see the *MVS System Management Facilities (SMF)* manual.

Reporting data in SMF

You can use the SMF program IFASMFDP to dump SMF records to a sequential data set so that they can be processed.

There are several ways to report on this data, for example:

- Define service level reporter (SLR) tables that can be used to load MQSeries trace records from SMF into SLR. You can do this while loading any other SMF data into SLR.
- Write an application program to read and report information from the SMF data set. You can then tailor the report to fit your exact needs.

Allocating additional SMF buffers

When you invoke a trace, you must ensure that you allocate adequate SMF buffers. Specify SMF buffering on the VSAM BUFSP parameter of the access method services DEFINE CLUSTER statement. Specify CISZ(4096) and BUFSP(81920) on the DEFINE CLUSTER statement for each SMF VSAM data set.

If an SMF buffer shortage occurs, SMF rejects any trace records sent to it. MQSeries sends a CSQW133I message to the OS/390 operator when this occurs. MQSeries treats the error as temporary and remains active even though SMF data could be lost. When the shortage has been alleviated and trace recording has resumed, MQSeries sends a CSQW123I message to the OS/390 operator.

Using other products with MQSeries

You can use other products to help you to improve the presentation of, or to augment statistics related to, performance and accounting.

Using Resource Measurement Facility

Resource Management Facility (RMF) is an IBM licensed program (program number 5685-029) that provides system-wide information on processor utilization, I/O activity, storage, and paging. You can use RMF to monitor the utilization of physical resources across the whole system dynamically. For more information, see the *MVS Resource Measurement Facility User's Guide*.

Using Performance Reporter for OS/390

You can use Performance Reporter for OS/390 to interpret RMF and SMF records.

Performance Reporter for OS/390 is an IBM licensed program (program number 5695-101) that enables you to manage the performance of your system by collecting performance data in a DB2 database and presenting the data in a variety of formats for use in systems management. Performance Reporter can generate graphic and tabular reports using systems management data it stores in its DB2 database. It includes an administration dialog, a reporting dialog, and a log collector, all of which interact with a standard DB2 database.

This is described in the *Performance Reporter for OS/390 Administration Guide*.

Using the CICS monitoring facility

The CICS monitoring facility provides performance information about each CICS transaction running. It can be used to investigate the resources used and the time spent processing transactions. For background information, see the *CICS Performance Guide* and the *CICS Customization Guide*.

Using MQSeries trace

You can record performance statistics and accounting data for MQSeries by using the MQSeries trace facility. The data generated by MQSeries is sent to:

- The System Management Facility (SMF), specifically as SMF record type 115, subtypes 1 and 2 for the performance statistics trace
- The SMF, specifically as SMF record type 116, for the accounting trace.

If you prefer, the data generated by the MQSeries accounting trace can also be sent to the generalized trace facility (GTF).

Starting MQSeries trace

You can start the MQSeries trace facility at any time by issuing the MQSeries command, `+cpf START TRACE`.

Note: Accounting data can be lost if the accounting trace is started or stopped while applications are running. In order to collect accounting data successfully, the following conditions must apply:

- The accounting trace must be active when an application starts, and it must still be active when the application finishes.
- If the accounting trace is stopped, any accounting data collection that was active will cease.

You can also start collecting trace information automatically if you specify YES on the SMFSTAT (SMF STATISTICS) and SMFACCT (SMF ACCOUNTING) parameters of the CSQ6SYSP macro. You can also use the CSQ6SYSP macro to control the collection interval for the statistics trace with the STATIME parameter.

For details, see “Using CSQ6SYSP” on page 68 but, before invoking an MQSeries trace, read “Using System Management Facility” on page 366.

Controlling MQSeries trace

To control the MQSeries trace data collection at start up, specify values for the parameters in the CSQ6SYSP macro when you install MQSeries, see page 68.

You can control MQSeries tracing when MQSeries is running with these commands:

- `+cpf START TRACE`
- `+cpf ALTER TRACE`
- `+cpf STOP TRACE`

For information about these commands, see the *MQSeries Command Reference* manual.

Specifying trace keywords

The commands and keywords you can specify to control trace are described in the *MQSeries Command Reference* manual. When you specify a trace number, you must also specify the trace type.

Specifying a destination

The DEST keyword specifies the location to which trace data is sent. Possible destinations are:

SMF	System Management Facility
GTF	Generalized Trace Facility (accounting trace only)
SRV	Serviceability routine for diagnostic use by IBM service personnel

The default destination for trace data is SMF.

When you specify a destination with the DEST keyword, use the appropriate abbreviation. For example, DEST(SMF).

For daily monitoring, information is sent to SMF. SMF data sets usually contain information from other systems; this information is not available for reporting until the SMF data set is dumped.

Effect of trace on MQSeries performance

Using the MQSeries trace facility can have a significant effect on MQSeries and transaction performance. For example, if you start a global trace for class 1 or for all classes, it is likely to increase CPU usage and transaction response times by approximately 50%. However, if you start a global trace for classes 2 to 4 alone, or a statistics or accounting trace, the increase in CPU usage and transaction response times is likely to be less than 1%.

Using MQSeries events

MQSeries *instrumentation events* provide information about errors, warnings, and other significant occurrences in a queue manager. You can monitor the operation of all your queue managers by incorporating these events into your own system management application.

MQSeries instrumentation events fall into the following categories:

Queue manager events

These events are related to the definitions of resources within queue managers. For example, an application attempts to put a message to a queue that does not exist.

Performance events

These events are notifications that a threshold condition has been reached by a resource. For example, a queue depth limit has been reached, or the queue was not serviced within a predefined time limit.

Channel events

These events are reported by channels as a result of conditions detected during their operation. For example, when a channel instance is stopped.

Note: Channel events are not produced if you are using the “CICS mover” for distributed queuing.

When an event occurs, the queue manager puts an *event message* on the appropriate *event queue*, if defined. The event message contains information about the event that can be retrieved by a suitable MQI application.

MQSeries events can be enabled using the MQSC commands or the operations and control panels. Channel events can only be disabled by altering the definition of the event queue to PUT(DISABLED).

See the *MQSeries Programmable System Management* for information about the MQSeries events that cause event messages to be generated, and for information about the format of these messages. See the *MQSeries Command Reference* for information about enabling the events.

Chapter 23. Interpreting MQSeries performance statistics

You must process any data you collect from SMF to extract useful information. To do this, you must understand the format of the type 115 records from SMF.

SMF type 115 record layout

The standard layout for SMF records involves three parts:

<i>Part of record</i>	<i>What it is used for</i>
SMF header	Provides format, identification, and time and date information about the record itself.
Self-defining section	Defines the location and size of the individual data records within the SMF record.
Data records	The actual data from MQSeries that you want to analyze.

For more information about SMF record formats, see the *MVS System Management Facilities (SMF)* manual.

SMF type 115 record header description

Table 19 shows the format of SMF record header (SM115).

Offsets		Type	Len	Name	Description	Example
Dec	Hex					
0	(0)	Structure	28	SM115	SMF record header.	
0	(0)	Unsigned	2	SM115LEN	SMF record length.	0294
2	(2)		2		Reserved.	0000
4	(4)	Unsigned	1	SM115FLG	System indicator.	1E
5	(5)	Unsigned	1	SM115RTY	Record type. The SMF record type, for MQSeries statistics records this is always 115 (X'73').	73
6	(6)	Unsigned	4	SM115TME	Time when SMF moved record.	004C551B
10	(A)	Unsigned	4	SM115DTE	Date when SMF moved record.	0098149F
14	(E)	Character	4	SM115SID	OS/390 subsystem ID. Defines the OS/390 subsystem on which the records were collected.	D4E5F4F1 (MV41)
18	(12)	Character	4	SM115SSI	MQSeries subsystem ID.	D4D8F3F7 (MQ37)
22	(16)	Unsigned	1	SM115STF	Record subtype.	02
23	(17)		1		Reserved.	00
24	(18)		4	SM115BUF	Reserved.	00000000
28	(1C)	Character	0	SM115END	End of SMF header and start of self-defining section.	

Note: The (hexadecimal) values in the right-hand column relate to Figure 97.

Using performance statistics

Figure 97 shows an example of part of an SMF type 115 record. The numbers in the left-hand column represent the offset, in hexadecimal, from the start of the record. Each line corresponds to sixteen bytes of data, where each byte is two hexadecimal characters, for example 0C. The characters in the right-hand column represent the printable characters for each byte. Non-printable characters are shown by a period (.) character. In this example, alternate fields are underlined to help you to see them; refer to Table 19 to identify them.

```
+00000000          0294 |          .m |
+00000002  00001E73 004C551B 0098149F D4E5F4F1 | .....<...q..MV41 |
+00000012  D4D8F3F7 02000000 00000000 02700024 | MQ37..... |
+00000022  00010000 00440030 00010000 0074004C | .....< |
+00000032  00010000 00C00068 00040000 02600010 | .....{.....-.. |
+00000042  0001 | .. |
:
```

Figure 97. Part of an SMF record showing the header and self-defining sections. Alternate fields in the header are underlined. The characters highlighted in bold represent a self-defining section.

Processing type 115 SMF records

When you process the data, verify that the records are from MQSeries and that they are the records you are expecting.

Validate the values of the following fields:

- SM115RTY, the SMF record number = X'73' (115)
- SM115STF, the record subtype, must be 01 or 02

Record subtypes

MQSeries statistics can be collected for two subtypes:

- 01** System information, for example, related to the logs.
- 02** MQSeries information about number of messages, MQSeries buffer and paging information, and so on.

The subtype is specified in the SM115STF field, see Table 19 on page 371. For example, in Figure 97, the record subtype is 02. "Sample SMF statistics records" on page 382 shows example of records for both subtypes.

Where the information comes from

The information provided in the SMF records comes from specific functional components of MQSeries. These are:

MQSeries component	Function
Message manager	Processes all MQI requests.
Data manager	Manages the links between messages and queues. It calls the buffer manager to process the pages with messages on them.

Buffer manager	Manages the buffer pools in virtual storage and the writing of pages to page sets as the buffer pools become full. Also manages the reading of pages from page sets.
Log manager	Manages the writing of log records, which are essential for maintaining the integrity of the system, if there is a backout request, or for recovery, if there is a system or media failure.

Self-defining sections for type 115 records

A self-defining section of a type 115 SMF record tells you where to find a statistics record, how long it is, and how many times that type of record is repeated (with different values). The self-defining sections follow after the header, at fixed offsets from the start of the SMF record.

Four types of self-defining section are available to users for type 115 records. Each self-defining section points to statistics data related to one of four MQSeries components. Table 20 summarizes the sources of the statistics, and the offsets to the start of the SMF record header.

<i>Table 20. Offsets to self-defining sections. Offsets are from the start of the SMF record and are fixed for each type of statistics source.</i>				
Source of statistics	Record subtype (SM115STF)	Offsets		See page
		Dec	Hex	
Message manager	02	36	(X'24')	374
Data manager	02	44	(X'2C')	375
Buffer manager	02	52	(X'34')	376
Log manager	01	116	(X'74')	380
Note: Other self-defining sections refer to data for IBM use only.				

Each self-defining record is two fullwords long and has this format:

```
sssssss1111nnnn
```

where:

- sssssss** Fullword containing the offset from start of the SMF record.
- 1111** Halfword giving the length of this data record.
- nnnn** Halfword giving the number of data records in this SMF record.

For example, in Figure 97 on page 372, the self-defining section for message manager statistics is shown in bold. It is located at offset X'24' from the start of the SMF record and contains this information:

- The offset of the message manager statistics is located X'00000044' bytes from the start of the SMF record.
- The message manager record is X'0030' bytes long.
- There is one record (X'0001').

Similarly, in Figure 97, the buffer manager self-defining section at X'34' specifies that the offset to the buffer manager statistics is X'000000C0', is of length X'0068', and occurs X'0004' times.

Message manager statistics

Note: Always use offsets in the self-defining sections to locate the statistics records.

Message manager statistics

The following table shows the format of the message manager statistics record:

Table 21. Structure of the message manager statistics record QMST

Offsets		Type	Len	Name	Description
Dec	Hex				
0	(0)	Structure	48	QMST	Message manager statistics
0	(0)	Bitstring	2	QMSTID	Control block identifier
2	(2)	Unsigned	2	QMSTLL	Control block length
4	(4)	Character	4	QMSTEYEC	Control block eye catcher (QMST)
8	(8)	Signed	4	QMSTOPEN	Number of MQOPEN requests
12	(C)	Signed	4	QMSTCLOS	Number of MQCLOSE requests
16	(10)	Signed	4	QMSTGET	Number of MQGET requests
20	(14)	Signed	4	QMSTPUT	Number of MQPUT requests
24	(18)	Signed	4	QMSTPUT1	Number of MQPUT1 requests
28	(1C)	Signed	4	QMSTINQ	Number of MQINQ requests
32	(20)		4		Reserved
36	(24)	Signed	4	QMSTSET	Number of MQSET requests
40	(28)		4		Reserved
44	(2C)	Signed	4	QMSTCALH	Number of “close handle” requests

Interpretation

The data gives you counts of different MQI requests.

Data manager statistics

The following table shows the format of the data manager statistics record:

Table 22. Structure of the data manager statistics record QIST

Offsets		Type	Len	Name	Description
Dec	Hex				
0	(0)	Structure	60	QIST	Data manager statistics
0	(0)	Bitstring	2	QISTID	Control block identifier
2	(2)	Unsigned	2	QISTLL	Control block length
4	(4)	Character	4	QISTEYEC	Control block eye catcher (QIST)
8	(8)	Unsigned	4	QISTMGET	Number of message get requests
12	(C)	Unsigned	4	QISTMPUT	Number of message put requests
16	(10)		4		Reserved
20	(14)	Signed	4	QISTDCRE	Number of object create requests
24	(18)	Signed	4	QISTDPUT	Number of object put requests
28	(1C)	Signed	4	QISTDDEL	Number of object delete requests
32	(20)	Signed	4	QISTDGET	Number of object get requests
36	(24)	Signed	4	QISTDLOC	Number of object locate requests
40	(28)		4		Reserved
44	(2C)	Signed	4	QISTALST	Number of Stgclass change requests
48	(30)		4		Reserved
52	(34)		4		Reserved
56	(38)		4		Reserved
60	(3C)		4		Reserved
64	(40)		4		Reserved
68	(44)		4		Reserved
72	(48)		4		Reserved

Interpretation

The data gives you counts of different object requests.

Buffer manager statistics

The following table shows the format of the buffer manager statistics record.

Note: If you have defined a buffer pool, but not used it, no values are set so the buffer manager statistics record will not contain any data.

Table 23. Structure of the buffer manager statistics record QPST

Offsets		Type	Len	Name	Description
Dec	Hex				
0	(0)	Structure	104	QPST	Buffer manager statistics.
0	(0)	Bitstring	2	QPSTID	Control block identifier.
2	(2)	Unsigned	2	QPSTLL	Control block length.
4	(4)	Character	4	QPSTEYEC	Control block eye catcher (QPST).
8	(8)	Signed	4	QPSTPOOL	Buffer pool identifier (0000-0003).
12	(C)	Signed	4	QPSTNBUF	Number of buffers in this buffer pool.
16	(10)	Signed	4	QPSTCBSL	Lowest number of available buffers.
20	(14)	Signed	4	QPSTCBS	Number of available buffers.
24	(18)	Signed	4	QPSTGETP	The number of page get requests where the current page contents are required. This might involve a read DASD operation if the page is not currently in the buffer pool.
28	(1C)	Signed	4	QPSTGETN	The number of get requests for a new - or empty - page (that is, no read operation is necessary).
32	(20)	Signed	4	QPSTRIO	The number of page read DASD operations.
36	(24)	Signed	4	QPSTSTW	The number of page updates.
40	(28)	Signed	4	QPSTTPW	Number of pages written to DASD.
44	(2C)	Signed	4	QPSTWIO	The number of page write operations.
48	(30)	Signed	4	QPSTIMW	The number of synchronous page write operations.
52	(34)	Signed	4	QPSTDWT	The number of times the asynchronous write processor was started.
56	(38)	Signed	4	QPSTDMC	The number of times the synchronous page processor was started because the synchronous write threshold was reached.
60	(3C)	Signed	4	QPSTSTL	The number of times a page get request did not find the page already in the buffer pool.
64	(40)	Signed	4	QPSTSTLA	Number of times the hash chain has been changed during a buffer steal.
68	(44)	Signed	4	QPSTSOS	The number of times NO available buffers were found.
72	(48)		32		Reserved.

Interpreting buffer manager statistics

The buffer manager is the component of MQSeries that handles the movement of data between DASD and virtual storage.

Buffer pools are areas of MQSeries virtual storage reserved to satisfy the buffering requirements for MQSeries queues. Each buffer pool contains an installation defined number of 4 KB virtual storage pages or buffers. Page sets are VSAM linear data sets and each page set is associated with a buffer pool. Queues are mapped to page sets via their storage class attribute. For more information on the relationship between these entities, see “Storage classes - mapping queues to page sets” on page 15.

A buffer pool can be the holding storage for many message queues. To be able to estimate the required size of the buffer pools, you must understand their characteristics and how to interpret the buffer manager statistics generated by MQSeries.

A buffer pool can hold MQSeries objects, as well as messages, in 4 KB virtual storage pages. MQSeries is designed to keep these pages in buffer pool virtual storage as long as possible in order to obtain the best performance.

However, if a buffer pool starts to fill up, updated pages are written out to their relevant DASD page sets to free up buffer pool space. This happens if, for example, messages are being put onto queues associated with the buffer faster than they are being taken off.

Information contained in pages that have been written out to DASD page sets can be read in again on demand.

Nonpersistent messages can also be written out to DASD if the buffer pools in which they reside become short on storage. This is because the page sets are the backing store for the buffer pools during MQSeries operation.

Ideally, a transaction pattern should be such that messages do not spend a long time on a queue waiting to be retrieved. This means that messages never have to spill over to DASD because the pages used to hold them remain in virtual storage.

Buffer pool management

To manage your buffer pools efficiently, you must consider the factors that affect the buffer pool I/O operations and also the statistics associated with the buffer pools.

DASD operations

The following factors affect buffer pool I/O operations.

- If a page containing the required data is not found in the buffer pool, it is read in synchronously to an available buffer from its DASD page set.
- Whenever a page is updated, it is put on an internal queue of pages to be (potentially) written out to DASD. This means that the buffer used by that page is unavailable for use by any other page until the buffer has been written to DASD.

Buffer manager statistics

- If the number of pages queued to be written to DASD exceeds 85% of the total number of buffers in the pool, an asynchronous write processor is started in order to put the buffers to DASD.

Similarly, should the number of buffers available for page get requests become less than 15% of the total number of buffers in the pool, then the asynchronous write processor is started in order to perform the write I/O operations.

The write processor stops when the number of pages queued to be written to DASD has fallen to 75% of the total number of buffer in the pool.

- If the number of pages queued for writing to DASD exceed 95% of the total number of buffers in the pool, all updates result in a synchronous write of the page to DASD.

Similarly, if the number of buffers available for page get requests becomes less than 5% of the total number of buffers in the pool, all updates result in a synchronous write of the page to DASD.

- If the number of buffers available for page get requests ever reaches zero, a transaction that encounters this condition is suspended until the asynchronous write processor has finished.
- If a page is frequently updated, the page spends most of its time on the queue of pages waiting to be written to DASD. Because this queue is in least recently used order, it is possible that a frequently updated page placed on this least recently used queue will never be written out to DASD. For this reason, at the time of update, if the page is found to have been waiting on the write to DASD queue for at least 2 checkpoints, it will be synchronously written to DASD. Updating occurs at checkpoint time.

The aim of the above algorithm is to maximize the time pages spend in buffer pool memory while allowing the system to function should system load put the buffer pool usage under stress.

Buffer pool statistics

MQSeries writes statistics to SMF for each buffer pool, if statistics trace gathering has been requested. Table 23 on page 376 shows the statistics that are recorded. The statistics are reset each time they are output.

The values of these fields can be used to improve the performance of your system. These are the important factors:

- The ratio of QPSTRIO to QPSTGETP shows the efficiency of page retrieval within the buffer pool. Increasing the buffer pool size should decrease this ratio and, therefore, increase the page retrieval efficiency.

If this does not happen, it indicates that pages are not being frequently reaccessed. This implies a transaction pattern where there is a long delay between messages being put and then subsequently retrieved.

- The ratio of QPSTGETN to QPSTGETP indicates the number of times an empty page, as opposed to a non-empty page, has been requested.

This ratio is more an indication of transaction pattern, than a value that can be used to tune the system.

- QPSTSTL is a count of the number of times a page access request did not find the page already in the buffer pool. Again, the lower the ratio of QPSTSTL to (QPSTGETP + QPSTGETN) is, the higher the page retrieval efficiency.

Increasing the buffer pool size should decrease this ratio but, if it does not, it is an indication that there are long delays between puts and gets.

- The higher the ratio of QPSTSTW to QPSTWIO, the better the efficiency of the asynchronous write processor. You can increase this ratio, and therefore the efficiency of the asynchronous write processor, by increasing the buffer pool size.
- QPSTIMW is a count of the number of times pages were written out synchronously. If QPSTDMC is zero, QPSTIMW is the number of times pages were found on the queue waiting for write I/O that had been there for at least two checkpoints.
- QPSTDWT is the number of times the asynchronous write processor was started because there was either more than 85% of the pages in the buffer pool waiting for write I/O, or there was less than 15% of the buffer pool available for read requests. Increasing the buffer pool size should reduce this value. If it does not, the pattern of access is one of long delays between puts and gets.
- QPSTDMC is the number of updates that were performed synchronously because there was either more than 95% of the pages in the buffer pool waiting for write I/O, or there was less than 5% of the buffer pool available for read requests. If this number is not zero, the buffer pool might be too small and should be enlarged. If increasing the buffer pool size does not reduce QPSTDMC to zero, there might be I/O contention on the DASD page sets.
- QPSTSOS is the number of times that there were no buffers available for page get requests. If QPSTSOS ever becomes non-zero, it shows that MQSeries is under severe stress. The buffer pool size should be significantly increased. If increasing the buffer pool size does not make the value of QPSTSOS zero, there might be I/O contention on the DASD page sets.

Log manager statistics

The following table shows the format of the log manager statistics record:

Table 24. Structure of the log manager statistics record QJST

Offsets		Type	Len	Name	Description
Dec	Hex				
0	(0)	Structure	64	QJST	Log manager statistics.
0	(0)	Character	2	QJSTID	Control block identifier.
2	(2)	Signed	2	QJSTLL	Control block length.
4	(4)	Character	4	QJSTEID	Control block eye catcher (QJST).
8	(8)	Signed	4	QJSTWRW	Write_request count - Wait. Tasks are suspended until the write to active log is complete.
12	(C)	Signed	4	QJSTWRNW	Write_request count - No wait. Asynchronous writes to the active log. Tasks are not suspended.
16	(10)	Signed	4	QJSTWRF	Write_request count - Force. Tasks are suspended until all the log records for this unit of recovery are written to the active log data set.
20	(14)	Signed	4	QJSTWTB	Wait count for unavailable buffers. Number of times a task was suspended because all the buffers were waiting to be written to the active log data set.
24	(18)	Signed	4	QJSTRBUF	Number of read log requests satisfied from in-storage buffers.
28	(1C)	Signed	4	QJSTRACT	Number of read log requests satisfied from the active log data set.
32	(20)	Signed	4	QJSTRARH	Number of read log requests satisfied from an archive log data set.
36	(24)	Signed	4	QJSTWTL	Number of read log requests delayed because the MAXALLC parameter in CSQ6LOGP limited the number of archive log data sets that could be used.
40	(28)	Signed	4	QJSTBSDS	Total number of bootstrap data set (BSDS) access requests.
44	(2C)	Signed	4	QJSTBFFL	The number of active log control intervals (CIs) created (log pages used).
48	(30)	Signed	4	QJSTBFWR	Number of calls made that wrote to active log buffers.
52	(34)	Signed	4	QJSTALR	Number of times an archive log data set was allocated for a read request.
56	(38)	Signed	4	QJSTALW	Number of times an archive log data set was allocated for a write request.
60	(3C)	Signed	4	QJSTCIOF	Count of CIs off-loaded to the archive data set.
64	(40)	Signed	4	QJSTLLCP	Number of times that checkpoint was invoked.

Interpreting log manager statistics

These counts are important:

1. The total number of log write requests:

$$N_{\text{logwrite}} = \text{QJSTWRW} + \text{QJSTWRNW} + \text{QJSTWRF}$$

2. The total number of log read requests:

$$N_{\text{logread}} = \text{QJSTRBUF} + \text{QJSTRACT} + \text{QJSTRARCH}$$

The problem symptoms that can be examined using log manager statistics are described in Table 25.

<i>Table 25 (Page 1 of 2). Problem symptoms that can be examined using log manager statistics</i>	
Symptom 1	QJSTWTB is non-zero.
Reason	Tasks are being suspended while the in-storage buffer is being written to the active log. There might be problems writing to the active log. The OUTBUFF parameter within CSQ6LOGP is too small.
Action	Investigate the problems writing to the active log. Increase the value of the OUTBUFF parameter within CSQ6LOGP.
Symptom 2	The ratio: $\text{QJSTARH}/N_{\text{logread}}$ is larger than normal.
Reason	Most log read requests should come from the output buffer or the active log. To satisfy requests for back out, unit-of-recovery records are read from the in-storage buffer, the active log, and the archived logs. A long-running unit of recovery, extending over a period of many minutes, might have log records spread across many different logs. This degrades performance because extra work has to be done to recover the log records.
Action	Change the application to reduce the length of a unit of recovery. Also, consider increasing the size of the active log to reduce the possibility of a single unit of recovery being spread out over more than one log.
Other pointers	The ratio $N_{\text{logread}}/N_{\text{logwrite}}$ gives an indication of how much work has to be backed out.
Symptom 3	The ratio: $\text{QJSTWL}/N_{\text{logread}}$ is greater than 1%.
Reason	Log reads were initiated that had to read from an archive log, but MQSeries was not able to allocate a data set because MAXALLC data sets were already allocated.
Action	Increase MAXALLC.

Sample SMF statistics records

Table 25 (Page 2 of 2). Problem symptoms that can be examined using log manager statistics

Symptom 4	QJSTLLCP is more than 10 per hour.
Reason	<p>On a busy system you would expect to see typically 10 checkpoints an hour. If the QJSTLLCP value is larger than this, it indicates a problem in the setup of the queue manager.</p> <p>The most likely reason for this is that the LOGLOAD parameter in CSQ6SYSP is too small. The other event that causes a checkpoint is when an active log fills up and switches to the next active log data set. If your logs are too small, this can cause frequent checkpoints.</p>
Action	Increase the LOGLOAD parameter, or increase the size of your log data sets as required.

Note: In the first set of statistics produced after system startup, there might be significant log activity due to the resolution of in-flight units of recovery.

Sample SMF statistics records

Figure 98 shows the SMF record for subtype 1. Subtype 1 includes the log manager statistics record. In this figure, the SMF record header, the self-defining section, and the statistics record are underlined.

In this example, the self-defining section, at offset X'74', refers to log manager statistics, see Table 24 on page 380. The statistics are located at offset X'00000164' from the start of the header and is X'40' bytes long. There is only one set of statistics identified by the eye catcher string QJST.

Figure 99 on page 384 shows the SMF record for subtype 2. Subtype 2 includes the buffer manager statistics. The SMF record header, the self-defining section, and the buffer manager statistics record are shown underlined. In this example, the self-defining section, at offset X'34', refers to buffer manager statistics (see Table 23 on page 376). The statistics are located at offset X'000000B0' from the start of the header and is X'68' bytes long.

There are four sets of statistics corresponding the four buffer pools numbered 0, 1, 2, and 3 respectively. Each set of data related to the buffer manager can be identified by the eye catcher string QPST.

The eye catcher strings are:

QIST	Data manager
QJST	Log manager (subtype 1, see Figure 98 on page 383)
QMST	Message manager
QPST	Buffer manager

00000000.	<u>01C80000</u>	<u>1E730030</u>	<u>C47A0093</u>	<u>238FD4E5</u>	.H.....D:.1..MV
00000010.	<u>E2D4E5C3</u>	<u>E3F8010C</u>	<u>00000000</u>	<u>00000000</u>	SMVCT8.....
0000001C.				000001A4u
00000020.	00240001	00000000	00000000	00000000
00000030.	00000000	00000000	00000000	0000007C@
00000040.	00400001	000000BC	00600001	00000000-.....
00000050.	00000000	00000000	00000000	00000000
00000060.	00000000	0000011C	00480001	00000000
00000070.	00000000	<u>00000164</u>	<u>00400001</u>	<u>00000000</u>
0000007C.				00002086f
00000080.	00000000	00002086	000040FA	00000000f..
00000090.	00000000	00000000	00000005	00002078
000000A0.	0000207E	00000000	00000005	00000000	...=.....
000000B0.	0000208D	00000000	00000000	
000000BC.				00000000
000000C0.	00000000	00000000	00000000	00000000
000000D0.	00000000	00000003	00000001	00000000
000000E0.	00000001	00000000	00000000	00000001
000000F0.	00000000	00000207	00000000	00000000
00000100.	00000000	00000000	00000000	00000000
0000011C.	00000000	00000000	00000000	
00000110.				003C0048
00000120.	D8E2E2E3	0000209E	00002080	00000002	QSST.....
00000130.	00000114	00000085	0000054A	00000536e..¢...
00000140.	00000575	0000088C	00000011	00004D64(
00000150.	00004CE7	00000000	00000000	00000000	..[X.....
00000160.	00000000			
00000164.		<u>00930044</u>	<u>D8D1E2E3</u>	<u>00000000</u>	.1. QJST....
00000170.	<u>000059BA</u>	<u>00000883</u>	<u>00000000</u>	<u>00000EED</u>c.....
00000180.	<u>0000009D</u>	<u>00000000</u>	<u>00000000</u>	<u>00000068</u>
00000190.	<u>00000218</u>	<u>000011E9</u>	<u>00000000</u>	<u>00000000</u>Z.....
000001A0.	<u>00000000</u>	<u>00000002</u>	<u>00000000</u>	<u>00000000</u>
000001A8.			0024011A	00010C10
000001B0.	0F049158	E5C3E3F8	A7FDDCC5	D3C24101	..j.VCT8x..ELB..
000001C0.	00000001	00000005	00000002	

Figure 98. SMF record 115, subtype 1. The subscripts identify: (1) the SMF record header, (2) the self-defining section, and (3) the log manager statistics record.

Sample SMF statistics records

+00000000				<u>0294</u>				.m
+00000002	<u>00001E73</u>	<u>004C551B</u>	<u>0098149F</u>	<u>D4E5F4F1</u>	<...q..MV41		
+00000012	<u>D4D8F3F7</u>	<u>02000000</u>	<u>00000000</u>	<u>02700024</u>		MQ37.....		
+00000022	00010000	00440030	00010000	0074004C	<		
+00000032	00010000	<u>00C00068</u>	<u>00040000</u>	<u>02600010</u>	{.....-		
+00000042	0001					..		
+00000044		D40F	0030D8D4	E2E30000		M...QMST.....		
+00000052	00060000	000D0000	000E0000	00000000			
+00000062	00000000	00000000	00000000	00000000			
+00000072	0000					..		
+00000074		C90F	004CD8C9	E2E30000		I..<QIST.....		
+00000082	000E0000	00000000	00000000	00000000			
+00000092	00000000	001A0000	000E0000	00000000			
+000000A2	00000000	00000000	00000000	00000000			
+000000B2	00000000	00000000	00000000	0003			
+000000C0				<u>D70F</u>				P.
+000000C2	<u>0068D8D7</u>	<u>E2E30000</u>	<u>00000000</u>	<u>07D00000</u>		..QPST.....}		..
+000000D2	<u>07CD0000</u>	<u>07CD0000</u>	<u>00210000</u>	<u>00000000</u>			
+000000E2	<u>00000000</u>	<u>00210000</u>	<u>00000000</u>	<u>00000000</u>			
+000000F2	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>			
+00000102	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>			
+00000112	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>			
+00000122	<u>00000000</u>	<u>0000</u>					
+00000128			D70F	0068D8D7		P...QPST..		
+00000132	00010000	07D00000	07CA0000	07CA0000	}		
+00000142	00300000	00040000	00000000	00290000			
+00000152	00000000	00000000	00000000	00000000			
+00000162	00000000	00030000	00000000	00000000			
+00000172	00000000	00000000	00000000	00000000			
+00000182	00000000	00000000	00000000	0000			
+00000190				<u>D70F</u>				P.
+00000192	<u>0068D8D7</u>	<u>E2E30000</u>	<u>00020000</u>	<u>041A0000</u>		..QPST.....		
+000001A2	<u>041A0000</u>	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>			
+000001B2	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>			
+000001C2	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>			
+000001D2	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>			
+000001E2	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>			
+000001F2	<u>00000000</u>	<u>0000</u>					
+000001F8			D70F	0068D8D7		P...QPST..		
+00000202	00030000	041A0000	041A0000	00000000			
+00000212	00000000	00000000	00000000	00000000			
+00000222	00000000	00000000	00000000	00000000			
+00000232	00000000	00000000	00000000	00000000			
+00000242	00000000	00000000	00000000	00000000			
+00000252	00000000	00000000	00000000	0000			
+00000260				<u>D30F</u>				L.
+00000262	0010D8D3	E2E30000	00000000	0000		..QLST.....		

Figure 99. SMF record 115, subtype 2. The underlined sections indicate the SMF record header, the self-defining section, and a buffer manager statistics record.

Chapter 24. Interpreting MQSeries accounting data

You must process any accounting data you collect from SMF to extract useful information. To do this, you must understand the format of the type 116 records from SMF.

SMF type 116 record layout

The standard layout for SMF records involves three parts:

<i>Part of record</i>	<i>What it is used for</i>
SMF header	Provides format, identification, and time and date information about the record itself.
Self-defining section	Defines the location and size of the individual data records within the SMF record.
Data records	The actual data from MQSeries that you want to analyze.

For more information about SMF record formats, see the *MVS System Management Facilities (SMF)* manual.

SMF type 116 record header description

Table 26 shows the format of SMF record header (SM116).

Offsets		Type	Len	Name	Description	Example
Dec	Hex					
0	(0)	Structure	28	SM116	SMF record header.	
0	(0)	Unsigned	2	SM116LEN	SMF record length.	01A4
2	(2)		2		Reserved.	0000
4	(4)	Unsigned	1	SM116FLG	System indicator.	1E
5	(5)	Unsigned	1	SM116RTY	Record type. The SMF record type, for MQSeries accounting records this is always 116 (X'74').	74
6	(6)	Unsigned	4	SM116TME	Time when SMF moved record.	005BD9BD
10	(A)	Unsigned	4	SM116DTE	Date when SMF moved record.	0094088F
14	(E)	Character	4	SM116SID	OS/390 subsystem ID. Defines the OS/390 subsystem on which the records were collected.	D7D4C5E2 (PMES)
18	(12)	Character	4	SM116SSI	MQSeries subsystem ID.	E5C3F140 (VC1)
22	(16)	Unsigned	1	SM116STF	Record subtype.	00
23	(17)		1		Reserved.	00
24	(18)		4	SM116SEQ	Reserved.	00000000
24	(18)		4	SM116BUF	Reserved.	00000000

Using accounting data

Table 26 (Page 2 of 2). SMF record header description						
Offsets		Type	Len	Name	Description	Example
Dec	Hex					
28	(1C)	Character	0	SM116END	End of SMF header and start of self-defining section.	

Note: The (hexadecimal) values in the right-hand column relate to Figure 100.

Figure 100 shows an example of part of an SMF type 116 record. The numbers in the left-hand column represent the offset, in hexadecimal, from the start of the record. Each line corresponds to sixteen bytes of data, where each byte is two hexadecimal characters, for example 0C. The characters in the right-hand column represent the printable characters for each byte. Non-printable characters are shown by a period (.) character. In this example, alternate fields are underlined to help you to see them; refer to Table 26 to identify them.

00000000.	<u>01A40000</u>	<u>1E74005B</u>	D9BD0094	088FD7D4	.u.....\$R..m..PM
00000010.	C5E2 <u>E5C3</u>	<u>F1400000</u>	00000000	00000134	ESVC1
00000020.	00700001	00000054	00B00001	00000104
00000030.	00300001	00000000	00000000	00000000
00000040.	00000000			

Figure 100. Part of an SMF record showing the header and self-defining sections. Alternate fields in the header are underlined. The characters highlighted in bold represent a self-defining section.

Processing type 116 SMF records

When you process the data, verify that the records are from MQSeries and that they are the records you are expecting.

Validate the value of the following fields:

- SM116RTY, the SMF record number = X'74' (116)
- SM116STF, the record subtype, must be 00

Where the information comes from

The information provided in the SMF records comes from the message manager component of MQSeries. This component processes all MQI requests.

Self-defining sections

A self-defining section of an SMF record tells you where to find an accounting record, how long it is, and how many times that type of record is repeated (with different values). The self-defining sections follow the header, at a fixed offset from the start of the SMF record.

Two types of self-defining section are available to users for type 116 records. Each self-defining section points to accounting related data. Table 27 summarizes the offsets from the start of the SMF record header.

<i>Table 27. Offsets to self-defining sections. Offsets are from the start of the SMF record and are fixed for each type of accounting source.</i>			
Source of accounting data	Offsets		See...
	Dec	Hex	
Message manager	28	(X'1C')	Table 28 on page 388
Message manager	44	(X'2C')	Table 29 on page 388
Note: Other self-defining sections refer to data for IBM use only.			

Each self-defining record is two fullwords long and has this format:

```
sssssss1111nnnn
```

where:

sssssss Fullword containing the offset from start of the SMF record.

1111 Halfword giving the length of this data record.

nnnn Halfword giving the number of data records in this SMF record.

For example, in Figure 100, the self-defining section for the type of message manager accounting data is shown in bold. It is located at offset X'2C' from the start of the SMF record and contains this information:

- The offset of the message manager accounting data is located X'00000104' bytes from the start of the SMF record.
- This message manager record is X'0030' bytes long.
- There is one record (X'0001').

Note: Always use offsets in the self-defining sections to locate the accounting records.

Message manager accounting

The following tables show the format of the message manager accounting records:

Table 28. Structure of the message manager accounting record QWHS

Offsets		Type	Len	Name	Description
Dec	Hex				
0	(0)	Structure	112	QWHS	Message manager accounting data.
0	(0)		12		Reserved.
12	(0C)	Character	4	QWHSSID	Subsystem name.
16	(10)		24		Reserved.
40	(28)	Character	8	QWHCAID	User ID associated with the OS/390 job.
48	(30)	Character	12	QWHCCV	Thread cross reference (see "Thread cross reference data" on page 389).
60	(3C)	Character	8	QWHCCN	Connection name.
68	(44)		8		Reserved.
76	(4C)	Character	8	QWHCOPID	User ID associated with the transaction.
84	(54)	Signed	4	QWHCATYP	Type of connecting system (1=CICS, 2=Batch or TSO, 3=IMS control region, 4=IMS MPP or BMP, 5=Command server, 6=Channel initiator, 7=RRS Batch).
88	(58)	Character	22	QWHCTOKN	Accounting token set to the OS/390 accounting information for the user.
110	(6E)		2		Reserved.

Table 29 (Page 1 of 2). Structure of the message manager accounting record QMAC

Offsets		Type	Len	Name	Description
Dec	Hex				
0	(0)	Structure	48	QMAC	Message manager accounting data.
0	(0)	Bitstring	2	QMACID	Control block identifier.
2	(2)	Unsigned	2	QMACLL	Control block length.
4	(4)	Character	4	QMACYEC	Control block eye catcher (QMAC).
8	(8)	Character	8	QMACCPUT	CPU time used (TOD format).
16	(10)	Signed	4	QMACPUTA	Number of MQPUT requests for messages of length 0 through 99 bytes.
20	(14)	Signed	4	QMACPUTB	Number of MQPUT requests for messages of length 100 through 999 bytes.
24	(18)	Signed	4	QMACPUTC	Number of MQPUT requests for messages of length 1000 through 9999 bytes.
28	(1C)	Signed	4	QMACPUTD	Number of MQPUT requests for messages of length greater than or equal to 10000 bytes.
32	(20)	Signed	4	QMACGETA	Number of MQGET requests for messages of length 0 through 99 bytes.
36	(24)	Signed	4	QMACGETB	Number of MQGET requests for messages of length 100 through 999 bytes.

Table 29 (Page 2 of 2). Structure of the message manager accounting record QMAC

Offsets		Type	Len	Name	Description
Dec	Hex				
40	(28)	Signed	4	QMACGETC	Number of MQGET requests for messages of length 1000 through 9999 bytes.
44	(2C)	Signed	4	QMACGETD	Number of MQGET requests for messages of length greater than or equal to 10000 bytes.

Interpretation

The QMAC data gives you information about the CPU time spent processing MQSeries calls, and counts of the number of **MQPUT** and **MQGET** requests for messages of different sizes.

The QWHC data gives you information about the user (for example, the user ID (QWHCAID) and the type of application (QWHCATYP)).

Records containing zero CPU time

Records are sometimes produced that contain zero CPU time in the QMACCPUT field. These records occur when long running TCBs identified to MQSeries either terminate or are prompted to output accounting records by accounting trace being stopped. Such TCBs exist in the CICS adapter and in the channel initiator (for distributed queuing without CICS). The number of these TCBs with zero CPU time depends upon how much activity there has been in the system:

- For the CICS adapter, this can result in up to nine records with zero CPU time.
- For the channel initiator, the number of records with zero CPU time can be up to the sum of Adapters + Dispatchers + 6, as defined in the channel initiator parameters.

These records reflect the amount of work done under the TCB, and can be ignored.

Thread cross reference data

The interpretation of the data in the thread cross reference (QWHCCV) field varies. This depends on what the data relates to:

- CICS (QWHCATYP=1) – see Table 30
- IMS (QWHCATYP=3 or 4) – see Table 31 on page 390
- Batch, TSO, or RRS Batch (QWHCATYP=2 or 7) – this field consists of binary zeros
- Others – no meaningful data

Table 30. Structure of the thread cross reference record for a CICS system

Offsets		Type	Len	Name	Description
Dec	Hex				
48	(30)	Character	4	QWHCTNO	CICS thread number.
52	(34)	Character	4	QWHCTRN	CICS transaction name.
56	(38)	Signed	4	QWHCTASK	CICS task number.

Sample SMF accounting record

Some entries contain blank characters. These apply to the TCB, rather than to a specific transaction.

Offsets		Type	Len	Name	Description
Dec	Hex				
48	(30)	Character	4	QWHCPST	IMS partition specification table (PST) region identifier.
52	(34)	Character	8	QWHCPSB	IMS program specification block (PSB) name.

Special considerations when using IMS accounting records

A single IMS application might write two SMF records. In this case, the figures from both records should be added to provide the correct totals for the IMS application.

Sample SMF accounting record

Figure 101 shows a type 116 SMF record. In this figure, the SMF record header, the self-defining section, and the accounting data record are underlined.

00000000.	<u>01A40000</u>	<u>1E74005B</u>	<u>D9BD0094</u>	<u>088FD7D4</u>	.u.....\$R..m..PM
00000010.	<u>C5E2E5C3</u>	<u>F1400000</u>	<u>00000000</u>	<u>00000000</u> ₁	ESVC1
0000001C.				00000134
00000020.	00700001	00000054	00B00001	<u>00000104</u>
00000030.	<u>00300001</u>	00000000	00000000	00000000
00000040.	00000000	00000000	00000000	00000000
00000050.	00000000	00000000	00000000	00000000
00000054.		A90CA45A	76B71A04	A90CA45C	z.u!....z.u*
00000060.	0F95CE03	00000000	0BE78EC5	00000000	.n.....X.E....
00000070.	2E052997	00000000	01F3BD00	00000000	...p....3.....
00000080.	02577700	0000000C	40404040	40404040
00000090.	00000000	00000000	00000001	00000000
000000A0.	00000000	00000000	00000000	00000000
000000B0.	00000000	00000000	00000000	00000000
000000C0.	00000000	00000000	00000000	00000000
000000D0.	00000000	00000000	00000000	00000000
000000E0.	00000000	00000000	00000000	00000000
000000F0.	00000000	00000000	00000000	00000000
00000100.	00000000			
00000104.		<u>D4140030</u>	<u>D8D4C1C3</u>	<u>00000000</u>	M...QMAC....
00000110.	<u>1D7AF539</u>	<u>0000000A</u>	<u>0000000A</u>	<u>0000000A</u>	.:5.....
00000120.	<u>0000000A</u>	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>
00000130.	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>	<u>00000000</u>
00000134.		0024011A	00030710	02DAACF00
00000140.	E5C3F140	A90CA45C	0FC24E03	00000003	VC1 z.u*.B+....
00000150.	00000003	00000001	004C0200	C3D6D5F0<..CON0
00000160.	F3F4F040	00000000	00000000	00000000	340
00000170.	D7C1E4D3	D7E4E340	00000000	00000000	PAULPUT
00000180.	C3D6D5F0	F3F4F040	00000002	00000000	CON0340
00000190.	00000000	00000000	00000000	00000000
000001A0.	00000000			

Figure 101. SMF record 116. The subscripts identify: (1) the SMF record header, (2) the self-defining section, and (3) an example of a message manager accounting record.

Part 8. Security

Chapter 25. Introduction to MQSeries security	393
Why you need to protect MQSeries resources	393
Implementing MQSeries security	394
Security controls and options	394
Resources you can protect	395
Connection security	395
API-resource security	395
Command security	397
Command resource security	397
Chapter 26. Using RACF classes and profiles	399
Using RACF security classes	400
RACF profiles	401
Switch profiles	401
How switches work	402
Profiles used to protect MQSeries resources	404
Profiles for connection security	404
Profiles for queue security	406
Profiles for processes	413
Profiles for namelists	414
Profiles for alternate user security	415
Profiles for context security	417
API-resource security access quick reference	419
Profiles for command security	421
Profiles for command resource security	422
Using the RESLEVEL security profile	425
RESLEVEL and Batch/TSO connections	425
RESLEVEL and system utilities	425
RESLEVEL and CICS connections	425
RESLEVEL and IMS connections	426
RESLEVEL and channel initiator connections	427
The RESLEVEL profile	428
Important notes on using RESLEVEL	430
User IDs for security checking	430
User IDs for connection security	430
User IDs for command security and command resource security	431
User IDs for resource security (MQOPEN and MQPUT1)	431
Blank user IDs and UACC levels	436
Auditing considerations	437
Auditing RESLEVEL	437
Statistics	438
Chapter 27. MQSeries security implementation	439
Security implementation checklist	439
MQSeries security management	441
User ID timeouts	441
User ID reverification	442
Security refreshes	442
Displaying security status	443
Customizing security	444

Security considerations for using MQSeries with CICS	445
Controlling the security of CICS transactions supplied by MQSeries	445
CICS adapter user IDs	446
Security considerations for the CICS bridge	447
Security considerations for using MQSeries with IMS	449
Using the OPERCMDS class	449
Security considerations for the IMS bridge	449
Security considerations for distributed queuing	453
Security considerations for cluster support	455
Security installation tasks	456
Setting up MQSeries data set and system security	456
Example security scenario	458
Security switch settings	458
MQSeries object definitions	459
User IDs used in scenarios	460
Security profiles and accesses required	460
Security problem determination	464
Violation messages	464
What to do if access is allowed or disallowed incorrectly	465

Chapter 25. Introduction to MQSeries security

Important

This section assumes you are using Resource Access Control Facility (RACF) as the external security manager (ESM) in your installation. Therefore, if you are using a different ESM, you must modify the techniques described here.

It also assumes that you are familiar with MQSeries concepts and terminology described in the *MQSeries Application Programming Guide*.

Where profile names are shown, replace the subsystem identifier (ssid) in the profile name with the name of the MQSeries subsystem you are using.

Why you need to protect MQSeries resources

Because MQSeries handles the transfer of information that is potentially valuable, it needs the safeguard of a security system. This is to ensure that the resources MQSeries owns and manages are protected from unauthorized access that might lead to the loss or disclosure of the information. It is essential that none of the following are accessed or changed by any unauthorized user or process:

- Connections to MQSeries
- MQSeries objects such as queues, processes, and namelists
- MQSeries transmission links
- MQSeries system control commands
- MQSeries messages
- Context information associated with messages

If you do nothing

If you do nothing about security, the most likely effect is that **all** users can access and change **every** resource. This includes not only local users, but also those on remote systems using distributed queuing or clients, where the logon security controls might be less strict than is normally the case for OS/390.

To provide the necessary security, MQSeries uses the OS/390 system authorization facility (SAF) to route authorization requests to an ESM, for example, RACF. MQSeries does no security verification of its own. Where distributed queuing or clients are being used, additional security measures might be required, for which MQSeries provides channel exits and the MCAUSER channel attribute.

The decision to allow access to an object is made by the ESM and MQSeries follows that decision. If the ESM cannot make a decision, MQSeries prevents access to the object.

Implementing MQSeries security

It is easier to set up and administer your security if you first decide on a set of naming conventions for your MQSeries objects.

To implement the security strategy for your MQSeries subsystem, you must decide:

- How security is to be used and implemented.
- Who is going to use the MQSeries system and resources.

Read through the topics listed above, noting the work that must be done before anyone uses the system. Then work through “Security implementation checklist” on page 439 to set up the security you need for each of your MQSeries subsystems. See “Security installation tasks” on page 456 to see how to tailor your security system.

To use the RACF examples as shown in this manual, you must be a suitably authorized user, for example, a SYSTEM SPECIAL user. You can enter the commands from either TSO in the RACF command form or the RACF ISPF panels.

Security controls and options

You can specify whether security is turned on for the whole MQSeries subsystem, and you can also control the number of user IDs checked for API-resource security.

Subsystem security

Subsystem security is a control that specifies whether any security checking is done on the whole MQSeries subsystem. If the security on your CICS, IMS, TSO, or batch system is adequate, you can turn off security checking for the MQSeries subsystem so that no further security checking takes place.

Note: This is the only check that affects other security checks. That is, if you turn off checking for the MQSeries subsystem, no other MQSeries checking is done; if you leave it turned on, MQSeries checks your security requirements for other MQSeries resources.

RESLEVEL

RESLEVEL is a RACF profile that controls the number of user IDs checked for MQSeries resource security. Normally, when a user attempts to access an MQSeries resource, RACF checks the relevant user ID or IDs to see if access is allowed to that resource. By defining a RESLEVEL profile you can control whether zero, one or, where applicable, two user IDs are checked.

There is only one RESLEVEL profile per queue manager. Control is implemented by the access that a user ID has to this profile.

Resources you can protect

When MQSeries starts, or when instructed by an operator command, MQSeries determines which resources you want to protect. You can control which security checks are performed for each individual queue manager. For example, you could implement a number of security checks on a production queue manager, but none on a test queue manager.

Connection security

Connection security checking is carried out either when an application program tries to connect to a queue manager by issuing an **MQCONN** request or when the CICS or IMS adapter issues a connection request. You can turn connection security checking off for a particular MQSeries subsystem, but if you do any user can connect to that subsystem.

For the CICS adapter, only the CICS address space user ID is used for the connection security check—not the individual CICS terminal user ID. For the IMS adapter, when the IMS control or dependent regions connect to MQSeries, the IMS address space user ID is checked.

API-resource security

Resources are checked when an application opens an object with an **MQOPEN** or an **MQPUT1** call. The RACF access needed to open an object depends on what open options are specified when the queue is opened.

API-resource security is subdivided into these checks:

- Queue
- Process
- Namelist
- Context
- Alternate user

No security checks are performed when opening the queue manager object.

Queue security

Queue security checking controls who is allowed to open which queue, and what options they are allowed to open it with. For example, a user might be allowed to open a queue called PAYROLL.INCREASE.SALARY to browse the messages on the queue (via the MQOO_BROWSE option), but not to remove messages from the queue (via one of the MQOO_INPUT_★ options). If you turn checking for queues off, any user can open any queue with any valid open option (that is, any valid MQOO_★ option on an **MQOPEN** or **MQPUT1** call).

Process security

Process security checking is carried out when a user opens a process definition object. If you turn checking for processes off, any user can open any process.

Namelist security

Namelist security checking is carried out when a user opens a namelist. If you turn checking for namelists off, any user can open any namelist.

Alternate user security

Alternate user security controls whether one user ID can use the authority of another user ID to open an MQSeries object.

For example:

- A server program running under user ID PAYSERV retrieves a request message from a queue that was put on the queue by user ID USER1.
- When the server program gets the request message, it processes the request and puts the reply back into the reply-to-queue specified with the request message.
- Instead of using its own user ID (PAYSERV) to authorize opening the reply-to-queue, the server can specify some other user ID, in this case, USER1. In this example, alternate user security would control whether user ID PAYSERV is allowed to specify user ID USER1 as an alternate user ID when opening the reply-to-queue.

If alternate user security is not active, any user can use any other user ID as an alternate user ID.

The alternate user ID is specified in the *AlternateUserId* field of the object descriptor (MQOD).

Note: You can use alternate user IDs on any MQSeries object, for example, processes or namelists. It does not affect the user ID used by any other resource managers, for example, for CICS security or for OS/390 data set security. See “Profiles for alternate user security” on page 415 for more information.

Context security

Context is information that is applicable to a particular message and is contained in the message descriptor (MQMD) that is part of the message. The context information comes in two sections:

Identity section The user of the application that first put the message to a queue. It consists of the following fields:

- *UserIdentifier*
- *AccountingToken*
- *ApplIdentityData*

Origin section The application that put the message on the queue where it is currently stored. It consists of the following fields:

- *PutApplType*
- *PutApplName*
- *PutDate*
- *PutTime*
- *ApplOriginData*

Applications can specify the context data when either an **MQOPEN** or an **MQPUT1** call is made. This data might be generated by the application, it might be passed

on from another message, or it might be generated by the queue manager by default. For example, context data can be used by server programs to check the identity of the requester, that is, did this message come from the correct application? Typically, the *UserIdentifier* field is used to determine the user ID of an alternate user.

You use context security to control whether the user can specify any of the context options on any **MQOPEN** or **MQPUT1** call. For information about the context options, see the *MQSeries Application Programming Guide*; for descriptions of the message descriptor fields relating to context, see the *MQSeries Application Programming Reference* manual. If you turn context security checking off, any user can use any of the context options that the queue security allows.

See “Profiles for context security” on page 417 for more information.

Command security

Command security checking is carried out when a user issues an MQSeries *command* from any of the sources described in “Managing MQSeries with commands and utilities” on page 18. A separate check can be made on the resource specified by the command as described in “Command resource security.”

If you turn off command checking, issuers of commands are not checked to see whether they have the authority to issue the command.

If MQSeries commands are entered from a console, the console must have the OS/390 SYS console authority attribute. Commands that are issued from the CSQINP1 or CSQINP2 data sets, or internally by the queue manager, are exempt from all security checking while those for CSQINPX use the user ID of the channel initiator address space. You should control who is allowed to update these data sets through normal data set protection. See “Profiles for command security” on page 421 for more information.

Security checking for the operations and control panels

If you are going to use the operations and control panels, you must have the appropriate authority to issue the commands corresponding to the actions that you choose. In addition, you must have READ access to all the *ssid.DISPLAY.object* profiles in the MQCMDS class (see Table 42 on page 423) because the panels use the various DISPLAY commands to gather the information that they present.

Command resource security

Some MQSeries commands, for example defining a local queue, involve the manipulation of MQSeries resources. When command resource security is active, each time a command involving a resource is issued, MQSeries checks to see if the user is allowed to change the definition of that resource.

You can use command resource security to help enforce naming standards. For example, a payroll administrator might be allowed to delete and define only queues with names beginning “PAYROLL”. If command resource security is inactive, no security checks are made on the resource that is being manipulated by the command. Do not confuse command resource security with command security; the two are independent.

Resource protection

Note: Turning off command resource security checking does not affect the resource checking that is done specifically for other types of processing that do not involve commands.

See “Profiles for command resource security” on page 422 for more information.

Chapter 26. Using RACF classes and profiles

This chapter discusses the following subjects:

- “Using RACF security classes” on page 400
- “RACF profiles” on page 401
- “Switch profiles” on page 401
- “Profiles used to protect MQSeries resources” on page 404
- “Profiles for command security” on page 421
- “Profiles for command resource security” on page 422
- “Using the RESLEVEL security profile” on page 425
- “User IDs for security checking” on page 430
- “Auditing considerations” on page 437

Using RACF security classes

RACF classes are used to hold the profiles required for MQSeries security checking. Each RACF class holds one or more profiles used at some point in the checking sequence, as shown in Table 32.

Member class	Group class	Contents
MQADMIN	GMQADMIN	Profiles: Used mainly for holding profiles for administration-type functions. For example: <ul style="list-style-type: none"> • Profiles for MQSeries security switches • The RESLEVEL security profile • Profiles for alternate user security • The context security profile • Profiles for command resource security
MQCONN		Profiles used for connection security
MQCMD5		Profiles used for command security
MQQUEUE	GMQQUEUE	Profiles used in queue resource security
MQPROC	GMQPROC	Profiles used in process resource security
MQQLIST	GMQQLIST	Profiles used in namelist resource security

Some classes have a related *group class* that enables you to put together groups of resources that have similar access requirements. For details about the difference between the member and group classes and when to use a member or group class, see the *Security Server (RACF) Security Administrator's Guide*.

The classes must be activated before security checks can be made. To activate all of the MQSeries classes, you use can use this RACF command:

```
SETROPTS CLASSACT(MQADMIN,MQQUEUE,MQPROC,MQQLIST,MQCONN,MQCMD5)
```

You should also ensure that you set up the classes so that they can accept generic profiles. You also do this with the RACF command SETROPTS, for example:

```
SETROPTS GENERIC(MQADMIN,MQQUEUE,MQPROC,MQQLIST,MQCONN,MQCMD5)
```

RACF profiles

All RACF profiles used by MQSeries are prefixed with the name of the subsystem that they are to be used by. For example, if queue manager STCD has a queue called QUEUE_FOR_LOST_CARD_LIST, the appropriate profile would be defined to RACF as:

```
RDEFINE MQQUEUE STCD.QUEUE_FOR_LOST_CARD_LIST
```

This means that different MQSeries subsystems sharing the same RACF database can have different security options. The subsystem ID in the profile cannot be generic.

Note: MQSeries allows the use of the percent character (%) in object names. However, RACF uses the % character as a single-character wild card. This means that when you define an object name with a % character in its name, you must consider this when you define the corresponding profile.

For example, for the queue CREDIT_CARD_%_RATE_INQUIRY, on queue manager CRDP, the profile would be defined to RACF as follows:

```
RDEFINE MQQUEUE CRDP.CREDIT_CARD_%_RATE_INQUIRY
```

This queue cannot be protected by a generic profile, such as, CRDP.**.

Switch profiles

To control the security checking performed by MQSeries, you must define *switch profiles*. A switch profile is a normal RACF profile that has a special meaning to MQSeries. The access list in switch profiles is not used by MQSeries.

Each switch profile that MQSeries detects turns off the checking for that type of resource. Switch profiles are activated during startup of the queue manager. If you change the switch profiles while the queue manager is running, you can get MQSeries to recognize the changes by issuing the MQSeries command REFRESH SECURITY.

The switch profiles must always be defined in the MQADMIN class. Table 33 on page 402 shows the valid switch profiles and the security type they control. Do not define them in the GMQADMIN class.

Note: In the descriptions that follow, the parts of profile names shown in uppercase must be entered exactly as shown. The lowercase 'ssid' part must be replaced by the queue manager name for the MQSeries subsystem you are setting up.

<i>Table 33. Switch profiles</i>	
Switch profile name	Type of resource or checking that is controlled
ssid.NO.SUBSYS.SECURITY	Subsystem security
ssid.NO.CONNECT.CHECKS	Connection security
ssid.NO.QUEUE.CHECKS	Queue security
ssid.NO.PROCESS.CHECKS	Process security
ssid.NO.NLIST.CHECKS	Namelist security
ssid.NO.CONTEXT.CHECKS	Context security
ssid.NO.ALTERNATE.USER.CHECKS	Alternate user security
ssid.NO.CMD.CHECKS	Command security
ssid.NO.CMD.RESC.CHECKS	Command resource security
Note: Generic switch profiles such as ssid.NO.** are not detected by MQSeries	

How switches work

MQSeries maintains a switch, which is associated with each of the switch profiles shown in Table 33. When a security switch is set on, the security checks associated with the switch are performed. When a security switch is set off, the security checks associated with switch are bypassed.

When a queue manager is started or when the MQADMIN class is refreshed by the MQSeries command REFRESH SECURITY, the queue manager first checks the status of RACF and the MQADMIN class. It sets its subsystem security switch off if it discovers one of these conditions:

- RACF is inactive or not installed
- The MQADMIN class is not defined
- The MQADMIN class has not been activated

If both RACF and the MQADMIN class are active, the queue manager checks the MQADMIN class to see whether any of the switch profiles have been defined. It first checks for the ssid.NO.SUBSYS.SECURITY profile. If this profile is defined, the queue manager sets its subsystem security switch off, and performs no further checks. If this profile is not defined, the queue manager sets the subsystem security switch on and checks whether any of the other switch profiles are present. If they are, the corresponding MQSeries switch is set off and that type of security is deactivated.

However, if any MQSeries switch is set on, the queue manager checks the status of the RACF class associated with the type of security corresponding to the MQSeries switch. If the class is not installed or not active, the MQSeries switch is set off. For example, process security checks are not carried out if the MQPROC class has not been activated. The class not being active is equivalent to defining an ssid.NO.PROCESS.CHECKS for every queue manager that uses this RACF database.

An example of defining switches

Three MQSeries subsystems have been defined called PROD, DEVT, and TEST respectively and all MQSeries RACF classes have been defined and activated. These subsystems have different security requirements:

- Queue manager PROD requires all MQSeries security checking to be active.

This is done by ensuring no PROD.NO.xx profiles are defined in the MQADMIN class. You can check this by using the RACF command, SEARCH, for example:

```
SEARCH CLASS(MQADMIN) MASK(PROD.NO)
```

and checking the resulting output.

- Queue manager DEVT requires only connection and queue security to be active.

This is done by defining DEVT.NO.xx profiles for each of the security types you do *not* want. For example:

```
RDEFINE MQADMIN DEVT.NO.CMD.CHECKS
RDEFINE MQADMIN DEVT.NO.CMD.RESC.CHECKS
RDEFINE MQADMIN DEVT.NO.PROCESS.CHECKS
RDEFINE MQADMIN DEVT.NO.NLIST.CHECKS
RDEFINE MQADMIN DEVT.NO.CONTEXT.CHECKS
RDEFINE MQADMIN DEVT.NO.ALTERNATE.USER.CHECKS
```

You should also check there are no other DEVT.NO.xx profiles defined in the MQADMIN class.

- Queue manager TEST does not require any security checking at all.

This is done by defining the NO.SUBSYS.SECURITY profile for TEST as follows:

```
RDEFINE MQADMIN TEST.NO.SUBSYS.SECURITY
```

Once MQSeries is active, you can display the current security settings by issuing the MQSeries command DISPLAY SECURITY.

You can also change the switch settings when MQSeries is running by defining or deleting the appropriate switch profile in the MQADMIN class. To make the changes to the switch settings active, you must issue the MQSeries command REFRESH SECURITY for the MQADMIN class.

See “Security refreshes” on page 442 for more details about using the MQSeries commands DISPLAY SECURITY and REFRESH SECURITY.

Profiles used to protect MQSeries resources

In addition to the switch profiles that might have been defined, RACF profiles must be defined to protect the MQSeries resources.

If you do not have a resource profile defined for a particular security check, and a user issues a request that would involve making that check, MQSeries denies access.

You do not need to define profiles for security types relating to any security switch profiles that you have deactivated.

Profiles for connection security

If connection security is active, you must define profiles in the MQCONN class and permit the necessary groups or user IDs access to those profiles, so that they can connect to MQSeries subsystems.

To enable a connection to be made, you must grant users RACF READ access to the appropriate profile.

Notes:

1. For information about the user IDs checked for different security requests, see “User IDs for security checking” on page 430.
2. Resource level security (RESLEVEL) checks are also made at connection time. For details, see “Using the RESLEVEL security profile” on page 425.

Connection security profiles for the Batch/TSO adapter

Profiles for checking connections from Batch or TSO take the form:

```
ssid.BATCH
```

RRS uses the Batch/TSO adapter for connection security.

For connection requests through Batch or TSO, you must permit the job or TSO user ID to access the connection profile. For example, the following RACF command allows Batch and TSO users in the CONNTQM1 group to connect to the queue manager TQM1:

```
RDEFINE MQCONN TQM1.BATCH UACC(NONE)  
PERMIT TQM1.BATCH CLASS(MQCONN) ID(CONNTQM1) ACCESS(READ)
```

Connection security profiles for the CICS adapter

Profiles for checking connections from CICS take the form:

```
ssid.CICS
```


For connection requests by CICS, you need only define the CICS address space user ID access to the connection profile.

For example, the following RACF commands allow the CICS address space user ID KCBCICS to connect to the queue manager TQM1:

```
RDEFINE MQCONN TQM1.CICS UACC(NONE)
PERMIT TQM1.CICS CLASS(MQCONN) ID(KCBCICS) ACCESS(READ)
```

Connection security profiles for the IMS adapter

Profiles for checking connections from IMS take the form:

```
ssid.IMS
```

For connection requests by IMS, define access to the connection profile for the IMS control and dependent region user IDs.

For example, the following RACF commands allow:

- The IMS region user ID, IMSREG, to connect to the queue manager TQM1.
- Users in group BMPGRP to submit BMP jobs.

```
RDEFINE MQCONN TQM1.IMS UACC(NONE)
PERMIT TQM1.IMS CLASS(MQCONN) ID(IMSREG,BMPGRP) ACCESS(READ)
```

Connection security profiles for distributed queuing

Profiles for checking connections from distributed queuing (without CICS ISC) take the form:

```
ssid.CHIN
```

For connection requests by the channel initiator, define access to the connection profile for the user ID used by the channel initiator started task address space.

For example, the following RACF commands allow the channel initiator address space running with user ID DQCTRL to connect to the queue manager TQM1:

```
RDEFINE MQCONN TQM1.CHIN UACC(NONE)
PERMIT TQM1.CHIN CLASS(MQCONN) ID(DQCTRL) ACCESS(READ)
```

Profiles for queue security

If queue security is active, you must define profiles in the MQQUEUE or GMQQUEUE classes and permit the necessary groups or user IDs access to these profiles, so they can issue MQSeries API requests that use queues.

Profiles for queue security take the form:

ssid.queueename

where queueename is the name of the queue being opened, as specified in the object descriptor on the **MQOPEN** or **MQPUT1** call. For details of how queue security operates when the queue name is that of an alias or a model queue, see “Considerations for alias queues” on page 407 and “Considerations for model queues” on page 409.

The RACF access required to open a queue depends on the **MQOPEN** or **MQPUT1** options specified. If more than one of the MQOO_* and MQPMO_* options is coded, the queue security check is performed for the highest RACF authority required.

Table 34. Access levels for queue security

MQOPEN or MQPUT1 option	RACF access level required to access ssid.queueename
MQOO_BROWSE	READ
MQOO_INQUIRE	READ
MQOO_BIND_*	UPDATE
MQOO_INPUT_*	UPDATE
MQOO_OUTPUT or MQPUT1	UPDATE
MQOO_PASS_ALL_CONTEXT MQPMO_PASS_ALL_CONTEXT	UPDATE
MQOO_PASS_IDENTITY_CONTEXT MQPMO_PASS_IDENTITY_CONTEXT	UPDATE
MQOO_SAVE_ALL_CONTEXT	UPDATE
MQOO_SET_IDENTITY_CONTEXT MQPMO_SET_IDENTITY_CONTEXT	UPDATE
MQOO_SET_ALL_CONTEXT MQPMO_SET_ALL_CONTEXT	UPDATE
MQOO_SET	ALTER

For example, on MQSeries subsystem QM77, all user IDs in the RACF group PAYGRP are to be given access to get messages from or put messages to all queues with names beginning with 'PAY.'. You can do this using these RACF commands:

```
RDEFINE MQQUEUE QM77.PAY.** UACC(NONE)
PERMIT QM77.PAY.** CLASS(MQQUEUE) ID(PAYGRP) ACCESS(UPDATE)
```

Also, all user IDs in the PAYGRP group must have access to put messages on queues that do not follow the PAY naming convention. For example:

```
REQUEST_QUEUE_FOR_PAYROLL
SALARY.INCREASE.SERVER
REPLIES.FROM.SALARY.MODEL
```

You can do this by defining profiles for these queues in the GMQQUEUE class and giving access to that class as follows:

```
RDEFINE GMQQUEUE PAYROLL.EXTRAS UACC(NONE)
      ADDMEM(QM77.REQUEST_QUEUE_FOR_PAYROLL,
             QM77.SALARY.INCREASE.SERVER,
             QM77.REPLIES.FROM.SALARY.MODEL)
PERMIT PAYROLL.EXTRAS CLASS(GMQQUEUE) ID(PAYGRP) ACCESS(UPDATE)
```

Note: If the RACF access level that an application has to a queue security profile is changed, the changes will only take effect for any new object handles obtained (that is, new **MQOPENS**) for that queue. Those handles already in existence at the time of the change retain their existing access to the queue. If an application is required to use its changed access level to the queue rather than its existing access level, it must close and re-open the queue for each object handle that requires the change.

Other types of security checks might also occur at the time the queue is opened depending on the open options specified and the types of security that are active. See also "Profiles for context security" on page 417 and "Profiles for alternate user security" on page 415. For a summary table showing the open options and the security authorization needed when queue, context, and alternate user security are all active, see Table 41 on page 419.

Considerations for alias queues

When you issue an **MQOPEN** or **MQPUT1** call for an alias queue, MQSeries makes a resource check against the queue name specified in the object descriptor (MQOD) on the call. It does not check if the user is allowed access to the target queue name.

For example, an alias queue called PAYROLL.REQUEST resolves to a target queue of PAY.REQUEST. If queue security is active, you need only be authorized to access the queue PAYROLL.REQUEST. No check is made to see if you are authorized to access the queue PAY.REQUEST.

Using alias queues to distinguish between MQGET and MQPUT requests

The range of MQI calls available in one access level can cause a problem if you want to restrict access to a queue to allow only the **MQPUT** call or only the **MQGET** call. A queue can be protected by defining two aliases that resolve to that queue: one that enables applications to get messages from the queue, and one that enable applications to put messages on the queue.

The following text gives you an example of how you can define your queues to MQSeries:

```
DEFINE QLOCAL(MUST_USE_ALIAS_TO_ACCESS) GET(ENABLED)
      PUT(ENABLED)

DEFINE QALIAS(USE_THIS_ONE_FOR_GETS) GET(ENABLED)
      PUT(DISABLED) TARGQ(MUST_USE_ALIAS_TO_ACCESS)

DEFINE QALIAS(USE_THIS_ONE_FOR_PUTS) GET(DISABLED)
      PUT(ENABLED) TARGQ(MUST_USE_ALIAS_TO_ACCESS)
```

You must also make these RACF definitions:

```
RDEFINE MQQUEUE ssid.MUST_USE_ALIAS_TO_ACCESS UACC(NONE)
RDEFINE MQQUEUE ssid.USE_THIS_ONE_FOR_GETS UACC(NONE)
RDEFINE MQQUEUE ssid.USE_THIS_ONE_FOR_PUTS UACC(NONE)
```

Then you ensure no users have access to the queue `ssid.MUST_USE_ALIAS_TO_ACCESS`, and give the appropriate users or groups access to the alias. You can do this using these RACF commands:

```
PERMIT ssid.USE_THIS_ONE_FOR_GETS CLASS(MQQUEUE)
      ID(GETUSER,GETGRP) ACCESS(UPDATE)
PERMIT ssid.USE_THIS_ONE_FOR_PUTS CLASS(MQQUEUE)
      ID(PUTUSER,PUTGRP) ACCESS(UPDATE)
```

This means user ID `GETUSER` and user IDs in the group `GETGRP` are only allowed to get messages on `MUST_USE_ALIAS_TO_ACCESS` through the alias queue `USE_THIS_ONE_FOR_GETS`; and user ID `PUTUSER` and user IDs in the group `PUTGRP` are only allowed to put messages through the alias queue `USE_THIS_ONE_FOR_PUTS`.

If you want to use a technique like this, you must inform your application developers, so they can design their programs appropriately.

Considerations for model queues

When you open a model queue, MQSeries security makes two queue security checks:

1. Are you authorized to access the model queue?
2. Are you authorized to access the dynamic queue to which the model queue resolves?

If the dynamic queue name contains a trailing * character, this * is replaced by a character string generated by MQSeries, to create a dynamic queue with a unique name. However, because the whole name, including this generated string, is used for checking authority, you should define generic profiles for these queues.

For example, an **MQOPEN** call uses a model queue name of CREDIT.CHECK.REPLY.MODEL and a dynamic queue name of CREDIT.REPLY.* on queue manager MQSP. To do this, you must issue the following RACF commands to define the necessary queue profiles:

```
RDEFINE MQQUEUE MQSP.CREDIT.CHECK.REPLY.MODEL
RDEFINE MQQUEUE MQSP.CREDIT.REPLY.**
```

You must also issue the corresponding RACF PERMIT commands to allow the user access to these profiles.

A typical dynamic queue name created by an **MQOPEN** is something like CREDIT.REPLY.A346EF00367849A0. The precise value of the last qualifier is unpredictable; this is why you should use generic profiles for such queue names.

A number of MQSeries utilities put messages on dynamic queues. You should define profiles for the following dynamic queue names, and provide RACF UPDATE access to the relevant user IDs (see “User IDs for security checking” on page 430 for the correct user IDs):

```
SYSTEM.CSQUTIL.* (used by CSQUTIL)
SYSTEM.CSQOREXX.* (used by the operations and control panels)
SYSTEM.CSQXCMD.* (used by the channel initiator when processing CSQINPX)
CSQ4SAMP.* (used by the MQSeries supplied samples)
```

You might also consider defining a profile to control use of the dynamic queue name used by default in the application programming copy members. The MQSeries-supplied copybooks contain a default *DynamicQName*, which is CSQ.*. This enables an appropriate RACF profile to be established.

Note: Do not allow application programmers to specify a single * for the dynamic queue name. If you do, you must define an ssid.** profile in the MQQUEUE class, and you would have to give it wide-ranging access. This means that this profile could also be used for other non-dynamic queues that do not have a more specific RACF profile. Your users could, therefore, gain access to queues you do not want them to access.

Close options on permanent dynamic queues

If an application opens a permanent dynamic queue that was created by another application and then attempts to delete that queue with an **MQCLOSE** option, some extra security checks are applied when the attempt is made. See Table 35.

<i>Table 35. Access levels for close options on permanent dynamic queues</i>	
MQCLOSE option	RACF access level required to ssid.queue name
MQCO_DELETE	ALTER
MQCO_DELETE_PURGE	ALTER

Security and remote queues

When a message is put on a remote queue, the queue security that is performed by the local queue manager depends on how the remote queue is specified when it is opened. For example:

1. If the remote queue has been defined on the local queue manager through the MQSeries command, DEFINE QREMOTE, the queue that is checked is the name of the remote queue. For example, if a remote queue is defined on queue manager MQS1 as follows:

```
DEFINE QREMOTE(BANK7.CREDIT.REFERENCE)
         RNAME(CREDIT.SCORING.REQUEST)
         RQMNAME(BNK7)
         XMITQ(BANK1.TO.BANK7)
```

In this case, a profile for BANK7.CREDIT.REFERENCE, must be defined in the MQQUEUE class.

2. If the *ObjectQMgrName* for the request does not resolve to the local queue manager, the queue used for queue security is the name of the transmission queue used to send messages to the remote queue manager specified by the *MQOD_ObjectQMgrName*.

For example, the transmission queue BANK1.TO.BANK7 is defined on queue manager MQS1. An **MQPUT1** request is then issued on MQS1 specifying *ObjectName* as BANK1.INTERBANK.TRANSFERS and an *ObjectQMgrName* of BANK1.TO.BANK7. In this case, the user performing the request must have access to MQS1.BANK1.TO.BANK7.

3. If you make an **MQPUT** request to a queue and specify *ObjectQMgrName* as the name of an alias of the local queue manager, only the queue name is checked for security, not that of the queue manager.

When the message gets to the remote queue manager it might be subject to additional security processing. For more information, see the *MQSeries Intercommunication* manual.

Dead-letter queue security

Undelivered messages can be put on a special queue called the dead-letter queue. If you have sensitive data that could possibly end up on this queue, you must consider the security implications of this because you do not want unauthorized users to be able to retrieve this data.

Each of the following must be able to put messages onto the dead-letter queue:

- Application programs.
- The channel initiator address space and any MCA user IDs. (If the RESLEVEL profile is not present, or is defined so that network-received user IDs are checked, the network-received user ID also needs authority to put messages on the dead-letter queue.)
- For distributed queuing using CICS, the various MCA transactions.
- CKTI, the MQSeries-supplied CICS task initiator.
- CSQQTRMN, the MQSeries-supplied IMS trigger monitor.

The only application able to retrieve messages from the dead-letter queue should be a 'special' application that processes these messages. However, a problem arises if you give applications RACF UPDATE authority to the dead-letter queue for **MQPUTs** because they can then automatically retrieve messages from the queue using **MQGET** calls. You cannot disable the dead-letter queue for get operations because, if you do, not even the 'special' applications could retrieve the messages.

One solution to this problem is set up a two-level access to the dead-letter queue. CKTI, message channel agent transactions or the channel initiator address space, and 'special' applications have direct access; other applications can only access the dead-letter queue through an alias queue. This alias is defined to allow applications to put messages on the dead-letter queue, but not to get messages from it.

This is how it might work:

1. Define the real dead-letter queue with attributes PUT(ENABLED) and GET(ENABLED), as shown in the sample thlqual.SCSQPROC(CSQ4INYG).
2. Give RACF UPDATE authority for the dead-letter queue to the following user IDs:
 - User IDs that the CKTI and the MCAs or channel initiator address space run under.
 - The user IDs associated with the 'special' dead-letter queue processing application.
3. Define an alias queue that resolves to the real dead-letter queue, but give the alias queue these attributes: PUT(ENABLED) and GET(DISABLED). Give the alias queue a name with the same stem as the dead-letter queue name but append the characters ".PUT" to this stem. For example, if the dead-letter queue name is ssid.DEAD.QUEUE, the alias queue name would be ssid.DEAD.QUEUE.PUT.

4. To put a message on the dead-letter queue, an application uses the alias queue. This what your application must do:
 - Retrieve the name of the real dead-letter queue. To do this, it opens the queue manager object using **MQOPEN** and then issues an **MQINQ** to get the dead-letter queue name.
 - Build the name of the alias queue by appending the characters '.PUT' to this name, in this case, ssid.DEAD.QUEUE.PUT.
 - Open the alias queue, ssid.DEAD.QUEUE.PUT.
 - Put the message on the real dead-letter queue by issuing an **MQPUT** against the alias queue.
5. Give the user ID associated with the application RACF UPDATE authority to the alias, but no access (authority NONE) to the real dead-letter queue. This means that:
 - The application can put messages onto the dead-letter queue using the alias queue.
 - The application cannot get messages from the dead-letter queue using the alias queue because the alias queue is disabled for get operations.

The application cannot get any messages from the real dead-letter queue either because it does have the correct RACF authority.

Table 36 summarizes the RACF authority required for the various participants in this solution.

<i>Table 36. RACF authority to the dead-letter queue and its alias</i>		
Associated user IDs	Real dead-letter queue (ssid.DEAD.QUEUE)	Alias dead-letter queue (ssid.DEAD.QUEUE.PUT)
MCA or channel initiator address space and CKTI	UPDATE	NONE
'Special' application (for dead-letter queue processing)	UPDATE	NONE
User-written application user IDs	NONE	UPDATE

If you use this method, the application cannot determine the maximum message length, MAXMSGL, of the dead-letter queue. This is because the MAXMSGL attribute cannot be retrieved from an alias queue. Therefore, your application should assume that the maximum message length is 4 MB, the maximum size MQSeries for OS/390 supports. The real dead-letter queue should also be defined with a MAXMSGL attribute of 4 MB.

Note: User-written application programs should not normally use alternate user authority to put messages on the dead-letter queue. This reduces the number of user IDs that have access to the dead-letter queue.

System queue security

Many of the system queues are accessed by the ancillary parts of the queue manager:

- The CSQUTIL utility
- The operations and control panels
- The channel initiator address space for distributed queuing without CICS
- The CICS transactions for distributed queuing using CICS

The user IDs under which these run must be given RACF access to these queues, as shown in the following table:

	CSQUTIL	Operations and control panels	Channel initiator for distributed queuing without CICS	Transactions for distributed queuing with CICS
SYSTEM.ADMIN.CHANNEL.EVENT			UPDATE	
SYSTEM.CHANNEL.COMMAND				UPDATE
SYSTEM.CHANNEL.INITQ			UPDATE	
SYSTEM.CHANNEL.SEQNO				UPDATE
SYSTEM.CHANNEL.SYNCQ			UPDATE	
SYSTEM.CLUSTER.COMMAND.QUEUE			ALTER	
SYSTEM.CLUSTER.REPOSITORY.QUEUE			UPDATE	
SYSTEM.CLUSTER.TRANSMIT.QUEUE			ALTER	
SYSTEM.COMMAND.INPUT	UPDATE	UPDATE	UPDATE	
SYSTEM.COMMAND.REPLY.MODEL	UPDATE	UPDATE	UPDATE	
SYSTEM.CSQOREXX.*		UPDATE		
SYSTEM.CSQUTIL.*	UPDATE			
SYSTEM.CSQXCMD.*			UPDATE	

Profiles for processes

If process security is active, you must define profiles in the MQPROC or GMQPROC classes and permit the necessary groups or user IDs access to these profiles, so they can use MQI requests that use processes. Profiles for processes take the form:

ssid.processname

where processname is the name of the process being opened. The following table shows the access required for opening a process.

Protecting MQSeries resources

<i>Table 37. Access levels for process security</i>	
MQOPEN option	RACF access level required to ssid.processname
MQOO_INQUIRE	READ

For example, on queue manager MQS9, the RACF group INQVPRC must be able to inquire (**MQINQ**) on all processes starting with the letter V. The RACF definitions for this would be:

```
RDEFINE MQPROC MQS9.V* UACC(NONE)
PERMIT MQS9.V* CLASS(MQPROC) ID(INQVPRC) ACCESS(READ)
```

Alternate user security might also be active, depending on the open options specified when a process definition object is opened.

Profiles for namelists

If namelist security is active, you define profiles in the MQNLIST or GMQNLIST classes and give the necessary groups or user IDs access to these profiles.

Profiles for namelists take the form:

```
ssid.namelistname
```

where `namelistname` is the name of the namelist being opened. The following table shows the access required for opening a namelist.

<i>Table 38. Access levels for namelist security</i>	
MQOPEN option	RACF access level required to ssid.namelistname
MQOO_INQUIRE	READ

For example, on queue manager PQM3, the RACF group DEPT571 must be able to inquire (**MQINQ**) on these namelists:

- All namelists starting with "DEPT571."
- PRINTER/DESTINATIONS/DEPT571
- AGENCY/REQUEST/QUEUES
- WAREHOUSE.BROADCAST

The RACF definitions to do this are:

```
RDEFINE MQNLIST PQM3.DEPT571.** UACC(NONE)
PERMIT PQM3.DEPT571.** CLASS(MQNLIST) ID(DEPT571) ACCESS(READ)

RDEFINE GMQNLIST NLISTS.FOR.DEPT571 UACC(NONE)
      ADDMEM(PQM3.PRINTER/DESTINATIONS/DEPT571,
            PQM3.AGENCY/REQUEST/QUEUES,
            PQM3.WAREHOUSE.BROADCAST)
PERMIT NLISTS.FOR.DEPT571 CLASS(GMQNLIST) ID(DEPT571) ACCESS(READ)
```

Alternate user security might be active, depending on the options specified when a namelist object is opened.

Profiles for alternate user security

If alternate user security is active, you must define profiles in the MQADMIN class and permit the necessary groups or user IDs access to these profiles, so that they can use the ALTERNATE_USER_AUTHORITY options when the queue is opened.

Profiles for alternate user security take the following form:

```
ssid.ALTERNATE.USER.alternateuserid
```

where *alternateuserid* is the value of the *AlternateUserId* field in the object descriptor. The following table shows the access when specifying an alternate user option.

<i>Table 39. Access levels for alternate user security</i>	
MQOPEN or MQPUT1 option	RACF access level required
MQOO_ALTERNATE_USER_AUTHORITY MQPMO_ALTERNATE_USER_AUTHORITY	UPDATE

In addition to alternate user security checks, other security checks for queue, process, namelist, and context security can also be made. The alternate user ID, if provided, is only used for security checks on queue, process definition, or namelist resources. For alternate user and context security checks, the user ID requesting the check is used. For details about how user IDs are handled, see “User IDs for security checking” on page 430. For a summary table showing the open options and the security checks required when queue, context and alternate user security are all active, see Table 41 on page 419.

An alternate user profile gives the requesting user ID access to resources associated with the user ID specified in the alternate user ID. For example, the payroll server running under user ID PAYSERV on queue manager QMPY processes requests from personnel user IDs, all of which start with PS. To cause the work performed by the payroll server to be carried out under the user ID of the requesting user, alternate user authority is used. The payroll server knows which user ID to specify as the alternate user ID because the requesting programs generate messages using the MQPMO_DEFAULT_CONTEXT put message option.

Protecting MQSeries resources

See “User IDs for security checking” on page 430 for more details about from where alternate user IDs are obtained.

The following example RACF definitions enable the server program to specify alternate user IDs starting with the characters PS:

```
REDEFINE MQADMIN QMPY.ALTERNATE.USER.PS* UACC(NONE)
PERMIT QMPY.ALTERNATE.USER.PS* CLASS(MQADMIN) ID(PAYSERV) ACCESS(UPDATE)
```

Notes:

1. The *AlternateUserId* field in the object descriptor is 12 bytes long. All 12 bytes are used in the profile checks, but only the first eight bytes are used as the user ID by MQSeries. If this user ID truncation is not desirable, application programs making the request should translate any alternate user ID over 8 bytes into something more appropriate.
2. If you specify MQOO_ALTERNATE_USER_AUTHORITY or MQPMO_ALTERNATE_USER_AUTHORITY and you do not specify an *AlternateUserId* field in the object descriptor, a user ID of blanks is used. For the purposes of the alternate user security check the user ID used for the *AlternateUserId* qualifier is -BLANK-. For example:

```
RDEF MQADMIN ssid.ALTERNATE.USER.-BLANK-
```

If the user is allowed to access this profile, all further checks are made with a user ID of blanks. For details of blank user IDs, see “Blank user IDs and UACC levels” on page 436.

The administration of alternate user IDs is easier if you have a naming convention for user IDs that enables you to use generic alternate user profiles. If they do not, you could use the RACF RACVARS feature. For details about using RACVARS, see the *Security Server (RACF) Security Administrator's Guide*.

When a message is put to a queue that has been opened with alternate user authority and the context of the message has been generated by the queue manager, the MQMD_USER_IDENTIFIER field is set to the alternate user ID.

Profiles for context security

If context security is active, you must define a profile in the MQADMIN class called:

```
ssid.CONTEXT
```

You must give the necessary groups or user IDs access to this profile. The following table shows the access level required, depending on the specification of the context options when the queue is opened.

Table 40. Access levels for context security

MQOPEN or MQPUT1 option	RACF access level required to ssid.CONTEXT
MQPMO_NO_CONTEXT	No context security check
MQPMO_DEFAULT_CONTEXT	No context security check
MQOO_SAVE_ALL_CONTEXT	No context security check
MQOO_PASS_IDENTITY_CONTEXT MQPMO_PASS_IDENTITY_CONTEXT	READ
MQOO_PASS_ALL_CONTEXT MQPMO_PASS_ALL_CONTEXT	READ
MQOO_SET_IDENTITY_CONTEXT MQPMO_SET_IDENTITY_CONTEXT	UPDATE
MQOO_SET_ALL_CONTEXT MQPMO_SET_ALL_CONTEXT	CONTROL
MQOO_OUTPUT or MQPUT1 (Usage=XMITQ)	CONTROL
<p>Note: The user ID of the receiving channel initiator address space (or the MCA user ID if one has been specified) requires CONTROL access to ssid.CONTEXT in order to put messages on the destination queue. If the RESLEVEL profile requires that two user IDs are checked, the network-received user ID also needs CONTROL access to ssid.CONTEXT.</p>	

If you put commands on the system-command input queue, use the default context put message option to associate the correct user ID with the command.

For example, the MQSeries-supplied utility program CSQUTIL can be used to off-load and reload messages in queues. When off-loaded messages are restored to a queue, the CSQUTIL utility uses the MQOO_SET_ALL_CONTEXT option to return the messages to their original state. In addition to the queue security required by this open option, context authority is also required. For example, if this authority is required by the group BACKGRP on queue manager MQS1, this would be defined by:

```
RDEFINE MQADMIN MQS1.CONTEXT UACC(NONE)
PERMIT MQS1.CONTEXT CLASS(MQADMIN) ID(BACKGRP) ACCESS(CONTROL)
```

Protecting MQSeries resources

Depending on the options specified, and the types of security performed, other types of security checks might also occur when the queue is opened. These include queue security (see “Profiles for queue security” on page 406), and alternate user security (see “Profiles for alternate user security” on page 415). For a summary table showing the open options and the security checks required when queue, context and alternate user security are all active, see Table 41 on page 419.

API-resource security access quick reference

Table 41 summarizes the **MQOPEN**, **MQPUT1**, and **MQCLOSE** options and the access required by the different resource security types.

Table 41. MQOPEN, MQPUT1, and MQCLOSE options and the security authorization required. Callouts shown like this (1) refer to the notes following this table.				
		Minimum RACF access level required		
	RACF class: RACF profile:	MQQUEUE (1) (2)	MQADMIN (3)	MQADMIN (4)
MQOPEN option				
MQOO_INQUIRE (1)		READ (5)	No check	No check
MQOO_BROWSE		READ	No check	No check
MQOO_INPUT_*		UPDATE	No check	No check
MQOO_SAVE_ALL_CONTEXT (6)		UPDATE	No check	No check
MQOO_OUTPUT (USAGE=NORMAL) (7)		UPDATE	No check	No check
MQOO_PASS_IDENTITY_CONTEXT (8)		UPDATE	READ	No check
MQOO_PASS_ALL_CONTEXT (8) (9)		UPDATE	READ	No check
MQOO_SET_IDENTITY_CONTEXT (8) (9)		UPDATE	UPDATE	No check
MQOO_SET_ALL_CONTEXT (8) (10)		UPDATE	CONTROL	No check
MQOO_OUTPUT (USAGE=XMITQ) (11)		UPDATE	CONTROL	No check
MQOO_SET		ALTER	No check	No check
MQOO_ALTERNATE_USER_AUTHORITY (1)		(12)	(12)	UPDATE
MQPUT1 option				
Put on a normal queue (7)		UPDATE	No check	No check
MQPMO_PASS_IDENTITY_CONTEXT		UPDATE	READ	No check
MQPMO_PASS_ALL_CONTEXT		UPDATE	READ	No check
MQPMO_SET_IDENTITY_CONTEXT		UPDATE	UPDATE	No check
MQPMO_SET_ALL_CONTEXT		UPDATE	CONTROL	No check
MQOO_OUTPUT Put on a transmission queue (11)		UPDATE	CONTROL	No check
MQPMO_ALTERNATE_USER_AUTHORITY		(13)	(13)	UPDATE
MQCLOSE option				
MQCO_DELETE (14)		ALTER	No check	No check
MQCO_DELETE_PURGE (14)		ALTER	No check	No check

Notes:

1. This option is not restricted to queues. Use the MQNLIST class for namelists, and the MQPROC class for processes.
2. Use RACF profile: ssid.localresourcename
3. Use RACF profile: ssid.CONTEXT
4. Use RACF profile: ssid.ALTERNATE.USER.alternateuserid
alternateuserid is the user identifier that is specified in the *AlternateUserId* field of the object descriptor. Note that all 12 characters of the *AlternateUserId* field are used for this check, unlike other checks where only the first 8 characters of a user identifier are used.
5. No check is made when opening the queue manager for inquiries.

Protecting MQSeries resources

6. MQOO_INPUT_★ must be specified as well. This is valid for a local, model or alias queue.
7. This check is done for a local or model queue that has a *Usage* queue attribute of MQUS_NORMAL, and also for an alias or remote queue (that is defined to the connected queue manager.) If the queue is a remote queue that is opened specifying an *ObjectQMgrName* (not the name of the connected queue manager) explicitly, the check is carried out against the queue with the same name as *ObjectQMgrName* (which must be a local queue with a *Usage* queue attribute of MQUS_TRANSMISSION).
8. MQOO_OUTPUT must be specified as well.
9. MQOO_PASS_IDENTITY_CONTEXT is implied as well by this option.
10. MQOO_PASS_IDENTITY_CONTEXT, MQOO_PASS_ALL_CONTEXT and MQOO_SET_IDENTITY_CONTEXT are implied as well by this option.
11. This check is done for a local or model queue that has a *Usage* queue attribute of MQUS_TRANSMISSION, and is being opened directly for output. It does not apply if a remote queue is being opened.
12. At least one of MQOO_INQUIRE, MQOO_BROWSE, MQOO_INPUT_★, MQOO_OUTPUT or MQOO_SET must be specified as well. The check carried out is the same as that for the other options specified.
13. The check carried out is the same as that for the other options specified.
14. This only applies when a permanent dynamic queue has been opened directly, that is, not opened through a model queue.

Profiles for command security

If you have not defined the command security switch profile, `ssid.NO.CMD.CHECKS` because you want security checking for commands, you must add profiles to the MQCMDS class.

The names of the RACF profiles for command security checking are based on the command names themselves. These profiles take the form:

```
ssid.verb.pkw
```

For example, the profile name for the MQSeries command `ALTER QLOCAL` in subsystem `AMQS` is:

```
AMQS.ALTER.QLOCAL
```

Table 42 on page 423 shows, for each MQSeries command, the profiles required for command security checking to be carried out, and the corresponding access level for each profile in the MQCMDS class.

Profiles for command resource security

If you have not defined the command resource security switch profile, `ssid.NO.CMD.RESC.CHECKS`, because you want security checking for resources associated with commands, you must add resource profiles to the MQADMIN class for each resource.

Profiles for command resource security checking take the form:

```
ssid.type.localresourcename
```

For example, the RACF profile name for command resource security checking against the model queue `CREDIT.WORTHY` in subsystem `AMQS` is:

```
AMQS.QUEUE.CREDIT.WORTHY
```

Because the profiles for all types of command resource are held in the MQADMIN class, the “type” part of the profile name is needed in the profile to distinguish between resources of different types that have the same name. The “type” part of the profile name can be `CHANNEL`, `QUEUE`, `PROCESS`, or `NAMELIST`. For example, a user might be authorized to define `ssid.QUEUE.PAYROLL.ONE`, but not authorized to define `ssid.PROCESS.PAYROLL.ONE`.

Table 42 on page 423 shows for each MQSeries command, the profiles you need to enable command resource security checking to be carried out, and the access level that you need for each in the MQADMIN class.

Command resource security checking for alias queues

When you define an alias queue, command resource security checks are only performed against the name of the alias queue, not against the name of the target queue to which the alias resolves.

Alias queues can resolve to both local and remote queues. If you do not want to permit users access to certain local or remote queues, you must do both of the following:

1. Do not allow the users access to these local and remote queues.
2. Restrict the users from being able to define aliases for these queues. That is, prevent them from being able to issue `DEFINE QALIAS` and `ALTER QALIAS` commands.

Command resource security checking for remote queues

When you define a remote queue, command resource security checks are performed only against the name of the remote queue. No checks are performed against the names of the queues specified in the `RNAME` or `XMITQ` attributes in the remote queue object definition. For more information about the attributes of queues, see the *MQSeries Command Reference* manual.

Table 42 (Page 1 of 2). Commands, profiles, and their access levels

Command	MQCMDSD		MQADMIN	
	Profile	Access level	Profile	Access level
ALTER CHANNEL	ssid.ALTER.CHANNEL	ALTER	ssid.CHANNEL.channel	ALTER
ALTER NAMELIST	ssid.ALTER.NAMELIST	ALTER	ssid.NAMELIST.namelist	ALTER
ALTER PROCESS	ssid.ALTER.PROCESS	ALTER	ssid.PROCESS.process	ALTER
ALTER QALIAS	ssid.ALTER.QALIAS	ALTER	ssid.QUEUE.queue	ALTER
ALTER QLOCAL	ssid.ALTER.QLOCAL	ALTER	ssid.QUEUE.queue	ALTER
ALTER QMGR	ssid.ALTER.QMGR	ALTER	No check	
ALTER QMODEL	ssid.ALTER.QMODEL	ALTER	ssid.QUEUE.queue	ALTER
ALTER QREMOTE	ssid.ALTER.QREMOTE	ALTER	ssid.QUEUE.queue	ALTER
ALTER SECURITY	ssid.ALTER.SECURITY	ALTER	No check	
ALTER STGCLASS	ssid.ALTER.STGCLASS	ALTER	No check	
ALTER TRACE	ssid.ALTER.TRACE	ALTER	No check	
ARCHIVE LOG	ssid.ARCHIVE.LOG	CONTROL	No check	
DEFINE BUFFPOOL	ssid.DEFINE.BUFFPOOL	ALTER	No check	
DEFINE CHANNEL	ssid.DEFINE.CHANNEL	ALTER	ssid.CHANNEL.channel	ALTER
DEFINE MAXSMSGS	ssid.DEFINE.MAXSMSGS	ALTER	No check	
DEFINE NAMELIST	ssid.DEFINE.NAMELIST	ALTER	ssid.NAMELIST.namelist	ALTER
DEFINE PROCESS	ssid.DEFINE.PROCESS	ALTER	ssid.PROCESS.process	ALTER
DEFINE PSID	ssid.DEFINE.PSID	ALTER	No check	
DEFINE QALIAS	ssid.DEFINE.QALIAS	ALTER	ssid.QUEUE.queue	ALTER
DEFINE QLOCAL	ssid.DEFINE.QLOCAL	ALTER	ssid.QUEUE.queue	ALTER
DEFINE QMODEL	ssid.DEFINE.QMODEL	ALTER	ssid.QUEUE.queue	ALTER
DEFINE QREMOTE	ssid.DEFINE.QREMOTE	ALTER	ssid.QUEUE.queue	ALTER
DEFINE STGCLASS	ssid.DEFINE.STGCLASS	ALTER	No check	
DELETE CHANNEL	ssid.DELETE.CHANNEL	ALTER	ssid.CHANNEL.channel	ALTER
DELETE NAMELIST	ssid.DELETE.NAMELIST	ALTER	ssid.NAMELIST.namelist	ALTER
DELETE PROCESS	ssid.DELETE.PROCESS	ALTER	ssid.PROCESS.process	ALTER
DELETE QALIAS	ssid.DELETE.QALIAS	ALTER	ssid.QUEUE.queue	ALTER
DELETE QLOCAL	ssid.DELETE.QLOCAL	ALTER	ssid.QUEUE.queue	ALTER
DELETE QMODEL	ssid.DELETE.QMODEL	ALTER	ssid.QUEUE.queue	ALTER
DELETE QREMOTE	ssid.DELETE.QREMOTE	ALTER	ssid.QUEUE.queue	ALTER
DELETE STGCLASS	ssid.DELETE.STGCLASS	ALTER	No check	
DISPLAY CHANNEL	ssid.DISPLAY.CHANNEL	READ	No check	
DISPLAY CHSTATUS	ssid.DISPLAY.CHSTATUS	READ	No check	
DISPLAY CLUSQMGR	ssid.DISPLAY.CLUSQMGR	READ	No check	
DISPLAY CMDSERV	ssid.DISPLAY.CMDSERV	READ	No check	
DISPLAY DQM	ssid.DISPLAY.DQM	READ	No check	
DISPLAY MAXSMSGS	ssid.DISPLAY.MAXSMSGS	READ	No check	
DISPLAY NAMELIST	ssid.DISPLAY.NAMELIST	READ	No check	
DISPLAY PROCESS	ssid.DISPLAY.PROCESS	READ	No check	
DISPLAY QMGR	ssid.DISPLAY.QMGR	READ	No check	
DISPLAY QUEUE	ssid.DISPLAY.QUEUE	READ	No check	

Command resources profiles

Table 42 (Page 2 of 2). Commands, profiles, and their access levels

Command	MQCMDSDS		MQADMIN	
	Profile	Access level	Profile	Access level
DISPLAY THREAD	ssid.DISPLAY.THREAD	READ	No check	
DISPLAY SECURITY	ssid.DISPLAY.SECURITY	READ	No check	
DISPLAY STGCLASS	ssid.DISPLAY.STGCLASS	READ	No check	
DISPLAY TRACE	ssid.DISPLAY.TRACE	READ	No check	
DISPLAY USAGE	ssid.DISPLAY.USAGE	READ	No check	
PING CHANNEL	ssid.PING.CHANNEL	CONTROL	ssid.CHANNEL.channel	CONTROL
RECOVER BSDS	ssid.RECOVER.BSDS	CONTROL	No check	
REFRESH CLUSTER	ssid.REFRESH.CLUSTER	ALTER	No check	
REFRESH SECURITY	ssid.REFRESH.SECURITY	ALTER	No check	
RESET CHANNEL	ssid.RESET.CHANNEL	CONTROL	ssid.CHANNEL.channel	CONTROL
RESET CLUSTER	ssid.RESET.CLUSTER	CONTROL	No check	
RESET TPIPE	ssid.RESET.TPIPE	CONTROL	No check	
RESOLVE CHANNEL	ssid.RESOLVE.CHANNEL	CONTROL	ssid.CHANNEL.channel	CONTROL
RESOLVE INDOUBT	ssid.RESOLVE.INDOUBT	CONTROL	No check	
RESUME QMGR	ssid.RESUME.QMGR	CONTROL	No check	
RVERIFY SECURITY	ssid.RVERIFY.SECURITY	ALTER	No check	
START CHANNEL	ssid.START.CHANNEL	CONTROL	ssid.CHANNEL.channel	CONTROL
START CHINIT	ssid.START.CHINIT	CONTROL	No check	
START CMDSERV	ssid.START.CMDSERV	CONTROL	No check	
START LISTENER	ssid.START.LISTENER	CONTROL	No check	
START TRACE	ssid.START.TRACE	CONTROL	No check	
STOP CHANNEL	ssid.STOP.CHANNEL	CONTROL	ssid.CHANNEL.channel	CONTROL
STOP CHINIT	ssid.STOP.CHINIT	CONTROL	No check	
STOP CMDSERV	ssid.STOP.CMDSERV	CONTROL	No check	
STOP LISTENER	ssid.STOP.LISTENER	CONTROL	No check	
STOP QMGR	ssid.STOP.QMGR	CONTROL	No check	
STOP TRACE	ssid.STOP.TRACE	CONTROL	No check	
SUSPEND QMGR	ssid.SUSPEND.QMGR	CONTROL	No check	

Notes:

- MQSeries does not check the authority of the user who issues the +cpf START QMGR command. However, you can use RACF facilities to protect the START xxxxMSTR command that is issued as a result of the +cpf START QMGR command. This is done by controlling access to the MVS.START.STC.xxxxMSTR profile in the RACF operator commands (OPERCMDSDS) class. For details of this, see the *Security Server (RACF) Security Administrator's Guide*. If you use this technique, and an unauthorized user tries to start MQSeries, MQSeries terminates with a reason code of 00F30216.
- The DISPLAY THREAD and DISPLAY USAGE commands might be issued internally by the queue manager; no authority is checked in these cases.

Using the RESLEVEL security profile

You can define a special profile in the MQADMIN class to control the number of user IDs checked for API-resource security. How this RESLEVEL profile affects API-resource security depends on how you are accessing MQSeries.

RESLEVEL and Batch/TSO connections

By default, when an MQSeries resource is being accessed through the Batch/TSO adapter, the user must be authorized to access that resource for the particular operation. You can bypass the security check by setting up an appropriate RESLEVEL definition.

Whether the user is checked or not is based on the user ID used at connect time; the TSO user ID or the job user ID.

For example, you can set up RESLEVEL so that when a user you trust accesses certain resources from Batch/TSO, no API-resource security checks are done; but when a user you do not trust tries to access the same resources, security checks are carried out as normal. You should set up RESLEVEL checking to bypass API-resource security checks only when you sufficiently trust the user and the programs run by that user.

RESLEVEL and system utilities

The operations and control panels and the CSQUTIL utility are batch applications. You can use RESLEVEL to bypass security checking for the SYSTEM.COMMAND.INPUT and SYSTEM.COMMAND.REPLY.MODEL queues that they use, but **not** for the dynamic queues SYSTEM.CSQOREXX.* and SYSTEM.CSQUTIL.*. Users must be authorized to use these queues as described in "System queue security" on page 413 in addition to any RESLEVEL authorization they are given.

RESLEVEL and CICS connections

By default, when an API-resource security check is made on a CICS connection, two user IDs are checked to see if access is allowed to the resource.

User IDs checked

The first user ID checked is that of the CICS address space. This is the user ID on the job card of the CICS job, or the user ID assigned to the CICS started task by the OS/390 STARTED class or the started procedures table. (It is not the CICS DFLTUSER.)

The second user ID checked is the user ID associated with the CICS transaction.

Completion codes

If one of these user IDs does not have access to the resource, the request fails with a completion code of MQRC_NOT_AUTHORIZED. Both the CICS address space user ID and the user ID of the person running the CICS transaction must have access to the resource at the correct level.

How RESLEVEL can affect the checks made

Depending on how you set up your RESLEVEL profile, you can change which user IDs are checked when access to a resource is requested. The possible checks are:

- Check the CICS address space user ID and the transaction user ID.
- Check the CICS address space user ID only.
- If the transaction is defined to CICS with RESSEC(NO), check the CICS address space user ID only⁵.
- If the transaction is defined to CICS with RESSEC(YES), check the CICS address space user ID and the transaction user ID⁵.
- Do not check any user IDs.

The user IDs checked depend on the user ID used at connection time, that is, the CICS address space user ID. This control enables you to bypass API-resource security checking for MQSeries requests coming from one system (for example, a test system, TESTCICS,) but to implement them for another (for example, a production system, PRODCICS).

Note: If you set up your CICS address space user ID with the “trusted” attribute in the STARTED class or the RACF started procedures table ICHRIN03, this overrides any user ID checks for the CICS address space established by the RESLEVEL profile for your queue manager (that is, the queue manager does not perform the security checks for the CICS address space). For more information, see the *CICS RACF Security Guide*.

RESLEVEL and IMS connections

By default, when an API-resource security check is made for an IMS adapter, two user IDs are checked to see if access is allowed to the resource.

The first user ID checked is that of the address space of the IMS region. This is taken from either the USER field from the job card or the user ID assigned to the region from the OS/390 STARTED class or the started procedures table (SPT).

The second user ID checked is associated with the work being done in the dependent region. It is determined according to the type of the dependent region as shown in Table 50 on page 432.

The setting of MQ RESLEVEL profiles cannot alter the user ID under which IMS transactions are scheduled from the IBM-supplied MQ-IMS trigger monitor program CSQQTRMN. This user ID is the PSBNAME of that trigger monitor, which by default is CSQQTRMN.

⁵ The status of the CICS security is NOT checked when taking into consideration the transaction RESSEC setting. For example, if CICS has been started with SEC=NO, but the transaction has been defined with RESSEC(YES), MQSeries still checks both user IDs.

Completion codes

If either the first or second IMS user ID does not have access to the resource, the request fails with a completion code of MQRC_NOT_AUTHORIZED.

How RESLEVEL can affect the checks made

Depending on how you set up your RESLEVEL profile, you can change which user IDs are checked when access to a resource is requested. The possible checks are:

- Check the IMS region address space user ID and the second user ID.
- Check IMS region address space user ID only.
- Do not check any user IDs.

RESLEVEL and channel initiator connections

By default, when an API-resource security check is made by the channel initiator, two user IDs are checked to see if access is allowed to the resource.

The user IDs checked can be that specified by the MCAUSER channel attribute, that received from the network, that of the channel initiator address space, or the alternate user ID for the message descriptor. This depends on the communication protocol you are using and the setting of the PUTAUT channel attribute. See “User IDs used by the channel initiator” on page 433 for more information.

Completion codes

If one of these user IDs does not have access to the resource, the request fails with a completion code of MQRC_NOT_AUTHORIZED.

How RESLEVEL can affect the checks made

Depending on how you set up your RESLEVEL profile, you can change which user IDs are checked when access to a resource is requested, and how many are checked.

The RESLEVEL profile

When an application tries to connect to MQSeries, MQSeries checks the access that the user ID associated with the adapter has to a profile in the MQADMIN class called:

```
ssid.RESLEVEL
```

The user IDs associated with each adapter are:

- The batch job ID or TSO user ID for the Batch/TSO adapter
- The CICS address space user ID for the CICS adapter
- The IMS region address space user ID for the IMS adapter
- The channel initiator address space user ID for the non-CICS mover.

This check is always performed unless the ssid.NO.SUBSYS.SECURITY switch has been set.

If there is no RESLEVEL profile, MQSeries enables checking of both the job and task (or alternate user) ID for a CICS or an IMS connection. For a batch connection, MQSeries enables checking of the job (or alternate) user ID. For the non-CICS mover, MQSeries enables checking of the MCA user ID and the channel (or alternate) user ID.

If there is a RESLEVEL profile, the level of checking depends on the environment and access level for the profile.

Table 43 shows the checks made for batch and TSO connections.

RACF access level	Level of checking
NONE	Check the job/TSO (or alternate) user ID.
READ	Check the job/TSO (or alternate) user ID.
UPDATE	Check the job/TSO (or alternate) user ID.
CONTROL	No check.
ALTER	No check.

Table 44 shows the checks made for CICS connections.

RACF access level	Level of checking
NONE	Check the CICS address space user ID and the task or alternate user ID.
READ	Check the CICS address space user ID.
UPDATE	Check the CICS address space user ID and, if the transaction has been defined with RESSEC=YES, also check the task or alternate user ID.
CONTROL	No check.
ALTER	No check.

Table 45 on page 429 shows the checks made for IMS connections.

<i>Table 45. Checks made at different RACF access levels for the IMS adapter</i>	
RACF access level	Level of checking
NONE	Check the IMS address space user ID and the IMS second user ID.
READ	Check the IMS address space user ID.
UPDATE	Check the IMS address space user ID.
CONTROL	No check.
ALTER	No check.

Table 46 shows the checks made for channel initiator connections.

<i>Table 46. Checks made at different RACF access levels for channel initiator connections</i>	
RACF access level	Level of checking
NONE	Check two user IDs.
READ	Check one user ID.
UPDATE	Check one user ID.
CONTROL	No check.
ALTER	No check.
Note: See "User IDs used by the channel initiator" on page 433 for a definition of the user IDs checked	

Table 47 through Table 53 on page 435 show how RESLEVEL affects which user IDs are checked for different MQI requests.

For example, you have a queue manager called QM66, where:

- User WS21B is to be exempt from resource security.
- CICS started task WXNCICS running under address space user ID CICSWXN is to perform full resource checking only for transactions defined with RESSEC(YES).

To define the appropriate RESLEVEL profile, issue the RACF command:

```
RDEFINE MQADMIN QM66.RESLEVEL UACC(NONE) AUDIT(ALL)
```

Then give the users access to this profile:

```
PERMIT QM66.RESLEVEL CLASS(MQADMIN) ID(WS21B) ACCESS(CONTROL)
PERMIT QM66.RESLEVEL CLASS(MQADMIN) ID(CICSWXN) ACCESS(UPDATE)
```

If you make these changes while the user IDs are connected to queue manager QM66, the users must disconnect and connect again before the change takes place.

User ID security checking

If subsystem security is not active when a user connects but, while this user is still connected, subsystem security becomes active, full resource security checking is applied to the user. The user must re-connect to get the correct RESLEVEL processing.

Important notes on using RESLEVEL

1. RESLEVEL is a very powerful option; it can cause the bypassing of all resource security checks, so that they cannot be audited by RACF.
2. Using the RESLEVEL profile means that normal security audit records are not taken. For example, if you put UAUDIT on a user, the access to the ssid.RESLEVEL profile in MQADMIN is not audited.
3. If you use the RACF WARNING option on the ssid.RESLEVEL profile, no RACF warning messages are produced.
4. If you do not have a RESLEVEL profile defined, you must be careful that no other profile in the MQADMIN class matches ssid.RESLEVEL. For example, if you have a profile in MQADMIN called ssid.** and no ssid.RESLEVEL profile, beware of the consequences of the ssid.** profile because it is used for the RESLEVEL check.

You should define an ssid.RESLEVEL profile and set the UACC to NONE, rather than not have a RESLEVEL profile at all. You should have as few users or groups in the access list as possible. For details about how to audit RESLEVEL access, see “Auditing considerations” on page 437.

5. If you make any changes to the RESLEVEL profile users must disconnect and connect again before the change takes place. (This includes stopping and restarting the channel initiator if the access that the distributed queuing address space user ID has to the RESLEVEL profile is changed.)

User IDs for security checking

MQSeries initiates security checks based on user IDs associated with users, terminals, applications, and so on. The following sections show the contents of the user IDs used for each type of security check.

User IDs for connection security

Issued from...	User ID contents
Batch/TSO, CICS, IMS	<p>The user ID normally found for connection security is one of these:</p> <ul style="list-style-type: none">• The TSO user ID.• The user ID assigned to a batch job via the USER JCL parameter.• The user ID assigned to a started task by the STARTED class or the started procedures table.

User IDs for command security and command resource security

Issued from...	User ID contents
CSQINP1 or CSQINP2	No check is made.
System command input queue	The user ID found in the <i>UserIdentifier</i> of the message descriptor of the message which contains the command. If the message does not contain a <i>UserIdentifier</i> , a user ID of blanks is passed to the security manager.
Console	The user ID signed onto the console. If the console is not signed on, the default user ID from the MQSeries subsystem initialization parameter module (CSQZPARM). This default is set the by CMDUSER operand on the CSQ6SYSP macro. To issue commands from a console, the console must have the OS/390 SYS AUTHORITY attribute.
SDSF/TSO console	TSO or job user ID.
MGCR (SVC34)	If MGCR is used with Utoken, the user ID in the Utoken. If MGCR is issued without the Utoken, the TSO or job user ID is used.
CSQUTIL	Job user ID.
CSQINPX	User ID of the channel initiator address space.

User IDs for resource security (MQOPEN and MQPUT1)

Table 47 through Table 53 on page 435 show the contents of the user IDs for normal and alternate user IDs for each type of adapter. The number of checks is defined by the RESLEVEL profile. The user ID checked is that used for **MQOPEN** or **MQPUT1** calls.

Note: All user ID fields are checked *exactly* as they are received. No conversions take place, and, for example, three user ID fields containing Bob, BOB, and bob are not equivalent.

User IDs checked for Batch

<i>Table 47. User ID checking for Batch/TSO-type user IDs</i>		
Profile name	Alternate user ID specified on open?	
	No	Yes
ssid.ALTERNATE.USER.userid	–	JOB
ssid.CONTEXT	JOB	JOB
ssid.localresourcename	JOB	ALT
Key: ALT Alternate user ID. JOB <ul style="list-style-type: none"> • The TSO user ID. • The user ID assigned to a batch job. • The user ID assigned to a started task by the STARTED class or the started procedures table. 		

User IDs checked for CICS

<i>Table 48. User ID checking for CICS-type user IDs</i>				
Profile name	Alternate user ID specified on open?			
	No		Yes	
	1 Check	2 Checks	1 Check	2 Checks
ssid.ALTERNATE.USER.userid	–	–	ADS	ADS+TXN
ssid.CONTEXT	ADS	ADS+TXN	ADS	ADS+TXN
ssid.localresourcename	ADS	ADS+TXN	ADS	ADS+ALT
Key: ALT Alternate user ID ADS The user ID associated with the CICS batch job or, if CICS is running as a started task, through the STARTED class or the started procedures table. TXN The user ID associated with the CICS transaction. This is normally the user ID of the terminal user who started the transaction. It can be the CICS DFLTUSER, a PRESET security terminal, or a manually signed-on user.				

User IDs checked for IMS

<i>Table 49. User ID checking for IMS-type user IDs</i>				
Profile name	Alternate user ID specified on open?			
	No		Yes	
	1 Check	2 Checks	1 Check	2 Checks
ssid.ALTERNATE.USER.userid	–	–	REG	REG+SEC
ssid.CONTEXT	REG	REG+SEC	REG	REG+SEC
ssid.localresourcename	REG	REG+SEC	REG	REG+ALT
Key: ALT Alternate user ID. REG The user ID is normally set through the STARTED class or the started procedures table or, if IMS is running, from a submitted job, via the USER JCL parameter. SEC The second user ID is associated with the work being done in a dependent region. It is determined according to Table 50.				

<i>Table 50. How the second user ID is determined for the IMS adapter</i>	
Types of dependent region	Hierarchy for determining the second user ID
<ul style="list-style-type: none"> • BMP message driven and successful GET UNIQUE issued. • IFP and GET UNIQUE issued. • MPP. 	User ID associated with the IMS transaction if the user is signed on. LTERM name if available. PSBNAME.
<ul style="list-style-type: none"> • BMP message driven and successful GET UNIQUE not issued. • BMP not message driven. • IFP and GET UNIQUE not issued. 	User ID associated with the IMS dependent region address space if this is not all blanks or all zeros. PSBNAME.

User IDs used by the channel initiator

The following sections describe the user IDs used and checked for TCP/IP channels, LU 6.2 channels, and client MQI requests issued over server-connection channels for both TCP/IP and LU 6.2.

You can use the PUTAUT parameter of the channel definition to determine the type of security checking used. To get consistent security checking throughout your MQSeries network, you can use the ONLYMCA and ALTMCA options.

User IDs checked for channels using TCP/IP

MCA user ID (MCA)

The user ID specified for the MCAUSER channel attribute at the receiver; if blank, the channel initiator address space user ID of the receiver or requester side is used.

Channel user ID (CHL)

For TCP/IP, where security is not supported by the communication system for the channel, the user ID of the channel initiator address space of the receiver or requestor end is used as the channel user ID on channels defined with PUTAUT set to DEF or CTX.

If PUTAUT is set to ONLYMCA or ALTMCA for the channel, the channel user ID is ignored and the MCA user ID of the receiver or requester is used.

Table 51. User IDs checked for TCP/IP channels

Profile Name	PUTAUT option specified on receiver or requester channel							
	DEF		CTX		ONLYMCA		ALTMCA	
	1 Check	2 Checks	1 Check	2 Checks	1 Check	2 Checks	1 Check	2 Checks
ssid.ALTERNATE.USER.userid	-	-	CHL	CHL + MCA	-	-	MCA	MCA
ssid.CONTEXT	CHL	CHL + MCA	CHL	CHL + MCA	MCA	MCA	MCA	MCA
ssid.localresourcename	CHL	CHL + MCA	CHL	CHL + ALT	MCA	MCA	MCA	MCA + ALT

Key:
 ALT Alternate user ID.
 CHL Channel user ID.
 MCA MCA user ID.

User IDs checked for channels using LU 6.2

MCA user ID (MCA)

The user ID specified for the MCAUSER channel attribute at the receiver; if blank, the channel initiator address space user ID of the receiver or requester side is used.

Channel user ID (CHL)

Sender-receiver, sender-requester, and server-requester channels

If PUTAUT is set to DEF or CTX on the receiver or requester channel, the channel user ID is the user ID received from the communications system when the channel is initiated.

- If the user ID received is blank, or no user ID is received, a channel user ID of blanks is used.
- If the sending channel is on OS/390, the channel user ID received is the channel initiator address space user ID of the sender.
- If the sending channel is on a different platform (for example, AIX or HP-UX, the channel user ID received is typically provided by the USERID parameter of the channel definition.

If PUTAUT is set to ONLYMCA or ALTMCA for the channel, any user ID received from the network is ignored, and the MCA user ID of the receiver is used.

Requester-server channels

If the channel is started from the requester, there is no opportunity to receive a network user ID (the channel user ID)

If PUTAUT is set to DEF or CTX on the requester channel, the channel user ID is that of the channel initiator address space of the requester because no user ID is received from the network.

If PUTAUT is set to ONLYMCA or ALTMCA, the channel user ID is ignored and the MCA user ID of the requester is used.

Table 52. User IDs checked for LU 6.2 channels

Profile Name	PUTAUT option specified on receiver or requester channel							
	DEF		CTX		ONLYMCA		ALTMCA	
	1 Check	2 Checks	1 Check	2 Checks	1 Check	2 Checks	1 Check	2 Checks
ssid.ALTERNATE.USER.userid	-	-	CHL	CHL + MCA	-	-	MCA	MCA
ssid.CONTEXT	CHL	CHL + MCA	CHL	CHL + MCA	MCA	MCA	MCA	MCA
ssid.localresourcename	CHL	CHL + MCA	CHL	CHL + ALT	MCA	MCA	MCA	MCA + ALT
Key: ALT Alternate user ID. CHL Channel user ID. MCA MCA user ID.								

User IDs checked for client MQI requests: This section described the user IDs checked for client MQI requests issued over server-connection channels for TCP/IP and LU 6.2. The MCA user ID and channel user ID are as for the TCP/IP and LU 6.2 channels described in the previous sections.

For server-connection channels, the MCA user ID received from the client is used if the MCAUSER attribute is blank. However, for the clients that use the MQ_USER_ID environment variable to supply the user ID, it is possible that no environment variable is set. In this case, the user ID that started the server channel is used. This is the user ID assigned to the channel initiator started task by the OS/390 started procedures table.

See the *MQSeries Clients* manual for more information.

For client **MQOPEN** and **MQPUT1** requests, use the following rules to determine the profile that will be checked:

- If the request specifies alternate user authority, a check is made against the *ssid.ALTERNATE.USER.userid* profile.
- If the request specifies context authority, a check is made against the *ssid.CONTEXT* profile.
- For all **MQOPEN** and **MQPUT1** requests, a check is made against the *ssid.localresourcename* profile.

When you have determined which profiles are checked, use the following table to determine which user IDs are checked against these profiles.

Table 53. User IDs checked for LU 6.2 and TCP/IP server-connection channels								
Profile Name	PUTAUT option specified on server-connection channel							
	DEF				ONLYMCA			
	Alternate user ID specified on open?				Alternate user ID specified on open?			
	No		Yes		No		Yes	
	1 Check	2 Checks	1 Check	2 Checks	1 Check	2 Checks	1 Check	2 Checks
ssid.ALTERNATE.USER.userid	-	-	CHL	CHL + MCA	-	-	MCA	MCA
ssid.CONTEXT	CHL	CHL + MCA	CHL	CHL + MCA	MCA	MCA	MCA	MCA
ssid.localresourcename	CHL	CHL + MCA	CHL	CHL + ALT	MCA	MCA	MCA	MCA + ALT
Key: ALT Alternate user ID. CHL Channel user ID. MCA MCA user ID.								

Examples

How you use these tables is best explained by some examples.

Example 1: In a CICS transaction where the RESLEVEL profile is set to READ, determine which user IDs are checked.

Answer: First, see how many CICS user IDs are checked based on the CICS address space user ID access to the RESLEVEL profile. From Table 44 on page 428, only one user ID is checked if the RESLEVEL profile is set to READ. Then, in Table 48, look in the appropriate “1 Check” column, depending on whether the alternate user authority open option is specified. In either case, the CICS address space (ADS) user ID is checked.

Example 2: Determine the user IDs checked for the following conditions:

- The RACF access level to the RESLEVEL profile, for a CICS address space user ID, is set to NONE.
- An **MQOPEN** call is made against a queue with MQOO_OUTPUT and MQOO_PASS_IDENTITY_CONTEXT.

Answer: First, see how many CICS user IDs are checked based on the CICS address space user ID access to the RESLEVEL profile. From Table 44 on page 428, two user IDs are checked if the RESLEVEL profile is set to NONE. Then, from Table 48, these checks are carried out:

- The ssid.ALTERNATE.USER.userid profile is not checked.
- The ssid.CONTEXT profile is checked with both the CICS address space user ID and the CICS transaction user ID.
- The ssid.localresourcename profile is checked with both the CICS address space user ID and the CICS transaction user ID.

This means that four security checks are made for this **MQOPEN** call.

Blank user IDs and UACC levels

Blank user IDs can exist when a user is manipulating messages using context or alternate user security, or when MQSeries is passed a blank user ID. For example, a blank user ID is used when a message is written to the system-command input queue without context.

Note: A user ID of '* ' (that is, an asterisk character followed by seven spaces) is treated as a blank user ID.

MQSeries passes the blank user ID to RACF and a RACF undefined user is signed on. All security checks then use the universal access (UACC) for the relevant profile. Depending on how you have set your access levels, the UACC might give the undefined user a wide-ranging access.

For example, if you issue this RACF command from TSO:

```
RDEFINE MQQUEUE Q.AVAILABLE.TO.EVERYONE UACC(UPDATE)
```


you define a profile that enables both OS/390-defined user IDs (that have not been put in the access list) and the RACF undefined user ID to put messages on, and get messages from, that queue.

To protect your MQSeries subsystem from blank user IDs you must plan your access levels carefully, and limit the number of people who can use context and alternate user security. You must prevent people using the RACF undefined user ID from getting access to resources that they should not. However, at the same time, you must allow access to people with defined user IDs. To do this, you can specify a user ID of asterisk (*) in a RACF command PERMIT. For example, these RACF commands prevent the RACF undefined user ID from gaining access to the queue to put or get messages:

```
RDEFINE MQQUEUE Q.AVAILABLE.TO.RACF.DEFINED.USERS.ONLY UACC(NONE)
PERMIT Q.AVAILABLE.TO.RACF.DEFINED.USERS.ONLY CLASS(MQQUEUE) ACCESS(UPDATE) ID(*)
```

Auditing considerations

The normal RACF auditing controls are available for conducting a security audit of a queue manager. The RACF auditing can be based upon:

- User IDs
- Resource classes
- Profiles

For more details, see the *Security Server (RACF) Auditor's Guide*.

Note: Auditing degrades performance; the more auditing you implement, the more performance is degraded. This is also a consideration for the use of the RACF WARNING option.

Auditing RESLEVEL

No normal RACF audit records are taken when the RESLEVEL check is made to see what access an address space user ID has to the ssid.RESLEVEL profile. Instead, MQSeries requests that RACF create a GENERAL audit record (event number 27).

These checks are only carried out at connect time, so the overhead should be minimal.

You can report the MQSeries general audit records using the RACF report writer (RACFRW). You could use the following RACFRW commands to report the RESLEVEL access:

```
RACFRW
SELECT PROCESS
EVENT GENERAL
LIST
END
```

A sample report from RACFRW, excluding the *Date*, *Time*, and *SYSID* fields, is shown in Figure 102 on page 438.

RACF REPORT - LISTING OF PROCESS RECORDS						PAGE 4
*JOB/USER	*STEP/	--TERMINAL--				
NAME	GROUP	ID	LVL	T	L	
WS21B	MQMGRP	IGJZM000	0	27	0	JOBID=(WS21B 93.111 09:44:57),USERDATA=()
TRUSTED	USER					AUTH=(NONE),REASON=(NONE)
						SESSION=TSOLOGON,TERMINAL=IGJZM000,
						LOGSTR='CSQH RESLEVEL CHECK PERFORMED AGAINST PROFILE(QM66.RESLEVEL),
						CLASS(MQADMIN), ACCESS EQUATES TO (CONTROL)',RESULT=SUCCESS,MQADMIN

Figure 102. Sample output from RACFRW showing RESLEVEL general audit records

From checking the LOGSTR data in the output above, you can see that TSO user WS21B has CONTROL access to QM66.RESLEVEL. This means that all resource security checks will be bypassed when user WS21B access QM66 resources.

For more information about using RACFRW, see the *Security Server (RACF) Auditor's Guide*.

Statistics

MQSeries does not gather any security statistics of its own. The only statistics are those that can be created by auditing.

Chapter 27. MQSeries security implementation

The chapter covers the following subjects:

- “Security implementation checklist”
- “MQSeries security management” on page 441
- “Customizing security” on page 444
- “Security considerations for using MQSeries with CICS” on page 445
- “Security considerations for using MQSeries with IMS” on page 449
- “Security considerations for distributed queuing” on page 453
- “Security considerations for cluster support” on page 455
- “Security installation tasks” on page 456
- “Example security scenario” on page 458
- “Security problem determination” on page 464

Security implementation checklist

This section gives a step-by-step procedure you can use to work out and define the security implementation for each of your MQSeries subsystems. Refer to other sections for details, in particular “Profiles used to protect MQSeries resources” on page 404.

If you require security checking to be implemented on at least one of your MQSeries subsystems, you must first activate the RACF MQADMIN class. Then, for each MQSeries subsystem, you must decide whether you need security checking on that subsystem. If you do *not* require security checking, you must define an ssid.NO.SUBSYS.SECURITY profile in the MQADMIN class.

If you *do* require security checking, follow this checklist to implement it:

- Do you need connection security?

Yes: Define appropriate connection profiles in the MQCONN class and permit the appropriate users or groups access to these profiles.

Note: Only users of the **MQCONN** API request or CICS or IMS address space user IDs need to have access to the corresponding connection profile.

No: Define an ssid.NO.CONNECT.CHECKS profile in the MQADMIN class.

- Do you need security checking on commands?

Yes: Activate the MQCMDS class. Define appropriate command profiles in the MQCMDS class and permit the appropriate users or groups access to these profiles.

No: Define an ssid.NO.CMD.CHECKS profile in the MQADMIN class.

- Do you need security on the resources used in commands?

Yes: Ensure the MQADMIN class is active. Define appropriate profiles for protecting resources on commands in the MQADMIN class and permit the appropriate users or groups access to these profiles. Set the CMDUSER parameter in CSQ6SYSP to the default user ID to be used for command security checks.

Security checklist

- No:** Define an `ssid.NO.RESC.COMD.CHECKS` profile in the MQADMIN class.
- Do you need queue security?

Yes: Activate the MQQUEUE class. Define appropriate queue profiles in the MQQUEUE class and permit the appropriate users or groups access to these profiles.

No: Define an `ssid.NO.QUEUE.CHECKS` profile in the MQADMIN class.
 - Do you need process security?

Yes: Activate the MQPROC class. Define appropriate process profiles and permit the appropriate users or groups access to these profiles.

No: Define an `ssid.NO.PROCESS.CHECKS` profile in the MQADMIN class.
 - Do you need namelist security?

Yes: Activate the MQNLIST class. Define appropriate namelist profiles in the MQNLIST class and permit the appropriate users or groups access to these profiles.

No: Define an `ssid.NO.NLIST.CHECKS` profile in the MQADMIN class.
 - Do any users need to protect the use of the **MQOPEN** or **MQPUT1** options relating to the use of context?

Yes: Ensure the MQADMIN class is active. Define an `ssid.CONTEXT` profile in the MQADMIN class and permit the appropriate users or groups access to this profile.

No: Define an `ssid.NO.CONTEXT.CHECKS` profile in the MQADMIN class.
 - Do you need to protect the use of alternate user IDs?

Yes: Ensure the MQADMIN class is active. Define the appropriate `ssid.ALTERNATE.USER.alternateuserid` profiles and permit the required users or groups access to these profiles.

No: Define the profile `ssid.NO.ALTERNATE.USER.CHECKS` in the MQADMIN class.
 - Do you need to tailor which user IDs are to be used for resource security checks through RESLEVEL?

Yes: Ensure the MQADMIN class is active. Define an `ssid.RESLEVEL` profile in the MQADMIN class and permit the required users or groups access to the profile.

No: Ensure that no generic profiles exist in the MQADMIN class that could apply to `ssid.RESLEVEL`. Define an `ssid.RESLEVEL` profile and ensure that no users or groups have access to it.
 - Do you need to 'time out' unused user IDs from MQSeries?

Yes: Determine what timeout values you would like to use and issue the MQSeries command `alter security` to change the `TIMEOUT` and `INTERVAL` parameters.

No: Issue the MQSeries command `ALTER SECURITY` to set the `INTERVAL` value to zero.

Note: Update the CSQINP1 data set used by your subsystem so that the MQSeries command ALTER SECURITY is issued automatically at every MQSeries start up.

- Do you use distributed queuing (without CICS)?

Yes: Determine the appropriate MCAUSER attribute value for each channel, and/or provide suitable channel security exits.

- Do you use clients?

Yes: Determine the appropriate MCAUSER attribute value for each server-connection channel, and/or provide suitable channel security exits.

MQSeries security management

MQSeries uses an in-storage table to hold information relating to each user and the access requests made by each user.

To manage this table efficiently and to reduce the number of requests made from MQSeries to the external security manager (ESM), these controls are available:

- User ID timeouts
- User ID reverification
- Security refreshes
- Displaying security status

These controls are available through both the operations and control panels and MQSC.

User ID timeouts

When a user accesses an MQSeries resource, the queue manager tries to sign this user on to the queue manager—if subsystem security is active. This means that the user is authenticated to RACF. This user remains signed on to MQSeries until either the queue manager is shut down, or until the user ID is “timed out” (the authentication lapses) or reverified (reauthenticated).

When a user is timed out, the user ID is “signed off” within the queue manager and any security related information retained for this user is discarded.

Users are eligible for time out when they have not used any MQSeries resources for a predetermined amount of time. This time period is set by the MQSeries command ALTER SECURITY. For a description of the command syntax, see the *MQSeries Command Reference* manual.

Two values can be specified in the ALTER SECURITY command:

TIMEOUT The time period in minutes that an unused user ID can remain signed on within the MQSeries subsystem.

INTERVAL The time period in minutes between MQSeries checks for user IDs for which the TIMEOUT has expired.

For example, if the TIMEOUT value is 30 and the INTERVAL value is 10, every 10 minutes MQSeries checks for user IDs that have not been used for 30 minutes. If such a user ID is found, that user ID is signed off within the queue manager.

Security management

Note: The signing on and off of the user within the queue manager is transparent to the application program and to the end user.

If you do not want to time out user IDs, set the INTERVAL value to zero.

Note: If you use values for INTERVAL or TIMEOUT other than the defaults, you must re-enter the command at every MQSeries startup. You can do this automatically by putting the MQSeries command ALTER SECURITY in the CSQINP1 data set for that queue manager.

User ID reverification

If the RACF definition of a user who is using MQSeries resources has been changed—for example, by connecting the user to a new group—you can tell the queue manager to sign this user on again the next time it tries to access an MQSeries resource. You can do this by using the MQSeries command RVERIFY SECURITY. For example:

- User HX0804 is getting and putting messages to the PAYROLL queues on queue manager PRD1. However HX0804, now requires access to some of the PENSION queues on the same queue manager (PRD1).
- The data security administrator connects user HX0804 to the RACF group that allows access to the PENSION queues.
- So that HX0804 can access the PENSION queues immediately—that is, without shutting down queue manager PRD1, or waiting for HX0804 to time out—you must use the MQSeries command:

```
RVERIFY SECURITY(HX0804)
```

Note: If you turn off user ID timeout for long periods of time (days or even weeks), while the queue manager is running, you must remember to perform an RVERIFY SECURITY for any users that have been revoked or deleted in that time.

Security refreshes

Whenever you change a RACF profile that is used by MQSeries security, you must tell the corresponding queue manager to refresh its in-storage list of RACF profiles. You can do this by using the MQSeries command REFRESH SECURITY. For a description of command syntax, see the *MQSeries Command Reference* manual.

You must also issue normal RACF refresh commands if you change generic profiles. For example, SETROPTS GENERIC(classname) REFRESH. You must also issue the MQSeries command, REFRESH SECURITY, if you change a profile in any of these RACF classes:

- MQADMIN
- MQQUEUE
- MQNLIST
- MQPROC

These are the only classes affected by the MQSeries command REFRESH SECURITY. You do *not* need to use REFRESH SECURITY if you change a profile in either the MQCONN or MQCMDS classes.

Note: A refresh of MQADMIN is not required if you change a RESLEVEL security profile.

For performance reasons, use REFRESH SECURITY as infrequently as possible, ideally at off-peak times. You can minimize the number of security refreshes by connecting users to RACF groups that are already in the access list for MQSeries profiles, rather than putting individual users in the access lists. In this way, you change the user rather than the resource profile. You can also RVERIFY SECURITY the appropriate user instead of refreshing security.

As an example of REFRESH SECURITY, suppose you define the new profiles to protect access to queues starting with INSURANCE.LIFE on queue manager PRMQ. You use these RACF commands:

```
RDEFINE MQQUEUE PRMQ.INSURANCE.LIFE.** UACC(NONE)
PERMIT PRMQ.INSURANCE.LIFE.** ID(LIFEGRP) ACCESS(UPDATE)
```

Because these profiles are generic, you must tell RACF to refresh the generic profiles for MQQUEUE. For example:

```
SETROPTS GENERIC(MQQUEUE) REFRESH
```

Then you must use this command to tell queue manager PRMQ that the queue profiles have changed:

```
REFRESH SECURITY(MQQUEUE)
```

If you issue a REFRESH SECURITY(*) or a REFRESH SECURITY(MQADMIN), the status of the security switches in the MQADMIN class are also checked. This means you can activate new security types, or de-activate them without having to restart the queue manager.

Displaying security status

To display the status of the security switches, and other security controls, you can issue the MQSeries command DISPLAY SECURITY. For a description of the command syntax, see the *MQSeries Command Reference* manual.

Figure 103 shows a typical output of the MQSeries command DISPLAY SECURITY ALL. The example shows that the queue manager that replied to the command has all MQSeries security active, except for namelist security. It also shows that user ID timeouts are active, and that every 12 minutes the queue manager checks for user IDs that have not been used in this queue manager for 54 minutes and removes them.

Note: This command shows the current security status. It does not necessarily reflect the current status of the switch profiles defined to RACF, or the status of the RACF classes. For example, the switch profiles might have been changed since the last restart of this queue manager or REFRESH SECURITY command.

Customizing security

```
CSQH015I +cpf SECURITY TIMEOUT = 54 MINUTES
CSQH016I +cpf SECURITY INTERVAL = 12 MINUTES
CSQH002I +cpf CSQHPDTC SUBSYSTEM SECURITY SWITCH SET ON
CSQH002I +cpf CSQHPDTC CONNECTION SECURITY SWITCH SET ON
CSQH002I +cpf CSQHPDTC COMMAND SECURITY SWITCH SET ON
CSQH002I +cpf CSQHPDTC CONTEXT SECURITY SWITCH SET ON
CSQH002I +cpf CSQHPDTC ALTERNATE USER SECURITY SWITCH SET ON
CSQH002I +cpf CSQHPDTC PROCESS SECURITY SWITCH SET ON
CSQH002I +cpf CSQHPDTC NAMELIST SECURITY SWITCH SET OFF
CSQH002I +cpf CSQHPDTC QUEUE SECURITY SWITCH SET ON
CSQH002I +cpf CSQHPDTC COMMAND RESOURCES SECURITY SWITCH SET ON
CSQ9022I +cpf CSQHPDTC ' DISPLAY SECURITY' NORMAL COMPLETION
```

Figure 103. Typical output from the MQSeries command `DISPLAY SECURITY`

Customizing security

If you want to change the way MQSeries security operates, you must do this via the SAF exit (ICHRFR00), or exits in your external security manager. To find out more about RACF exits, see the *Security Server (RACF) External Security Interface (RACROUTE) Macro Reference* manual.

Note: Because MQSeries optimizes calls to the ESM, RACROUTE requests will not be made on, for example, every open for a particular queue by a particular user.

Security considerations for using MQSeries with CICS

The CICS adapter provides this information to MQSeries specifically for use in MQSeries security:

- Whether CICS resource-level security is active for this transaction—as specified on the RESSEC or RSLC operand of the RDO TRANSACTION definition.
- User IDs.

For terminal tasks where a user has not signed on, the user ID is the CICS user ID associated with the terminal and is either:

- The default CICS user ID as specified on the CICS parameter DFLTUSER SIT
- A preset security user ID specified on the terminal definition

For non-terminal tasks, the CICS adapter tries to get a user ID with an EXEC CICS ASSIGN command. If this is unsuccessful, the adapter tries to get the user ID using EXEC CICS INQUIRE TASK. If security is active in CICS, and the non-terminal attached transaction is defined with CMDSEC(YES), the CICS adapter passes a user ID of blanks to MQSeries.

For more information about RACF security management in the CICS environment, see the *CICS RACF Security Guide*.

Controlling the security of CICS transactions supplied by MQSeries

The CKTI and CKAM transactions are designed to be run without a terminal; no user should have access to these transactions. These transactions are examples of what the *CICS RACF Security Guide* calls “category 1 transactions”. For information about to set these transactions up in CICS and RACF, see the information about category 1 transactions in the *CICS RACF Security Guide*.

If you want a user to administer the CICS adapter, you must authorize them to these transactions:

Transaction	What it does
CKQC	Controls the CICS adapter functions
CKBM	Controls the CICS adapter functions
CKRT	Controls the CICS adapter functions
CKCN	Connect
CKSD	Disconnect
CKRS	Statistics
CKDP	Full screen display
CKDL	Line mode display
CKSQ	CKTI START/STOP

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

You should define these transactions to CICS with RESSEC(NO) and CMDSEC(NO). For more details, see the *CICS RACF Security Guide*. For information about the security of the CICS transactions supplied by MQSeries for remote queuing, see the *MQSeries Intercommunication* manual.

CICS adapter user IDs

The user ID associated with CICS adapter is that of the MQSeries-supplied task initiator transaction, CKTI. This section describes some of the implications of this.

User ID checking for MQSeries resources during PLTPI and PLTSD

If an MQSeries resource is accessed during the CICS PLTPI phase, the user ID passed to MQSeries is blanks. If an MQSeries resource is accessed during the CICS PLTSD phase, the user ID passed to MQSeries is the user ID associated with the shutdown transaction.

If CKTI is started during the CICS PLTPI phase, the user ID of the CKTI task is the CICS sysidnt. This means that a user ID with the same name as the CICS sysidnt must be defined and given access to the required MQSeries resources, for example, initiation queues.

Terminal user IDs

If CKTI is started from a terminal from the CKQC transaction or a user-written program that links to CSQCSSQ, the user ID that CKTI uses is the same as the user ID of the terminal that started CKTI.

Automating starting of CKTI

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

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

However, when the CICS adapter alert monitor reconnects CICS to MQSeries, after, for example, an MQSeries restart, only the CKTI specified at the initial MQSeries connection is restarted. You must automate starting any extra CKTIs yourself.

Propagating the CKTI user ID to other CICS transactions

If CKTI starts other CICS transactions, for example, message channel agents (MCAs) or user-written CICS applications, the user ID of CKTI is propagated to these applications. For example, if CKTI is running under user ID CIC1 and a trigger event occurs which requires the sender MCA transaction, CKSG, to be started, the CKSG transaction also runs under user ID CIC1. Therefore user ID CIC1 must have access to the required transmission queue.

Security considerations for the CICS bridge

When you run the CICS bridge, you can specify the level of authentication you want to take place. If requested, the bridge checks the user ID and password extracted from the MQSeries request message before running the CICS program named in the request message.

Notes:

1. If you have not specified a user ID or password in a message, the bridge task runs with the LOCAL level of authentication, even if you started the bridge monitor with a different authentication option.
2. The options that include password (or passticket) validation require a CICS bridge header (MQCIH) to be provided. See the *MQSeries Application Programming Reference* manual for more information about the MQCIH header.

The level of authentication you can use is described below:

LOCAL This is the default. CICS programs run by the bridge task are started with the CICS DFLTUSER user ID, therefore run with the authority associated with this user ID. There is no checking of user IDs or passwords. If a CICS program is run that tries to access protected resources, it will probably fail.

IDENTIFY When you start the monitor task with the IDENTIFY authentication option, the bridge task is started with the user ID specified in the message (MQMD). CICS programs run by the bridge run with the user ID extracted from the MQMD. There is no password checking, the user ID is treated as trusted.

VERIFY_UOW When you start the monitor task with the VERIFY_UOW authentication option, the monitor task checks the user ID and password by issuing the EXEC CICS VERIFY PASSWORD command before starting the bridge task. CICS programs run by the bridge run with the user ID extracted from the MQMD. If the user ID or password is invalid, the request fails with return code MQCRC_SECURITY_ERROR.

VERIFY_ALL This is the same as VERIFY_UOW except that the bridge task checks the user ID and password in **every** message. This is not applicable for 3270 transactions.

If you have not specified a user ID in a message, or you have not provided a password, the CICS program started by the CICS bridge runs with the user ID set to the CICS DFLTUSER, regardless of the option requested. If you want more than one level of authentication checking performed, run a monitor task for each level you need.

Table 54 and Table 55 on page 448 summarize the level of authority of the bridge monitor and the bridge tasks, and the use of the MQMD user ID.

Monitor started by	At a signed on terminal	Monitor authority
From a terminal or EXEC CICS LINK within a program	Yes	Signed on user ID

<i>Table 54 (Page 2 of 2). CICS bridge monitor security</i>		
Monitor started by	At a signed on terminal	Monitor authority
From a terminal or EXEC CICS LINK within a program	No	CICS default user ID
EXEC CICS START with user ID	–	User ID from START
EXEC CICS START without user ID	–	CICS default user ID
The MQSeries trigger monitor CKTI	–	CICS default user ID

<i>Table 55. CICS bridge task security</i>	
AUTH	Bridge task authority
LOCAL	CICS default user ID
IDENTIFY	MQMD UserIdentifier
VERIFY_UOW	MQMD UserIdentifier
VERIFY_ALL	MQMD UserIdentifier

The options IDENTIFY, VERIFY_UOW, and VERIFY_ALL need the user ID of the bridge monitor defined to RACF as a surrogate of all the user IDs used in request messages. This is in addition to the user ID in the message being defined to RACF. (A surrogate user is one who has the authority to start work on behalf of another user, without knowing the other user's password.)

For more information on surrogate user security, see the *CICS RACF Security Guide*.

Note: When IDENTIFY security is being used, you might see abend AICO for CKBP if you try to run with a user ID that has been revoked. The error reply will have return code MQCRC_BRIDGE_ERROR with reason MQFB_CICS_BRIDGE_FAILURE.

Authority

Components of the bridge need authority to either put to or get from the various MQSeries queues. In summary:

- The monitor and all bridge tasks need authority to get messages from the bridge request queue.
- A bridge tasks need authority to put messages to its reply-to queue.
- To ensure any error replies are received, the monitor should have authority to put messages to all reply-to queues.
- Bridge tasks should have authority to put messages to the dead-letter queue.
- The monitor needs authority to put messages to the dead-letter queue, unless you want the bridge to stop if an error occurs.

See Table 54 on page 447 to determine the correlation between user IDs and authority.

Security considerations for using MQSeries with IMS

Using the OPERCMDS class

If you are using RACF to protect resources in the OPERCMDS class, ensure that your MQSeries system has authority to issue the MODIFY command to any IMS system to which it can connect.

Security considerations for the IMS bridge

There are four aspects that you must consider when deciding your security requirements for the IMS bridge, these are:

- What security authorization is needed to connect MQSeries to IMS (“Connecting to IMS”)
- How much security checking is performed on applications using the bridge to access IMS (“Application access control” on page 450)
- Which IMS resources these applications are allowed to use (“Security checking on IMS” on page 451)
- What authority is to be used for messages that are put and got by the bridge (“Security checking done by the bridge” on page 452)

When you define your security requirements for the IMS bridge you must consider the following:

- Messages passing across the bridge might have originated from applications on platforms that do not offer strong security features
- Messages passing across the bridge might have originated from applications that are not controlled by the same enterprise or organization

Connecting to IMS

The IMS bridge is an OTMA client. The connection to IMS operates under the user ID of the MQSeries for OS/390 address space. This is normally defined as a member of the started task group. This user ID must be granted access to the OTMA group (unless the /SECURE OTMA setting is NONE). To do this, define the following profile in the FACILITY class:

```
IMSXCF.xcfgname.xcfmname
```

Where `xcfname` is the XCF group name and `xcfname` is the XCF member name of MQSeries.

You must give your MQSeries subsystem user ID read access to this profile.

Notes:

1. If you change the authorities in the FACILITY class, you must issue the RACF command `SETROPTS RACLIST(FACILITY) REFRESH` to activate the changes.
2. If profile `qmgr.NO.SUBSYS.SECURITY` exists in the MQADMIN class, no user ID will be passed to IMS and the connection will fail unless the /SECURE OTMA setting is NONE.

Application access control

For each IMS system that the IMS bridge connects to, you can define the following RACF profile in the FACILITY class to determine how much security checking is performed for each message passed to the IMS system.

```
IMSXCF.xcfgname.xcfmname
```

Where *xcfname* is the XCF group name and *xcfname* is the XCF member name for IMS. (You need to define a separate profile for each IMS system.)

The access level you allow for the MQSeries subsystem user ID in this profile is returned to MQSeries when the IMS bridge connects to IMS, and indicates the level of security that is required on subsequent transactions. For subsequent transactions, MQSeries requests the appropriate services from RACF and, where the user ID is authorized, passes the message to IMS.

OTMA does not support the IMS /SIGN command; however, MQSeries allows you to set the access checking for each message to enable implementation of the necessary level of control.

The following access level information can be returned:

NONE or NO PROFILE FOUND

This indicates that maximum security is required, that is, authentication is required for every transaction. A check is made to verify that the user ID specified in the *UserIdentifier* field of the MQMD structure, and the password or passticket in the *Authenticator* field of the MQIHL structure are known to RACF, and are a valid combination. A Utoken is created with a password or passticket, and passed to IMS; the Utoken is not cached.

Note: If profile qmgr.NO.SUBSYS.SECURITY exists in the MQADMIN class, this level of security overrides whatever is defined in the profile.

READ

This indicates that the same authentication is to be performed as above under the following circumstances:

- The first time that a specific user ID is encountered
- When the user ID has been encountered before but the cached Utoken was not created with a password or passticket

MQSeries requests a Utoken if required, and passes it to IMS.

Note: If a request to reverify security has been actioned, all cached information is lost and a Utoken is requested the first time each user ID is subsequently encountered.

UPDATE

A check is made that the user ID in the *UserIdentifier* field of the MQMD structure is known to RACF.

A Utoken is built and passed to IMS; the Utoken is cached.

CONTROL/ALTER

These indicate that no security Utokens need to be provided for any user IDs for this IMS system. (You would probably only use this for development and test systems.)

Notes:

1. This access is defined when MQSeries connects to IMS, and lasts for the duration of the connection. To change the security level, the access to the security profile must be changed and then the bridge stopped and restarted (for example, by stopping and restarting OTMA).
2. If you change the authorities in the FACILITY class, you must issue the RACF command SETROPTS RACLIST(FACILITY) REFRESH to activate the changes.
3. You can use a password or a passticket, but you must remember that the IMS bridge does not encrypt data. For information about using passtickets, see "Using RACF passtickets in the IMS header" on page 452.
4. Some of the above might be affected by security settings in IMS, using the /SECURE OTMA command.
5. Cached Utoken information is held for the duration defined by the INTERVAL and TIMEOUT parameters of the MQSeries ALTER SECURITY command.

Security checking on IMS

Each MQSeries message that passes across the bridge contains the following security information:

- A user ID contained in the *UserIdentifier* field of the MQMD structure
- The security scope contained in the *SecurityScope* field of the MQIIH structure (if the MQIIH structure is present)
- A Utoken (unless the MQSeries sub system has CONTROL or ALTER access to the relevant IMSXCF.xcfname.xcfmname profile)

The security checks made depend on the setting by the IMS command /SECURE OTMA, as follows:

/SECURE OTMA NONE

No security checks are made for the transaction.

/SECURE OTMA CHECK

The *UserIdentifier* field of the MQMD structure is passed to IMS for transaction or command authority checking.

An ACEE (Accessor Environment Element) is built in the IMS control region.

/SECURE OTMA FULL

The *UserIdentifier* field of the MQMD structure is passed to IMS for transaction or command authority checking.

An ACEE is built in the IMS dependent region as well as the IMS control region.

/SECURE OTMA PROFILE

The *UserIdentifier* field of the MQMD structure is passed to IMS for transaction or command authority checking

The *SecurityScope* field in the MQIIH structure is used to determine whether to build an ACEE in the IMS dependent region as well as the control region.

Notes:

1. If you change the authorities in the TIMS or CIMS class, or the associated group classes GIMS or DIMS, you must issue the following IMS commands to activate the changes:
 - /MODIFY PREPARE RACF
 - /MODIFY COMMIT
2. If you do not use /SECURE OTMA PROFILE, any value specified in the *SecurityScope* field of the MQIHL structure is ignored.

Security checking done by the bridge

When the bridge puts or gets a message, the following authorities are used:

Getting a message from the bridge queue

No security checks are performed.

Putting an exception, or COA report message

Uses the authority of the user ID in the *UserIdentifier* field of the MQMD structure.

Putting a reply message

Uses the authority of the user ID in the *UserIdentifier* field of the MQMD structure of the original message

Putting a message to the dead-letter queue

No security checks are performed.

Notes:

1. If you change the MQSeries class profiles, you must issue the MQSeries command REFRESH SECURITY(*) to activate the changes.
2. If you change the authority of a user, you must issue the MQSeries command RVERIFY SECURITY to activate the change.

Using RACF passtickets in the IMS header

If you want to use a passticket instead of a password in the IMS header (MQIHL), you should use an application name as if you were creating a passticket for an OS/390 batch job. That is, the APPL field should be of the form MVSxxxx, where xxxx is the SMFID of the OS/390 system on which the target queue manager runs.

A passticket is built from a user ID, the target application name (APPL), and a secret key. It is an 8-byte value containing uppercase alphabetic and numeric characters. It can be used only once, and is valid for a 20 minute period centered on the time that it was created. For full information about passtickets, see the *Security Server (RACF) Security Administrator's Guide*.

Passtickets in IMS headers are given to RACF by MQSeries, not IMS.

Security considerations for distributed queuing

This section discusses security considerations for the non-CICS mover. If you are using the CICS mover, see “Security considerations for distributed queuing (using CICS ISC)” on page 482. If you are using clustering, you should also read “Security considerations for cluster support” on page 455.

If you are using resource security, you should consider the following if you are using distributed queuing:

- The channel initiator address space needs RACF UPDATE access to these system queues:
 - SYSTEM.CHANNEL.INITQ
 - SYSTEM.CHANNEL.SYNCQ
 - SYSTEM.COMMAND.INPUT
 - SYSTEM.ADMIN.CHANNEL.EVENT

and to all the user destination queues, and also the dead-letter queue (but see “Dead-letter queue security” on page 411).

- The channel initiator address space (and the user ID specified by MCAUSER on each channel if you have a RESLEVEL profile specifying that two user IDs are to be checked) need ALTER access to all the user transmission queues.
- The channel initiator address space (or the MCA user ID if one has been specified) also need RACF CONTROL access to the ssid.CONTEXT profile in the MQADMIN class, and, depending on the RESLEVEL profile, the network-received user ID might also need CONTROL access to this profile. See “Profiles for context security” on page 417 and “RESLEVEL and channel initiator connections” on page 427 for more information.

If you are using the CSQINPX input data set, the channel initiator also needs READ access to CSQINPX, and UPDATE access to data set CSQOUTX and dynamic queues SYSTEM.CSQXCMD.*.

- The channel initiator address space connection requests use a connection type of CHIN, for which appropriate access security must be set, see “Connection security profiles for distributed queuing” on page 405.
- The channel initiator address space needs appropriate access to queue manager data sets, see “Authorizing access to data sets” on page 457.
- The distributed queuing commands (for example, DEFINE CHANNEL, START CHINIT, START LISTENER, and so on) should have appropriate command security set, see Table 42 on page 423.
- Channels, particularly receivers and server-connections, need appropriate security to be set up; see “User IDs for security checking” on page 430 for more information. See also the *MQSeries Clients* manual for information about server-connection security.
- The user ID specified by MCAUSER on each channel (and if you have a RESLEVEL profile specifying that two user IDs are to be checked, the network-received user ID if there is one being used), need the following:
 - RACF UPDATE access to the appropriate destination queues and the dead-letter queue
 - RACF CONTROL access to the ssid.CONTEXT profile if context checking is performed at the receiver

Distributed queuing security

- For clients, the appropriate RACF access to the resources to be used.
- Set appropriate APPC security if you are using the LU 6.2 transmission protocol. (Use the APPCLU RACF class for example.) For information about setting up security for APPC, see the following manuals:
 - *MVS Planning: APPC/MVS Management*
 - *APPC Security: MVS/ESA, CICS/ESA, and OS/2* (redbook)

Outbound transmissions use the “SECURITY(SAME)” APPC option. This means that the user ID of the channel initiator address space and its default profile (RACF GROUP) are flowed across the network to the receiver with an indicator that the user ID has already been verified (ALREADYV).

If the receiving side is also OS/390, the user ID and profile are verified by APPC and the user ID is presented to the receiver channel and used as the network user ID.

In an environment where the queue manager is using APPC to communicate with another queue manager on the same or another OS/390 system, you need to ensure that either:

- The VTAM definition for the communicating LU specifies SETACPT(ALREADYV)
- There is a RACF APPCLU profile for the connection between LUs that specifies CONVSEC(ALREADYV)
- If the RACF access level that either the channel user ID or network-received user ID has to a destination queue is changed, this change will only take effect for new object handles (that is, new **MQOPENS**) for the destination queue. The times when MCAs open and close queues is variable; if a channel is already running when such an access change is made, the MCA can continue to put messages on the destination queue using the existing security access of the user ID(s) rather than the updated security access. To avoid this, you should stop and re-start the channels to enforce the updated access level.

Security considerations for cluster support

This section discusses the security considerations for cluster support.

You can use the MCA user ID and security exits to authenticate cluster channels (as with conventional channels). The security exit on the cluster-receiver channel must check that the queue manager is permitted access to the server queue manager's clusters. You can start to use MQSeries cluster support without having to change your existing queue access security, however you must allow other queue managers in the cluster to write to the `SYSTEM.CLUSTER.COMMAND.QUEUE` if they are to join the cluster.

MQSeries cluster support does not provide a mechanism to limit a member of a cluster to the client role only. As a result, you must be sure that you trust any queue managers that you allow into the cluster. If any queue manager in the cluster creates a queue with a particular name, it will be able to receive messages for that queue, regardless of whether the application putting messages to that queue intended this or not.

To restrict the membership of a cluster, you need to take the same action that you would take to prevent queue managers connecting to receiver channels. You can achieve this by writing a security exit program on the receiver channel or by writing an exit program to prevent unauthorized queue managers from writing to the `SYSTEM.CLUSTER.COMMAND.QUEUE`.

Note: It is not advisable to permit applications to open the `SYSTEM.CLUSTER.TRANSMIT.QUEUE` directly, just as it is not advisable to permit an application to open any other transmission queue directly.

If you are using resource security you should consider the following in addition to the considerations discussed in “Security considerations for distributed queuing” on page 453:

- The channel initiator needs RACF ALTER access to the following system queues:
 - `SYSTEM.CLUSTER.COMMAND.QUEUE`
 - `SYSTEM.CLUSTER.TRANSMIT.QUEUE`.

and UPDATE access to `SYSTEM.CLUSTER.REPOSITORY.QUEUE`

It also needs READ access to any namelists used for clustering.

- The cluster support commands (REFRESH and RESET CLUSTER, SUSPEND and RESUME QMGR) should have appropriate command security set (as described in Table 42 on page 423).

Security installation tasks

When MQSeries is first installed, you must perform these security-related tasks:

1. Set up MQSeries data set and system security by:
 - Authorizing the queue manager started-task procedure xxxxMSTR and the distributed queuing started-task procedure xxxxCHIN to run under RACF.
 - Authorizing access to queue manager data sets.
2. Set up RACF definitions for MQSeries security.

Setting up MQSeries data set and system security

The possible users of MQSeries data sets include:

- The queue manager itself.
- MQSeries administrators who need to create MQSeries data sets, run utility programs, and so on.
- Application programmers, who need to use the MQSeries-supplied copybooks, include data sets, macros, and so on.
- Applications involving one or more of the following:
 - Batch jobs
 - TSO users
 - CICS regions
 - IMS regions

For all of these potential users, protect the MQSeries data sets with RACF.

You must also control access to all your 'CSQINP' data sets.

RACF authorization of started-task procedures

Some MQSeries data sets should be for the exclusive use of the queue manager. If you protect your MQSeries data sets using RACF, you must also authorize the queue manager started-task procedure xxxxMSTR, and the distributed queuing started-task procedure xxxxCHIN, using RACF. To do this, use either:

- The STARTED class.
- The started procedures table (ICHRIN03).

(Any changes you make to the RACF started procedures table require that you IPL your OS/390 system before the changes can take effect.)

For more information, see the *Security Server (RACF) System Programmer's Guide*.

The RACF user ID identified must have the required access to the data sets in the started-task procedure. For example, if you associate a queue manager started task procedure called CSQ1MSTR with the RACF user ID QMGRCSQ1, the user ID QMGRCSQ1 must have access to the OS/390 resources accessed by the CSQ1 queue manager.

The RACF user IDs associated with the queue manager and channel initiator started task procedures should not have the TRUSTED attribute set.

Authorizing access to data sets

The MQSeries data sets should be protected so that no unauthorized user can run a queue manager instance, or gain access to any queue manager data. To do this, use normal OS/390 RACF data set protection. For more information, see the *Security Server (RACF) Security Administrator's Guide*.

Table 56 summarizes the RACF access that the queue manager started task procedure must have to the different data sets.

RACF access	Data sets
READ	<ul style="list-style-type: none"> • thlqual.SCSQAUTH and thlqual.SCSQANLx (where x is the language letter for your national language). • The data sets referred to by CSQINP1, CSQINP2 and CSQXLIB in the queue manager's started task procedure.
UPDATE	<ul style="list-style-type: none"> • All page sets and log and BSDS data sets.
ALTER	<ul style="list-style-type: none"> • All archive data sets.

Table 57 summarizes the RACF access that the started task procedure for distributed queuing must have to the different data sets.

RACF access	Data sets
READ	<ul style="list-style-type: none"> • thlqual.SCSQAUTH, thlqual.SCSQANLx (where x is the language letter for your national language), and thlqual.SCSQMVR1 or thlqual.SCSQMVR2. • LE/370 library data sets. • The data sets referred to by CSQXLIB and CSQINPX in the distributed queuing started task procedure.
UPDATE	<ul style="list-style-type: none"> • Data sets CSQOUTX and CSQSNAP • Dynamic queues SYSTEM.CSQXCMD.★

Example security scenario

This section describes an example security scenario, showing the security settings required. The scenario uses 2 queue managers on OS/390, called QM1 and QM2.

An application uses the **MQPUT1** call to put messages to queues on queue manager QM1. Some of the messages are then forwarded to queues on QM2, using TCP and LU 6.2 channels. The application could be a batch application or a CICS application, and the messages are put using the **MQPMO_SET_ALL_CONTEXT** option. This is illustrated in Figure 104.

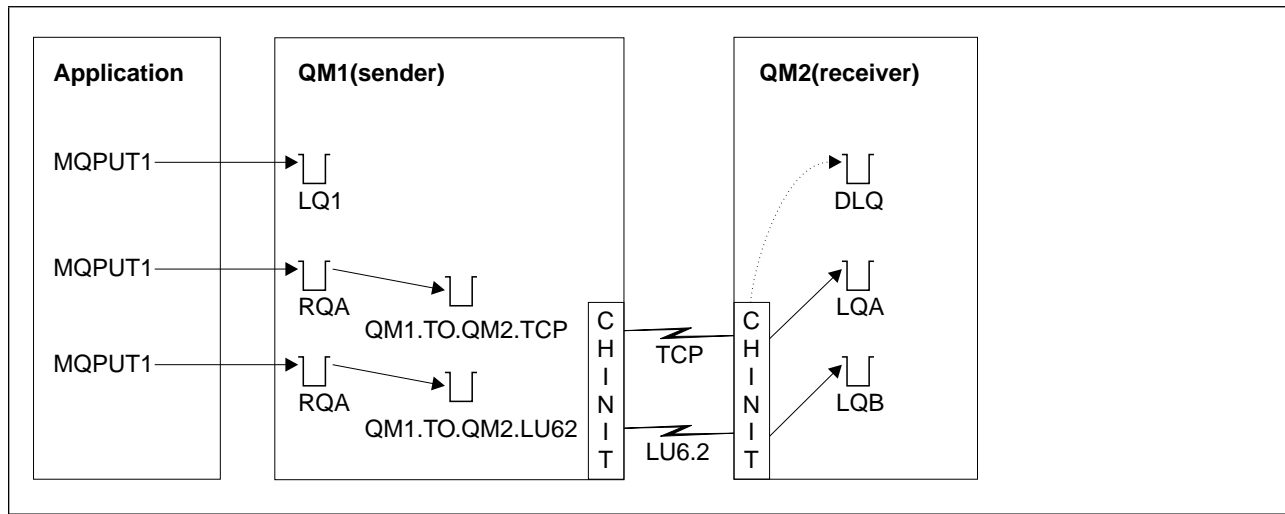


Figure 104. Example security scenario

The following assumptions are made about the queue managers:

- All the required MQSeries definitions have been predefined or have been made through the CSQINP2 data set processed at queue manager startup.

If they have not, you will need the appropriate access authority to the commands needed to define these objects.

- All the RACF profiles required have been defined and appropriate access authorities have been granted, before the queue manager and channel initiators started.

If they have not, you will need the appropriate authority to issue the RACF commands required to define all the profiles needed and grant the appropriate access authorities to those profiles. You will also need the appropriate authority to issue the MQSeries security commands to start using the new security profiles.

Security switch settings

The following security switches are set for both queue managers:

- Subsystem security on
- Queue security on
- Alternate user security on
- Context security on
- Process security off
- Namelist security off
- Connection security on

- Command security on
- Command resource security on

The following profiles are defined in the MQADMIN class to turn process and namelist security off:

```
QM1.NO.PROCESS.CHECKS
QM1.NO.NLIST.CHECKS
QM2.NO.PROCESS.CHECKS
QM2.NO.NLIST.CHECKS
```

MQSeries object definitions

The following objects are defined on the two queue managers. The definitions use the defaults supplied with MQSeries, unless otherwise stated.

Queue manager QM1

The following queues are defined on queue manager QM1:

LQ1 A local queue.

RQA A remote queue definition, with the following attributes:

- RNAME(LQA)
- RQMNAME(QM2)
- XMITQ(QM1.TO.QM2.TCP)

RQB A remote queue definition, with the following attributes:

- RNAME(LQB)
- RQMNAME(QM2)
- XMITQ(QM1.TO.QM2.LU62)

QM1.TO.QM2.TCP

A transmission queue.

QM1.TO.QM2.LU62

A transmission queue.

The following channels are defined on QM1:

QM1.TO.QM2.TCP

A sender channel definition, with the following attributes:

- Channel name = QM1.TO.QM2.TCP
- CHLTYPE(SDR)
- TRPTYPE(TCP)
- XMITQ(QM1.TO.QM2.TCP)
- CONNAME(QM2TCP)

QM1.TO.QM2.LU62

A sender channel definition, with the following attributes:

- Channel name = QM1.TO.QM2.LU62
- CHLTYPE(SDR)
- TRPTYPE(LU62)
- XMITQ(QM1.TO.QM2.LU62)
- CONNAME(QM2LU62)

Example security scenario

(See “Security considerations for distributed queuing” on page 453 for information about setting up APPC security.)

Queue manager QM2

The following queues have been defined on queue manager QM2:

LQ1 A local queue.

LQB A local queue.

DLQ A local queue that is used as the dead-letter queue.

The following channels have been defined on QM2:

QM1.TO.QM2.TCP

A receiver channel definition, with the following attributes:

- Channel name = QM1.TO.QM2.TCP
- CHLTYPE(RCVR)
- TRPTYPE(TCP)
- PUTAUT(CTX)
- MCAUSER(MCATCP)

QM1.TO.QM2.LU62

A receiver channel definition, with the following attributes:

- Channel name = QM1.TO.QM2.LU62
- CHLTYPE(RCVR)
- TRPTYPE(LU62)
- PUTAUT(CTX)
- MCAUSER(MCALU62)

(See “Security considerations for distributed queuing” on page 453 for information about setting up APPC security.)

User IDs used in scenarios

The following user IDs are used:

BATCHID Batch application (Job or TSO ID)

MSGUSR *UserIdentifier* in MQMD (context user ID)

MOVER1 QM1 channel initiator address space user ID

MOVER2 QM2 channel initiator address space user ID

MCATCP MCAUSER specified on the TCP receiver channel definition

MCALU62 MCAUSER specified on the LU 6.2 receiver channel definition

CICSAD1 CICS address space ID

CICSTX1 CICS task user ID

Security profiles and accesses required

Table 58 through Table 62 on page 463 show the security profiles that are required to enable the scenario to work:

Class	Profile	User ID	Access
MQCONN	QM1.CHIN	MOVER1	READ
MQADMIN	QM1.RESLEVEL	BATCHID	NONE
MQADMIN	QM1.RESLEVEL	CICSAD1	NONE
MQADMIN	QM1.CONTEXT	MOVER1	CONTROL

Table 58 (Page 2 of 2). Security profiles for the example scenario

Class	Profile	User ID	Access
MQQUEUE	QM1.SYSTEM.COMMAND.INPUT	MOVER1	UPDATE
MQQUEUE	QM1.SYSTEM.CHANNEL.SYNCQ	MOVER1	UPDATE
MQQUEUE	QM1.SYSTEM.CHANNEL.INITQ	MOVER1	UPDATE
MQQUEUE	QM1.SYSTEM.COMMAND.REPLY.MODEL	MOVER1	UPDATE
MQQUEUE	QM1.SYSTEM.ADMIN.CHANNEL.EVENT	MOVER1	UPDATE
MQQUEUE	QM1.QM1.TO.QM2.TCP	MOVER1	ALTER
MQQUEUE	QM1.QM1.TO.QM2.LU62	MOVER1	ALTER
MQCONN	QM2.CHIN	MOVER2	READ
MQADMIN	QM2.RESLEVEL	MOVER2	NONE
MQADMIN	QM2.CONTEXT	MOVER2	CONTROL
MQQUEUE	QM2.SYSTEM.COMMAND.INPUT	MOVER2	UPDATE
MQQUEUE	QM2.SYSTEM.CHANNEL.SYNCQ	MOVER2	UPDATE
MQQUEUE	QM2.SYSTEM.CHANNEL.INITQ	MOVER2	UPDATE
MQQUEUE	QM2.SYSTEM.COMMAND.REPLY.MODEL	MOVER2	UPDATE
MQQUEUE	QM2.SYSTEM.ADMIN.CHANNEL.EVENT	MOVER2	UPDATE
MQQUEUE	QM2.DLQ	MOVER2	UPDATE

Security profiles required for a batch application

The batch application runs under user ID BATCHID on QM1. It connects to queue manager QM1 and puts messages to the following queues:

- LQ1
- RQA
- RQB

It uses the MQPMO_SET_ALL_CONTEXT and MQPMO_ALTERNATE_USER_AUTHORITY options. The alternate user ID found in the *UserIdentifier* field of the message descriptor (MQMD) is MSGUSR.

The following profiles are required on queue manager QM1:

Table 59. Sample security profiles for the batch application on queue manager QM1

Class	Profile	User ID	Access
MQCONN	QM1.BATCH	BATCHID	READ
MQADMIN	QM1.CONTEXT	BATCHID	CONTROL
MQQUEUE	QM1.LQ1	BATCHID	UPDATE
MQQUEUE	QM1.RQA	BATCHID	UPDATE
MQQUEUE	QM1.RQB	BATCHID	UPDATE

Example security scenario

The following profiles are required on queue manager QM2 for messages put to queue RQA on queue manager QM1 (for the TCP channel):

<i>Table 60. Sample security profiles for queue manager QM2 using TCP</i>			
Class	Profile	User ID	Access
MQADMIN	QM2.ALTERNATE.USER.MSGUSR	MCATCP	UPDATE
MQADMIN	QM2.ALTERNATE.USER.MSGUSR	MOVER2	UPDATE
MQADMIN	QM2.CONTEXT	MCATCP	CONTROL
MQADMIN	QM2.CONTEXT	MOVER2	CONTROL
MQQUEUE	QM2.LQA	MOVER2	UPDATE
MQQUEUE	QM2.LQA	MSGUSR	UPDATE
MQQUEUE	QM2.DLQ	MOVER2	UPDATE
MQQUEUE	QM2.DLQ	MSGUSR	UPDATE

Notes:

1. The user ID passed in the MQMD of the message is used as the user ID for the **MQPUT1** on queue manager QM2 because the receiver channel was defined with PUTAUT=CTX and MCAUSER=MCATCP.
2. The MCAUSER field of the receiver channel definition is set to MCATCP; this user ID is used in addition to the channel initiator address space user ID for the checks carried out against the alternate user ID and context profile.
3. The MOVER2 user ID and the *UserIdentifier* in the message descriptor (MQMD) are used for the resource checks against the queue.
4. The MOVER2 and MSGUSR user IDs both need access to the dead-letter queue so that messages that cannot be put to the destination queue can be sent there.
5. Two user IDs are checked on all three checks performed because RESLEVEL is set to NONE.

The following profiles are required on queue manager QM2 for messages put to queue RQB on queue manager QM1 (for the LU 6.2 channel):

<i>Table 61 (Page 1 of 2). Sample security profiles for queue manager QM2 using LU 6.2</i>			
Class	Profile	User ID	Access
MQADMIN	QM2.ALTERNATE.USER.MSGUSR	MCALU62	UPDATE
MQADMIN	QM2.ALTERNATE.USER.MSGUSR	MOVER1	UPDATE
MQADMIN	QM2.CONTEXT	MCALU62	CONTROL
MQADMIN	QM2.CONTEXT	MOVER1	CONTROL
MQQUEUE	QM2.LQB	MOVER1	UPDATE
MQQUEUE	QM2.LQB	MSGUSR	UPDATE
MQQUEUE	QM2.DLQ	MOVER1	UPDATE

<i>Table 61 (Page 2 of 2). Sample security profiles for queue manager QM2 using LU 6.2</i>			
Class	Profile	User ID	Access
MQQUEUE	QM2.DLQ	MSGUSR	UPDATE
<p>Notes:</p> <ol style="list-style-type: none"> 1. The user ID passed in the MQMD of the message is used as the user ID for the MQPUT1 on queue manager QM2 because the receiver channel was defined with PUTAUT=CTX and MCAUSER=MCALU62. 2. The MCA user ID is set to the value of the MCAUSER field of the receiver channel definition (MCALU62). 3. Because LU 6.2 supports security on the communications system for the channel, the user ID received from the network is used as the channel user ID (MOVER1). 4. Two user IDs are checked on all three checks performed because RESLEVEL is set to NONE. 5. MCALU62 and MOVER1 are used for the checks performed against the alternate user ID and Context profiles, and MOVER1 and MSGUSR are used for the checks against the queue profile. 6. The MOVER1 and MSGUSR user IDs both need access to the dead-letter queue so that messages that cannot be put to the destination queue can be sent there. 			

Security profiles required for a CICS application

The CICS application uses a CICS address space user ID of CICSAD1 and a CICS task user ID of CICSTX1. The security profiles required on queue manager QM1 are different to those required for the batch application. The profiles required on queue manager QM2 are the same as for the batch application.

The following profiles are required on queue manager QM1:

<i>Table 62. Sample security profiles for the CICS application on queue manager QM1</i>			
Class	Profile	User ID	Access
MQCONN	QM1.CICS	CICSAD1	READ
MQADMIN	QM1.CONTEXT	CICSAD1	CONTROL
MQADMIN	QM1.CONTEXT	CICSTX1	CONTROL
MQQUEUE	QM1.LQ1	CICSAD1	UPDATE
MQQUEUE	QM1.LQ1	CICSTX1	UPDATE
MQQUEUE	QM1.RQA	CICSAD1	UPDATE
MQQUEUE	QM1.RQA	CICSTX1	UPDATE
MQQUEUE	QM1.RQB	CICSAD1	UPDATE
MQQUEUE	QM1.RQB	CICSTX1	UPDATE

Security problem determination

This section describes the conditions under which violation messages can be generated in an MQSeries application program and provides a checklist to be implemented if the ESM is not controlling access in the way that you expect.

Violation messages

An MQRC_NOT_AUTHORIZED can be returned to an application program because:

- A user is not allowed to connect to the queue manager. In this case, you get an ICH408I message in the Batch/TSO, CICS, or IMS joblog.
- A user signon to the queue manager has failed because, for example, the job user ID is not valid or appropriate, or the task user ID or alternate user ID is not valid. One or more of these user IDs might not be valid because they have been revoked or deleted. In this case, you get an ICHxxxx message and possibly an IRRxxxx message in the queue manager joblog giving the reason for the signon failure. For example:

```
ICH408I USER(NOTDFND ) GROUP(          ) NAME(???          )
LOGON/JOB INITIATION - USER AT TERMINAL          NOT RACF-DEFINED
IRR012I VERIFICATION FAILED. USER PROFILE NOT FOUND
```

- An alternate user has been requested, but the job or task user ID does not have access to the alternate user ID. For this failure, you get a violation message in the joblog of the relevant queue manager.
- A context option has been used or is implied by opening a transmission queue for output, but the job user ID or, where applicable, the task or alternate user ID does not have access to the context option. In this case, a violation message is put in the joblog of the relevant queue manager.
- An unauthorized user has attempted to access a secured queue manager object, for example, a queue. In this case, an ICH408I message for the violation is put in the joblog of the relevant queue manager. This violation might be due to the job or, when applicable, the task or alternate user ID.

Violation messages for command security and command resource security can also be found in the joblog of the queue manager.

If the ICH408I violation message shows the queue manager jobname rather than a user ID, this is normally the result of a blank alternate user ID being specified. For example:

```
ICH408I JOB(MQS1MSTR) STEP(MQS1MSTR)
MQS1.PAYROLL.REQUEST CL(MQQUEUE)
INSUFFICIENT ACCESS AUTHORITY
ACCESS INTENT(UPDATE ) ACCESS ALLOWED(NONE )
```

You can find out who is allowed to use blank alternate user IDs by checking the access list of the MQADMIN profile ssid.ALTERNATE.USER.-BLANK-.

An ICH408I violation message can also be generated by:

- A command being sent to the system-command input queue without context. User-written programs that write to the system-command input queue should always use a context option. For more information, see “Profiles for context security” on page 417.
- When the job accessing the MQSeries resource does not have a user ID associated with it, or when an MQSeries adapter cannot extract the user ID from the adapter environment.

What to do if access is allowed or disallowed incorrectly

In addition to the steps detailed in the *Security Server (RACF) Security Administrator's Guide*, use this checklist if access to a resource appears incorrectly controlled:

- Are the switch profiles correctly set?
 - Is RACF active?
 - Are the MQSeries RACF classes installed and active?
Use the RACF command, SETROPTS LIST, to check this.
 - Use the MQSeries command, DISPLAY SECURITY, to display the current switch status from the queue manager.
 - Check the switch profiles in the MQADMIN class.
Use the RACF commands, SEARCH and RLIST, for this.
 - Re-check the RACF switch profiles by issuing the MQSeries command, REFRESH SECURITY(MQADMIN).
- Has the RACF resource profile changed? For example, has universal access on the profile changed or has the access list of the profile changed?
 - Is the profile generic?
If it is, issue the RACF command, SETROPTS GENERIC(classname) REFRESH.
 - Have you refreshed the security on this queue manager?
If required, issue the MQSeries command, REFRESH SECURITY(*).
- Has the RACF definition of the user changed? For example, has the user been connected to a new group or has the user access authority been revoked?
 - Have you re-verified the user by issuing the MQSeries command, RVERIFY SECURITY(userid)?
- Are security checks being bypassed due to RESLEVEL?
 - Check the connecting user ID's access to the RESLEVEL profile. Use the RACF audit records to determine what the RESLEVEL is set to.
 - If you are running from CICS, check the transaction's RESSEC setting.
 - If RESLEVEL has been changed while a user is connected, they must disconnect and re-connect before the new RESLEVEL setting takes effect.

Security problem determination

Part 9. Appendixes

Appendix A. Macros intended for customer use

The macros identified in this appendix are provided as programming interfaces for customers by MQSeries.

Note: Do not use as programming interfaces any MQSeries macros other than those identified in this appendix.

General-use programming interface macros

The following macros are provided to enable you to write programs that use the services of MQSeries. The macros are supplied in library thlqual.SCSQMACS.

CMQA
CMQCDA
CMQCFA
CMQCFHA
CMQCFILA
CMQCFINA
CMQCFSLA
CMQCFSTA
CMQCIHA
CMQCXPA
CMQDLHA
CMQDXPA
CMQGMOA
CMQIIHA
CMQMDA
CMQMDEA
CMQODA
CMQPMOA
CMQRMHA
CMQTMA
CMQTMCA
CMQWCRA
CMQWDRA
CMQWIHA
CMQWPRA
CMQWXP
CMQXA
CMQXCALA
CMQXCFBA
CMQXCFCA
CMQXCDFCA
CMQXCINA
CMQXCVCA
CMQXPA
CMQXQHA
CMQXWDA

General-use programming interface copy files

The following COBOL copy files are provided to enable you to write programs that use the services of MQSeries. The copy files are supplied in library thlqual.SCSQCOBC.

CMQCDL
CMQCDV
CMQCFHL
CMQCFHV
CMQCFILL
CMQCFILV
CMQCFINL
CMQCFINV
CMQCFSSL
CMQCFSLV
CMQCFSTL
CMQCFSTV
CMQCFV
CMQCIHL
CMQCIHV
CMQCXPL
CMQCXPV
CMQDLHL
CMQDLHV
CMQGMOL
CMQGMOV
CMQIIHL
CMQIIHV
CMQMDEL
CMQMDEV
CMQMDL
CMQMDV
CMQODL
CMQODV
CMQPMOL
CMQPMOV
CMQRMHL
CMQRMHV
CMQTML
CMQTMV
CMQTM2L
CMQTM2V
CMQWIHL
CMQWIHV
CMQV
CMQXV
CMQXQHL
CMQXQHV

Product-sensitive programming interface macros

The following macros are provided to enable you to write programs that use the services of MQSeries. The macros are supplied in library thlqual.SCSQMACS.

CSQBDEF
CSQQDEFX
CSQQLITX

General-use programming interface include files

The following C include files are provided to enable you to write programs that use the services of MQSeries. The files are supplied in library thlqual.SCSQC370.

CMQC
CMQXC
CMQCFC

The following PL/I include files are provided to enable you to write programs that use the services of MQSeries. The files are supplied in library thlqual.SCSQPLIC.

CMQP
CMQEPP
CMQXP
CMQCFP

Appendix B. Using OTMA exits in IMS

If you want to send output from an IMS transaction to MQSeries, and that transaction did not originate in MQSeries, you need to code one or more IMS OTMA exits.

Similarly if you want to send output to a non-OTMA destination, and the transaction did originate in MQSeries, you also need to code one or more IMS OTMA exits.

The following exits are available in IMS to enable you to customize processing between IMS and MQSeries:

- An OTMA pre-routing exit
- A destination resolution user (DRU) exit

Exit names

You must name the pre-routing exit DFSYPRX0. You can name the DRU exit anything, as long as it does not conflict with a module name already in IMS.

Specifying the destination resolution user exit name

You can use the *Druexit* parameter of the OTMACON keyword of the CSQ6SYSP macro to specify the name of the OTMA DRU exit to be run by IMS.

We suggest you adopt a naming convention of DRU0xxxx, where xxxx is the name of your MQSeries system.

If you do not specify the name of a DRU exit in the OTMACON parameter, the default is DFSYDRU0. A sample of this module is supplied by IMS. See the *IMS/ESA Customization Guide* for information about this.

Naming convention for IMS destination

You need a naming convention for the destination to which you send the output from your IMS program. This is the destination that is set in the CHNG call of your IMS application, or that is pre-set in the IMS PSB.

A sample scenario

We suggest the OTMA destination name is synonymous with the MQSeries system name, for example the MQSeries system name repeated. (In this case, if the MQSeries system name is VCPE, the destination set by the CHNG call is VCPEVCPE.)

The pre-routing exit DFSYPRX0

You must first code a pre-routing exit DFSYPRX0. Parameters passed to this routine by IMS are documented in *IMS/ESA Customization Guide*.

This exit tests whether the message is intended for a known OTMA destination (in our example VCPEVCPE). If it is, the exit must check whether the transaction sending the message originated in OTMA. If so, it will already have an OTMA header, so you should exit from DFSYPRX0 with register 15 set to 0.

- If the transaction sending the message did not originate in OTMA, you must set the client name to be a valid OTMA client. This is the XCF member-name of the MQSeries system to which you want to send the message. The *IMS/ESA Customization Guide* tells you where to set this. We suggest you set your client name (in the OTMACON parameter of the CSQ6SYSP macro) to be the queue manager name. This is the default. You should then exit from DFSYPRX0 setting register 15 to 4.
- If the transaction sending the message originated in OTMA, and the destination is non-OTMA, you should set register 15 to 8 and exit.
- In all other cases, you should set register 15 to 0.

If you set the OTMA client name to one that is not known to IMS, your application CHNG or ISRT call returns an A1 status code.

For an IMS system communicating with more than one MQSeries system, you should repeat the logic above for each MQSeries system.

Sample assembler code to achieve the above is shown in Figure 105:

```

                TITLE 'DFSYPRX0: OTMA PRE-ROUTING USER EXIT'
DFSYPRX0 CSECT
DFSYPRX0 AMODE 31
DFSYPRX0 RMODE ANY
*
                SAVE (14,12),,DFSYPRX0&SYSDATE&SYSTEMTIME
                SPACE 2
                LR   R12,R15                MODULE ADDRESSABILITY
                USING DFSYPRX0,R12
*
                L    R2,12(,R1)            R2 -> OTMA PREROUTE PARMS
*
                LA   R3,48(,R2)            R3 AT ORIGINAL OTMA CLIENT (IF ANY)
                CLC  0(16,R3),=XL16'00'    OTMA ORIG?
                BNE  OTMAIN                 YES, GO TO THAT CODE
*
NOOTMAIN DS 0H                                NOT OTMA INPUT
                LA   R5,8(,R2)            R5 IS AT THE DESTINATION NAME
                CLC  0(8,R5),=C'VCPEVCPE'  IS IT THE OTMA UNSOLICITED DEST?
                BNE  EXIT0                 NO, NORMAL PROCESSING
*
                L    R4,80(,R2)           R4 AT ADDR OF OTMA CLIENT
                MVC  0(16,R4),=CL16'VCPE'  CLIENT OVERRIDE
                B    EXIT4                 AND EXIT
*
OTMAIN DS 0H                                OTMA INPUT
                LA   R5,8(,R2)            R5 IS AT THE DESTINATION NAME
                CLC  0(8,R5),=C'VCPEVCPE'  IS IT THE OTMA UNSOLICITED DEST?
                BNE  EXIT8                 NO, NORMAL PROCESSING

```

Figure 105 (Part 1 of 2). OTMA pre-routing exit assembler sample

```

*
EXIT0   DS 0H
        LA  R15,0           RC = 0
        B   BYEBYE

*
EXIT4   DS 0H
        LA  R15,4           RC = 4
        B   BYEBYE

*
EXIT8   DS 0H
        LA  R15,8           RC = 8
        B   BYEBYE

*
BYEBYE  DS 0H
        RETURN (14,12),,RC=(15)  RETURN WITH RETURN CODE IN R15
        SPACE 2
        REQUATE
        SPACE 2
        END

```

Figure 105 (Part 2 of 2). OTMA pre-routing exit assembler sample

The destination resolution user exit

If you have set register 15 to 4 in DFSYPRX0, or if the source of the transaction was OTMA **and** you set Register 15 to 0, your DRU exit is invoked. In our example, the DRU exit name is DRU0VCPE.

The DRU exit checks if the destination is VCPEVCPE. If it is, it sets the OTMA user data (in the OTMA prefix) as follows:

Offset (decimal)	OTMA user data
0	OTMA user data length (in this example, 334)
2	MQMD
326	Reply to format

These offsets are where the MQSeries-IMS bridge expects to find this information.

We suggest that the DRU exit is as simple as possible. Therefore, in this sample, all messages originating in IMS for a particular MQSeries system will be put to the same MQSeries queue.

If the message needs to be persistent, IMS must use a synchronized transaction pipe. To do this, the DRU exit must set the OUTPUT flag. For further details, please refer to the *IMS/ESA Customization Guide*.

You should write an MQSeries application to process this queue, and use information from the MQMD structure, the MQIIH structure (if present), or the user data, to route each message to its destination.

A sample assembler DRU exit is shown in Figure 106 on page 476.

```

                TITLE 'DRU0VCPE: OTMA DESTINATION RESOLUTION USER EXIT'
DRU0VCPE CSECT
DRU0VCPE AMODE 31
DRU0VCPE RMODE ANY
*
                SAVE (14,12),,DRU0VCPE&SYSDATE&SYSTIME
                SPACE 2
                LR R12,R15                                MODULE ADDRESSABILITY
                USING DRU0VCPE,R12
*
                L R2,12(,R1)                                R2 -> OTMA DRU PARMS
*
                L R5,88(,R2)                                R5 ADDR OF OTMA USERDATA
                LA R6,2(,R5)                                R6 ADDR OF MQMD
                USING MQMD,R6                                AS A BASE
*
                LA R4,MQMD_LENGTH+10                       SET THE OTMA USERDATA LEN
                STH R4,0(,R5)                               = LL + MQMD + 8
*
                MVI 0(R6),X'00'                            ...NULL FIRST BYTE
                MVC 1(255,R6),0(R6)                       ...AND PROPAGATE IT
                MVC 256(MQMD_LENGTH-256+8,R6),255(R6)    ...AND PROPAGATE IT
*
VCPE DS 0H
                CLC 44(16,R2),=CL16'VCPE'                IS DESTINATION VCPE?
                BNE EXIT4                                NO, THEN DEST IS NON-OTMA
                MVC MQMD_REPLYTOQ,=CL48'IMS.BRIDGE.UNSOLICITED.QUEUE'
                MVC MQMD_REPLYTOQMGR,=CL48'VCPE'         SET QNAME AND QMGRNAME
                MVC MQMD_FORMAT,MQFMT_IMS                SET MQMD FORMAT NAME
                MVC MQMD_LENGTH(8,R6),MQFMT_IMS_VAR_STRING
*
*
                B EXIT0                                    SET REPLYTO FORMAT NAME
*
EXIT0 DS 0H
                LA R15,0                                    SET RC TO OTMA PROCESS
                B BYEBYE                                    AND EXIT
*
EXIT4 DS 0H
                LA R15,4                                    SET RC TO NON-OTMA
                B BYEBYE                                    AND EXIT
*
BYEBYE DS 0H
                RETURN (14,12),,RC=(15)                 RETURN CODE IN R15
                SPACE 2
                REQUATE
                SPACE 2
                CMQA EQUONLY=NO
                CMQMDA DSECT=YES
                SPACE 2
                END

```

Figure 106. Sample assembler DRU exit

Appendix C. Upgrading and applying service to TCP/IP, Language Environment, or OS/390 Callable Services

The following tables show you what you need to do to MQSeries for OS/390 if you upgrade your level of, or apply service to, the following products:

- TCP/IP
- Language Environment®
- OS/390 Callable Services (APPC and RRS for example)

Table 63. Service has been applied or the product has been upgraded to a new release

Product	Action if using CALLLIBS	Action if using LINK
TCP/IP	You need to do this only if the TCP/IP module DSPREFIX in the SEZACMTX library has been changed. 1. Run REPORT CALLLIBS for DDDEF SEZACMTX. 2. Run the job generated by REPORT CALLLIBS.	No action required provided that the SMP/E zones were set up for automatic relinking, and the CSQ8LDQM job has been run.
Language Environment	1. Run REPORT CALLLIBS for DDDEFs SCEELKED and SCEESPC. 2. Run the job generated by REPORT CALLLIBS.	No action required provided that the SMP/E zones were set up for automatic relinking, and the CSQ8LDQM job has been run.
Callable Services	1. Run REPORT CALLLIBS for DDDEF CSSLIB. 2. Run the job generated by REPORT CALLLIBS.	No action required provided that the SMP/E zones were set up for automatic relinking, and the CSQ8LDQM job has been run.

Table 64 (Page 1 of 2). One of the products has been updated to a new release in a new SMP/E environment and libraries

Product	Action if using CALLLIBS	Action if using LINK
TCP/IP	1. Change the DDDEF for SEZACMTX to point to the new library. 2. Run REPORT CALLLIBS for DDDEF SEZACMTX. 3. Run the job generated by REPORT CALLLIBS.	1. Delete the XZMOD subentries for the following LMOD entries in the MQSeries for OS/390 target zone: <ul style="list-style-type: none"> • CSQXRCTL • CSQXSUPR • CSQXTCP 2. Set up the appropriate ZONEINDEXs between the MQSeries zones and the TCP/IP zone. 3. Tailor CSQ8LDQM to refer to the new zone on the FROMZONE parameter of the LINK commands (CSQ8LDQM can be found in the SCSQINST library). 4. Run CSQ8LDQM.

Service and upgrade considerations

Table 64 (Page 2 of 2). One of the products has been updated to a new release in a new SMP/E environment and libraries

Product	Action if using CALLLIBS	Action if using LINK
Language Environment	<ol style="list-style-type: none"> 1. Change the DDDEFs for SCEELKED and SCEESPC to point to the new library. 2. Run REPORT CALLLIBS for DDDEF SEZACMTX. 3. Run the job generated by REPORT CALLLIBS. 	<ol style="list-style-type: none"> 1. Delete the XZMOD subentries for the following LMOD entries in the MQSeries for OS/390 target zone: CMQXDCST, CMQXRCTL, CMQXSUPR, CSQCBE00, CSQCBP00, CSQCBP10, CSQCBR00, CSQUCVX, CSQVXPCB, CSQVXSPT, CSQXDCST, CSQXRCTL, CSQXSUPR, CSQXTCP, CSQXTNSV, IMQB23IC, IMQB23IM, IMQB23IR, IMQS23IC, IMQS23IM, IMQS23IR 2. Set up the appropriate ZONEINDEXs between the MQSeries zones and the Language Environment zones. 3. Tailor CSQ8LDQM to refer to the new zone on the FROMZONE parameter of the LINK commands. CSQ8LDQM can be found in the SCSQINST library. 4. Run CSQ8LDQM.
Callable services	<ol style="list-style-type: none"> 1. Change the DDDEF for CSSLIB to point to the new library. 2. Run REPORT CALLLIBS for DDDEF CSSLIB. 3. Run the job generated by REPORT CALLLIBS. 	<ol style="list-style-type: none"> 1. Delete the XZMOD subentries for the following LMOD entries in the MQSeries for OS/390 target zone: CMQXRCTL, CMQXSUPR, CSQBSRV, CSQILPLM, CSQXJST, CSQXSUPR, CSQ3AMGP, CSQ3EPX, CSQ3REPL, 2. Set up the appropriate ZONEINDEXs between the MQSeries zones and the Callable Services zones. 3. Tailor CSQ8LDQM to refer to the new zone on the FROMZONE parameter of the LINK commands. CSQ8LDQM can be found in the SCSQINST library. 4. Run CSQ8LDQM.

Appendix D. Enabling distributed queuing using CICS ISC

To enable distributed queuing using CICS ISC (the “CICS mover”), you must do the tasks described in the following sections:

- “Defining MQSeries programs and data sets as CICS resources”
- “Defining the channel definitions” on page 480
- “Defining the CKMQ transient data queue” on page 481
- “Defining MQSeries queues, triggers, and processes” on page 481
- “Defining CICS resources used by distributed queuing” on page 481
- “Security considerations for distributed queuing (using CICS ISC)” on page 482

Note: You must also define access security as described in “Security considerations for distributed queuing (using CICS ISC)” on page 482.

Prerequisites are that you have installed the CICS mover feature, and that the CICS adapter component has already been set up (see “Setting up the CICS adapter” on page 118).

Defining MQSeries programs and data sets as CICS resources

As part of installing the CICS adapter, you might already have updated the CICS system definition (CSD) data set. If you have already done this, go to “Defining the channel definitions” on page 480.

The thlqual.SCSQPROC library includes a member called CSQ4D100. This member contains the resource definition online (RDO) statements required for distributed queuing. These RDO statements must be included in the CSD of both the local CICS system and the remote CICS system to be used by the distributed queuing facility.

Notes:

1. You might have to customize CSQ4D100; in particular, the definition for the channel definition data set might have to be changed to include a data set name. There is a note at the beginning of CSQ4D100 that explains this.
2. The CSQKCDF file definition must specify a variable record format, that is, RECORDFORMAT(V). You must not change this format.

The group created is called CSQKDQ1. This group can be included in a group LIST so that the definitions are available at CICS startup. A cold start of your CICS system is required. Figure 107 shows an example of JCL that can be used to do this using the CICS DFHCSDUP offline utility.

Enabling distributed queuing using CICS ISC

```
//CSDL00KC EXEC PGM=DFHCSDUP,REGION=4096K
//STEPLIB DD DSN=CICS330.SDFHLOAD,DISP=SHR
//DFHCSD DD DSN=your.cics.csd,DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(100,10))
//SYSPRINT DD SYSOUT=A
//SYSIN DD DSN=MQM.CSQ1.USER(CSQ4D100),DISP=SHR
// DD *
ADD GROUP(CSQKDQ1) LIST(yourlist)
/*
```

Figure 107. Adding the distributed queuing definitions to the CICS CSD. This JCL sample assumes that the group CSQKDQ1 does not already exist on your CICS system.

Defining the channel definitions

You must also define the CSQKCDF data set for the channel definitions to be used by the distributed queuing facility. A data set definition is required on both the local and remote CICS systems.

The member CSQ4CHDF of thlqual.SCSQPROC contains the JCL to define the CSQKCDF data set. You must modify the JCL so that the data set high level qualifier and volume attributes conform to the naming conventions at your installation.

When the data set has been defined this DD statement can be added to your CICS startup procedure:

```
//CSQKCDF DD DSN=thlqual.CDFILE,DISP=SHR
```

Figure 108. Adding a DD statement to the CICS startup procedure

Alternatively, you can modify the DSNAME field of the CSQKCDF file definition in the CSQKDQ1 group to contain the data set name. CICS then dynamically allocates the data set, removing the need to modify the CICS startup procedure.

Notes:

1. You must not change the supplied values for the RECORDSIZE and KEYS parameters ((400 400) and (20 8) respectively) of the DEFINE CLUSTER functional command in CSQ4CHDF.
2. You should have only one channel definition file for each queue manager. A single CICS system should own the channel definition file; the other CICS systems should define it as a remote file.
3. The channel definitions must be available, via function shipping if necessary, to all CICS regions running distributed queuing programs.

Defining the CKMQ transient data queue

Messages from the MQSeries distributed queue management facility are normally sent to the system console. However, these can be routed to the CKMQ extra-partition transient data queue. CSQ4DCT2 contains a sample DCT entry for CKMQ.

The sample DCT entries, CSQ4DCT1 and CSQ4DCT2, should already be incorporated with those of the existing DCT as part of installing the CICS adapter. See step 2 on page 120. If you have not done this, follow the instructions in “System definition” on page 120. Then add a DD statement for the CKMQ transient data queue to your CICS startup procedure. For example:

```
//MQMMSG DD SYSOUT=*
```

See also “EXEC CICS LINK interface messages” on page 125.

Defining MQSeries queues, triggers, and processes

You must include the required queue definitions in your MQSeries subsystem. Distributed queuing requires a queue for use with sequence numbers and logical units of work identifiers (LUWID). You must ensure that a queue is available with the name SYSTEM.CHANNEL.SEQNO.

To pass commands to a running channel program, you need to ensure that a channel command queue exists for your system with the name SYSTEM.CHANNEL.COMMAND.

The member CSQ4DISQ in the thlqual.SCSQPROC library contains the queue definitions required for distributed queuing and examples of definitions of your own that you will need. You must customize this sample before you use it, then you can include this member in the CSQINP2 DD concatenation of the MQSeries startup procedure or you can use the COMMAND function in CSQUTIL utility to issue the required DEFINE commands.

Defining CICS resources used by distributed queuing

The distributed queuing facilities on the local and remote CICS system require the definition of certain CICS resources for communication to be established. Before starting a channel, you must define these resources using the CICS RDO facility:

ISC LU 6.2 CONNECTION

This can be one of:

- An LU 6.2 single session terminal
- An LU 6.2 single session connection
- An LU 6.2 parallel session connection

SESSIONS

You must define enough sessions to accommodate all the channels that might be active at the same time.

PROFILE (optional)

Profile definitions can be created so that channels are allocated a session from a specific mode group.

Distributed queuing (using CICS ISC)

For information about the definition of these CICS resources, see these books:

- *CICS Intercommunication Guide* for defining CICS ISC links.
- *CICS System Definition Guide* for guidance on implementing ISC in a CICS system.
- *CICS Resource Definition Guide* manual for defining resources to CICS.

Defining access security

You need to give the required access to the distributed queuing transactions. See “Security considerations for distributed queuing (using CICS ISC)” for information about this.

Setting up communications

For information on this, and all other aspects of distributed queuing using CICS ISC, see the *MQSeries Intercommunication* manual.

Security considerations for distributed queuing (using CICS ISC)

This section discusses security considerations for the “CICS mover.”

When defining and starting channels for distributed queuing, the transactions used require access to certain MQSeries for OS/390 and CICS resources. The list below shows the transactions that are used for distributed queuing and the access requirements that might be needed. Security is not a mandatory requirement and these examples are only relevant where you are using resource security.

CKMC This transaction will require RACF UPDATE access to the following resources:

- The CSQKCDF VSAM file in CICS
- The SYSTEM.CHANNEL.SEQNO local queue in MQSeries for OS/390
- The SYSTEM.CHANNEL.COMMAND local queue in MQSeries for OS/390

The CKMC transaction only needs RACF UPDATE access to the above resources under certain conditions:

- For the CSQKCDF file, only when the following functions are performed:
 - CREATE a channel
 - COPY a channel
 - DELETE a channel
 - ALTER a channel
- For the SYSTEM.CHANNEL.SEQNO local queue, only when the following functions are performed:
 - RESYNC a channel
 - RESET a channel
 - RESOLVE a channel
- For the system.channel.command local queue when requesting stop for a channel.

All other functions only require RACF READ access.

CKSG This transaction will require RACF READ access to the following resources:

- The CSQKCDF VSAM file in CICS

RACF UPDATE access to the following resources:

- The SYSTEM.CHANNEL.SEQNO local queue in MQSeries for OS/390
- The SYSTEM.CHANNEL.COMMAND local queue in MQSeries for OS/390
- The dead-letter queue (see “Dead-letter queue security” on page 411 for information about how to achieve this)

and RACF ALTER access to the following resources:

- The transmission queue specified in the channel definition in MQSeries for OS/390

CKSV This transaction will require RACF READ access to the following resources:

- The CSQKCDF VSAM file in MQSeries for OS/390

RACF UPDATE access to the following resources:

- The SYSTEM.CHANNEL.SEQNO local queue in MQSeries for OS/390
- The SYSTEM.CHANNEL.COMMAND local queue in MQSeries for OS/390
- The dead-letter queue (see “Dead-letter queue security” on page 411 for information about how to achieve this)

and RACF ALTER access to the following resources:

- The transmission queue specified in the channel definition in MQSeries for OS/390

CKRQ This transaction will require RACF READ access to the following resources:

- The CSQKCDF VSAM file in CICS

and RACF UPDATE access to the following resources:

- The SYSTEM.CHANNEL.SEQNO local queue in MQSeries for OS/390
- In MQSeries for OS/390, either
 - The object name passed in the *RemoteQname* field of the MQXQH structure, or
 - The transmission queue representing the remote queue manager, if the value in the *RemoteQMgrName* field of the MQXQH structure does not match the local queue manager name.
- In MQSeries for OS/390 the SYSTEM.CHANNEL.COMMAND local queue
- The dead-letter queue (see “Dead-letter queue security” on page 411 for information about how to achieve this)

CKRC This transaction will require RACF READ access to the following resources:

- The CSQKCDF VSAM file in CICS

and RACF UPDATE access to the following resources:

- The SYSTEM.CHANNEL.SEQNO local queue in MQSeries for OS/390
- The SYSTEM.CHANNEL.COMMAND local queue
- In MQSeries for OS/390, either

Distributed queuing (using CICS ISC)

- The object name passed in the *RemoteQName* field of the MQXQH structure, or
- The transmission queue representing the remote queue manager, if the value in the *RemoteQmgrName* field of the MQXQH structure does not match the local queue manager name
- The dead-letter queue (see “Dead-letter queue security” on page 411 for information about how to achieve this)

Appendix E. Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM documentation or non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those documents or Web sites. The materials for those documents or Web sites are not part of the materials for this IBM product and use of those documents or Web sites is at your own risk.

Notices

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Programming interface information

This book is intended to help you to administer and operate MQSeries for OS/390.

This book also documents General-use Programming Interface and Associated Guidance Information and Product-sensitive Programming Interface and Associated Guidance Information provided by MQSeries for OS/390.

General-use programming interfaces allow the customer to write programs that obtain the services of MQSeries for OS/390.

General-use Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

_____ General-use programming interface _____

General-use Programming Interface and Associated Guidance Information...

_____ End of General-use programming interface _____

Product-sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of MQSeries for OS/390. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

_____ Product-sensitive programming interface _____

Product-sensitive Programming Interface and Associated Guidance Information...

_____ End of Product-sensitive programming interface _____

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX	AS/400	BookManager
C/370	CICS	CICS/ESA
CUA	DB2	DFSMS
DFSMS/MVS	IBM	IMS
IMS/ESA	Language Environment	MQ
MQSeries	MVS/ESA	MVS/DFP
NetView	OpenEdition	OS/2
OS/390	OS/400	RACF
RMF	VSE/ESA	VTAM

Lotus, Freelance, and Word Pro are trademarks of Lotus Development Corporation in the United States, or other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

Other company, product, or service names, may be the trademarks or service marks of others.

Part 10. Glossary and index

Glossary of terms and abbreviations

This glossary defines MQSeries terms and abbreviations used in this book. If you do not find the term you are looking for, see the Index or the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

This glossary includes terms and definitions from the *American National Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42 Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.

A

abend reason code. A 4-byte hexadecimal code that uniquely identifies a problem with MQSeries for OS/390. A complete list of MQSeries for OS/390 abend reason codes and their explanations is contained in the *MQSeries for OS/390 Messages and Codes* manual.

active log. See *recovery log*.

adapter. An interface between MQSeries for OS/390 and TSO, IMS, CICS, or batch address spaces. An adapter is an attachment facility that enables applications to access MQSeries services.

address space. The area of virtual storage available for a particular job.

address space identifier (ASID). A unique, system-assigned identifier for an address space.

administrator commands. MQSeries commands used to manage MQSeries objects, such as queues, processes, and namelists.

affinity. An association between objects that have some relationship or dependency upon each other.

alert. A message sent to a management services focal point in a network to identify a problem or an impending problem.

alert monitor. In MQSeries for OS/390, a component of the CICS adapter that handles unscheduled events occurring as a result of connection requests to MQSeries for OS/390.

alias queue object. An MQSeries object, the name of which is an alias for a base queue defined to the local

queue manager. When an application or a queue manager uses an alias queue, the alias name is resolved and the requested operation is performed on the associated base queue.

allied address space. See *ally*.

ally. An OS/390 address space that is connected to MQSeries for OS/390.

alternate user security. A security feature in which the authority of one user ID can be used by another user ID; for example, to open an MQSeries object.

APAR. Authorized program analysis report.

application environment. The software facilities that are accessible by an application program. On the OS/390 platform, CICS and IMS are examples of application environments.

application queue. A queue used by an application.

archive log. See *recovery log*.

ARM. Automatic Restart Management

ASID. Address space identifier.

asynchronous messaging. A method of communication between programs in which programs place messages on message queues. With asynchronous messaging, the sending program proceeds with its own processing without waiting for a reply to its message. Contrast with *synchronous messaging*.

attribute. One of a set of properties that defines the characteristics of an MQSeries object.

authorization checks. Security checks that are performed when a user tries to issue administration commands against an object, for example to open a queue or connect to a queue manager.

authorized program analysis report (APAR). A report of a problem caused by a suspected defect in a current, unaltered release of a program.

Automatic Restart Management (ARM). An OS/390 recovery function that can improve the availability of specific batch jobs or started tasks, and therefore result in faster resumption of productive work.

B

backout. An operation that reverses all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins. Contrast with *commit*.

basic mapping support (BMS). An interface between CICS and application programs that formats input and output display data and routes multiple-page output messages without regard for control characters used by various terminals.

BMS. Basic mapping support.

bootstrap data set (BSDS). A VSAM data set that contains:

- An inventory of all active and archived log data sets known to MQSeries for OS/390
- A wrap-around inventory of all recent MQSeries for OS/390 activity

The BSDS is required if the MQSeries for OS/390 subsystem has to be restarted.

browse. In message queuing, to use the MQGET call to copy a message without removing it from the queue. See also *get*.

browse cursor. In message queuing, an indicator used when browsing a queue to identify the message that is next in sequence.

BSDS. Bootstrap data set.

buffer pool. An area of main storage used for MQSeries for OS/390 queues, messages, and object definitions. See also *page set*.

C

call back. In MQSeries, a requester message channel initiates a transfer from a sender channel by first calling the sender, then closing down and awaiting a call back.

CCF. Channel control function.

CCSID. Coded character set identifier.

CDF. Channel definition file.

channel. See *message channel*.

channel control function (CCF). In MQSeries, a program to move messages from a transmission queue to a communication link, and from a communication link to a local queue, together with an operator panel interface to allow the setup and control of channels.

channel definition file (CDF). In MQSeries, a file containing communication channel definitions that associate transmission queues with communication links.

channel event. An event indicating that a channel instance has become available or unavailable. Channel events are generated on the queue managers at both ends of the channel.

checkpoint. A time when significant information is written on the log. Contrast with *syncpoint*.

CI. Control interval.

CL. Control Language.

client. A run-time component that provides access to queuing services on a server for local user applications. The queues used by the applications reside on the server. See also *MQSeries client*.

client application. An application, running on a workstation and linked to a client, that gives the application access to queuing services on a server.

client connection channel type. The type of MQI channel definition associated with an MQSeries client. See also *server connection channel type*.

cluster. A network of queue managers that are logically associated in some way.

cluster queue. A queue that is hosted by a cluster queue manager and made available to other queue managers in the cluster.

cluster queue manager. A queue manager that is a member of a cluster. A queue manager may be a member of more than one cluster.

cluster transmission queue. A transmission queue that transmits all messages from a queue manager to any other queue manager that is in the same cluster. The queue is called SYSTEM.CLUSTER.TRANSMIT.QUEUE.

coded character set identifier (CCSID). The name of a coded set of characters and their code point assignments.

command. In MQSeries, an administration instruction that can be carried out by the queue manager.

command prefix (CPF). In MQSeries for OS/390, a character string that identifies the queue manager to which MQSeries for OS/390 commands are directed, and from which MQSeries for OS/390 operator messages are received.

command processor. The MQSeries component that processes commands.

command server. The MQSeries component that reads commands from the system-command input queue, verifies them, and passes valid commands to the command processor.

commit. An operation that applies all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins. Contrast with *backout*.

completion code. A return code indicating how an MQI call has ended.

connect. To provide a queue manager connection handle, which an application uses on subsequent MQI calls. The connection is made either by the MQCONN call, or automatically by the MQOPEN call.

connection handle. The identifier or token by which a program accesses the queue manager to which it is connected.

context. Information about the origin of a message.

context security. In MQSeries, a method of allowing security to be handled such that messages are obliged to carry details of their origins in the message descriptor.

control interval (CI). A fixed-length area of direct access storage in which VSAM stores records and creates distributed free spaces. The control interval is the unit of information that VSAM transmits to or from direct access storage.

controlled shutdown. See *quiesced shutdown*.

CPF. Command prefix.

Cross Systems Coupling Facility (XCF). Provides the OS/390 coupling services that allow authorized programs in a multisystem environment to communicate with programs on the same or different OS/390 systems.

D

datagram. The simplest message that MQSeries supports. This type of message does not require a reply.

DCI. Data conversion interface.

dead-letter queue (DLQ). A queue to which a queue manager or application sends messages that it cannot deliver to their correct destination.

default object. A definition of an object (for example, a queue) with all attributes defined. If a user defines an object but does not specify all possible attributes for that object, the queue manager uses default attributes in place of any that were not specified.

deferred connection. A pending event that is activated when a CICS subsystem tries to connect to MQSeries for OS/390 before MQSeries for OS/390 has been started.

dequeue. To remove a message from a queue. Contrast with *enqueue*.

distributed application. In message queuing, a set of application programs that can each be connected to a different queue manager, but that collectively constitute a single application.

distributed queue management (DQM). In message queuing, the setup and control of message channels to queue managers on other systems.

DLQ. Dead-letter queue.

DQM. Distributed queue management.

dual logging. A method of recording MQSeries for OS/390 activity, where each change is recorded on two data sets, so that if a restart is necessary and one data set is unreadable, the other can be used. Contrast with *single logging*.

dual mode. See *dual logging*.

dynamic queue. A local queue created when a program opens a model queue object. See also *permanent dynamic queue* and *temporary dynamic queue*.

E

enqueue. To put a message on a queue. Contrast with *dequeue*.

environment. See *application environment*.

ESM. External security manager.

event. See *channel event*, *instrumentation event*, *performance event*, and *queue manager event*.

event data. In an event message, the part of the message data that contains information about the event (such as the queue manager name, and the application that gave rise to the event). See also *event header*.

event header. In an event message, the part of the message data that identifies the event type of the reason code for the event.

event message. Contains information (such as the category of event, the name of the application that caused the event, and queue manager statistics) relating to the origin of an instrumentation event in a network of MQSeries systems.

event queue. The queue onto which the queue manager puts an event message after it detects an event. Each category of event (queue manager, performance, or channel event) has its own event queue.

external security manager (ESM). A security product that is invoked by the OS/390 System Authorization Facility. RACF is an example of an ESM.

F

FIFO. First-in-first-out.

first-in-first-out (FIFO). A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time. (A)

forced shutdown. A type of shutdown of the CICS adapter where the adapter immediately disconnects from MQSeries for OS/390, regardless of the state of any currently active tasks. Contrast with *quiesced shutdown*.

G

GCPC. Generalized command preprocessor.

generalized command preprocessor (GCPC). An MQSeries for OS/390 component that processes MQSeries commands and runs them.

Generalized Trace Facility (GTF). An OS/390 service program that records significant system events, such as supervisor calls and start I/O operations, for the purpose of problem determination.

get. In message queuing, to use the MQGET call to remove a message from a queue. See also *browse*.

global trace. An MQSeries for OS/390 trace option where the trace data comes from the entire MQSeries for OS/390 subsystem.

GTF. Generalized Trace Facility.

H

handle. See *connection handle* and *object handle*.

hardened message. A message that is written to auxiliary (disk) storage so that the message will not be lost in the event of a system failure. See also *persistent message*.

I

immediate shutdown. In MQSeries, a shutdown of a queue manager that does not wait for applications to disconnect. Current MQI calls are allowed to complete, but new MQI calls fail after an immediate shutdown has been requested. Contrast with *quiesced shutdown* and *preemptive shutdown*.

in-doubt unit of recovery. In MQSeries, the status of a unit of recovery for which a syncpoint has been requested but not yet confirmed.

initialization input data sets. Data sets used by MQSeries for OS/390 when it starts up.

initiation queue. A local queue on which the queue manager puts trigger messages.

input/output parameter. A parameter of an MQI call in which you supply information when you make the call, and in which the queue manager changes the information when the call completes or fails.

input parameter. A parameter of an MQI call in which you supply information when you make the call.

instrumentation event. A facility that can be used to monitor the operation of queue managers in a network of MQSeries systems. MQSeries provides instrumentation events for monitoring queue manager resource definitions, performance conditions, and channel conditions. Instrumentation events can be used by a user-written reporting mechanism in an administration application that displays the events to a system operator. They also allow applications acting as agents for other administration networks to monitor reports and create the appropriate alerts.

Interactive Problem Control System (IPCS). A component of OS/390 that permits online problem management, interactive problem diagnosis, online debugging for disk-resident abend dumps, problem tracking, and problem reporting.

Interactive System Productivity Facility (ISPF). An IBM licensed program that serves as a full-screen editor and dialog manager. It is used for writing application programs, and provides a means of generating standard

screen panels and interactive dialogues between the application programmer and terminal user.

IPCS. Interactive Problem Control System.

ISPF. Interactive System Productivity Facility.

L

listener. In MQSeries distributed queuing, a program that monitors for incoming network connections.

local definition. An MQSeries object belonging to a local queue manager.

local definition of a remote queue. An MQSeries object belonging to a local queue manager. This object defines the attributes of a queue that is owned by another queue manager. In addition, it is used for queue-manager aliasing and reply-to-queue aliasing.

local queue. A queue that belongs to the local queue manager. A local queue can contain a list of messages waiting to be processed. Contrast with *remote queue*.

local queue manager. The queue manager to which a program is connected and that provides message queuing services to the program. Queue managers to which a program is not connected are called *remote queue managers*, even if they are running on the same system as the program.

log. In MQSeries, a file recording the work done by queue managers while they receive, transmit, and deliver messages, to enable them to recover in the event of failure.

logical unit of work (LUW). See *unit of work*.

M

machine check interrupt. An interruption that occurs as a result of an equipment malfunction or error. A machine check interrupt can be either hardware recoverable, software recoverable, or nonrecoverable.

MCA. Message channel agent.

MCI. Message channel interface.

message. (1) In message queuing applications, a communication sent between programs. See also *persistent message* and *nonpersistent message*. (2) In system programming, information intended for the terminal operator or system administrator.

message channel. In distributed message queuing, a mechanism for moving messages from one queue

manager to another. A message channel comprises two message channel agents (a sender at one end and a receiver at the other end) and a communication link. Contrast with *MQI channel*.

message channel agent (MCA). A program that transmits prepared messages from a transmission queue to a communication link, or from a communication link to a destination queue. See also *message queue interface*.

message channel interface (MCI). The MQSeries interface to which customer- or vendor-written programs that transmit messages between an MQSeries queue manager and another messaging system must conform. A part of the MQSeries Framework.

message descriptor. Control information describing the message format and presentation that is carried as part of an MQSeries message. The format of the message descriptor is defined by the MQMD structure.

message priority. In MQSeries, an attribute of a message that can affect the order in which messages on a queue are retrieved, and whether a trigger event is generated.

message queue. Synonym for *queue*.

message queue interface (MQI). The programming interface provided by the MQSeries queue managers. This programming interface allows application programs to access message queuing services.

message queuing. A programming technique in which each program within an application communicates with the other programs by putting messages on queues.

message sequence numbering. A programming technique in which messages are given unique numbers during transmission over a communication link. This enables the receiving process to check whether all messages are received, to place them in a queue in the original order, and to discard duplicate messages.

messaging. See *synchronous messaging* and *asynchronous messaging*.

model queue object. A set of queue attributes that act as a template when a program creates a dynamic queue.

MQI. Message queue interface.

MQI channel. Connects an MQSeries client to a queue manager on a server system, and transfers only MQI calls and responses in a bidirectional manner. Contrast with *message channel*.

MQSC. MQSeries commands.

MQSeries. A family of IBM licensed programs that provides message queuing services.

N

namelist. An MQSeries object that contains a list of names, for example, queue names.

nonpersistent message. A message that does not survive a restart of the queue manager. Contrast with *persistent message*.

null character. The character that is represented by X'00'.

O

object. In MQSeries, an object is a queue manager, a queue, a process definition, a channel, a namelist, or a storage class (OS/390 only).

object descriptor. A data structure that identifies a particular MQSeries object. Included in the descriptor are the name of the object and the object type.

object handle. The identifier or token by which a program accesses the MQSeries object with which it is working.

off-loading. In MQSeries for OS/390, an automatic process whereby a queue manager's active log is transferred to its archive log.

Open Transaction Manager Access (OTMA). A transaction-based, connectionless client/server protocol. It functions as an interface for host-based communications servers accessing IMS TM applications through the OS/390 Cross Systems Coupling Facility (XCF). OTMA is implemented in an OS/390 sysplex environment. Therefore, the domain of OTMA is restricted to the domain of XCF.

OTMA. Open Transaction Manager Access.

output log-buffer. In MQSeries for OS/390, a buffer that holds recovery log records before they are written to the archive log.

output parameter. A parameter of an MQI call in which the queue manager returns information when the call completes or fails.

P

page set. A VSAM data set used when MQSeries for OS/390 moves data (for example, queues and messages) from buffers in main storage to permanent backing storage (DASD).

pending event. An unscheduled event that occurs as a result of a connect request from a CICS adapter.

percolation. In error recovery, the passing along a preestablished path of control from a recovery routine to a higher-level recovery routine.

performance event. A category of event indicating that a limit condition has occurred.

performance trace. An MQSeries trace option where the trace data is to be used for performance analysis and tuning.

permanent dynamic queue. A dynamic queue that is deleted when it is closed only if deletion is explicitly requested. Permanent dynamic queues are recovered if the queue manager fails, so they can contain persistent messages. Contrast with *temporary dynamic queue*.

persistent message. A message that survives a restart of the queue manager. Contrast with *nonpersistent message*.

ping. In distributed queuing, a diagnostic aid that uses the exchange of a test message to confirm that a message channel or a TCP/IP connection is functioning.

platform. In MQSeries, the operating system under which a queue manager is running.

point of recovery. In MQSeries for OS/390, the term used to describe a set of backup copies of MQSeries for OS/390 page sets and the corresponding log data sets required to recover these page sets. These backup copies provide a potential restart point in the event of page set loss (for example, page set I/O error).

preemptive shutdown. In MQSeries, a shutdown of a queue manager that does not wait for connected applications to disconnect, nor for current MQI calls to complete. Contrast with *immediate shutdown* and *quiesced shutdown*.

process definition object. An MQSeries object that contains the definition of an MQSeries application. For example, a queue manager uses the definition when it works with trigger messages.

Q

queue. An MQSeries object. Message queuing applications can put messages on, and get messages from, a queue. A queue is owned and maintained by a queue manager. Local queues can contain a list of messages waiting to be processed. Queues of other types cannot contain messages—they point to other queues, or can be used as models for dynamic queues.

queue manager. (1) A system program that provides queuing services to applications. It provides an application programming interface so that programs can access messages on the queues that the queue manager owns. See also *local queue manager* and *remote queue manager*. (2) An MQSeries object that defines the attributes of a particular queue manager.

queue manager event. An event that indicates:

- An error condition has occurred in relation to the resources used by a queue manager. For example, a queue is unavailable.
- A significant change has occurred in the queue manager. For example, a queue manager has stopped or started.

queuing. See *message queuing*.

quiesced shutdown. (1) In MQSeries, a shutdown of a queue manager that allows all connected applications to disconnect. Contrast with *immediate shutdown* and *preemptive shutdown*. (2) A type of shutdown of the CICS adapter where the adapter disconnects from MQSeries, but only after all the currently active tasks have been completed. Contrast with *forced shutdown*.

quiescing. In MQSeries, the state of a queue manager prior to it being stopped. In this state, programs are allowed to finish processing, but no new programs are allowed to start.

R

RBA. Relative byte address.

reason code. A return code that describes the reason for the failure or partial success of an MQI call.

receiver channel. In message queuing, a channel that responds to a sender channel, takes messages from a communication link, and puts them on a local queue.

recovery log. In MQSeries for OS/390, data sets containing information needed to recover messages, queues, and the MQSeries subsystem. MQSeries for OS/390 writes each record to a data set called the *active log*. When the active log is full, its contents are off-loaded to a DASD or tape data set called the *archive log*. Synonymous with *log*.

relative byte address (RBA). The displacement in bytes of a stored record or control interval from the beginning of the storage space allocated to the data set to which it belongs.

remote queue. A queue belonging to a remote queue manager. Programs can put messages on remote queues, but they cannot get messages from remote queues. Contrast with *local queue*.

remote queue manager. To a program, a queue manager that is not the one to which the program is connected.

remote queue object. See *local definition of a remote queue*.

remote queuing. In message queuing, the provision of services to enable applications to put messages on queues belonging to other queue managers.

reply message. A type of message used for replies to request messages. Contrast with *request message* and *report message*.

reply-to queue. The name of a queue to which the program that issued an MQPUT call wants a reply message or report message sent.

report message. A type of message that gives information about another message. A report message can indicate that a message has been delivered, has arrived at its destination, has expired, or could not be processed for some reason. Contrast with *reply message* and *request message*.

repository. A collection of information about the queue managers that are members of a cluster. This information includes queue manager names, their locations, their channels, what queues they host, and so on.

requester channel. In message queuing, a channel that may be started remotely by a sender channel. The requester channel accepts messages from the sender channel over a communication link and puts the messages on the local queue designated in the message. See also *server channel*.

request message. A type of message used to request a reply from another program. Contrast with *reply message* and *report message*.

RESLEVEL. In MQSeries for OS/390, an option that controls the number of CICS user IDs checked for API-resource security in MQSeries for OS/390.

resolution path. The set of queues that are opened when an application specifies an alias or a remote queue on input to an MQOPEN call.

resource • storage class

resource. Any facility of the computing system or operating system required by a job or task. In MQSeries for OS/390, examples of resources are buffer pools, page sets, log data sets, queues, and messages.

resource manager. An application, program, or transaction that manages and controls access to shared resources such as memory buffers and data sets. MQSeries, CICS, and IMS are resource managers.

Resource Recovery Services (RRS). An OS/390 facility that provides 2-phase syncpoint support across participating resource managers.

responder. In distributed queuing, a program that replies to network connection requests from another system.

resynch. In MQSeries, an option to direct a channel to start up and resolve any in-doubt status messages, but without restarting message transfer.

return codes. The collective name for completion codes and reason codes.

rollback. Synonym for *back out*.

RRS. Resource Recovery Services.

S

SAF. System Authorization Facility.

security enabling interface (SEI). The MQSeries interface to which customer- or vendor-written programs that check authorization, supply a user identifier, or perform authentication must conform. A part of the MQSeries Framework.

SEI. Security enabling interface.

sender channel. In message queuing, a channel that initiates transfers, removes messages from a transmission queue, and moves them over a communication link to a receiver or requester channel.

sequential delivery. In MQSeries, a method of transmitting messages with a sequence number so that the receiving channel can reestablish the message sequence when storing the messages. This is required where messages must be delivered only once, and in the correct order.

sequential number wrap value. In MQSeries, a method of ensuring that both ends of a communication link reset their current message sequence numbers at the same time. Transmitting messages with a sequence number ensures that the receiving channel can reestablish the message sequence when storing the messages.

server. (1) In MQSeries, a queue manager that provides queue services to client applications running on a remote workstation. (2) The program that responds to requests for information in the particular two-program, information-flow model of client/server. See also *client*.

server channel. In message queuing, a channel that responds to a requester channel, removes messages from a transmission queue, and moves them over a communication link to the requester channel.

server connection channel type. The type of MQI channel definition associated with the server that runs a queue manager. See also *client connection channel type*.

service interval. A time interval, against which the elapsed time between a put or a get and a subsequent get is compared by the queue manager in deciding whether the conditions for a service interval event have been met. The service interval for a queue is specified by a queue attribute.

service interval event. An event related to the service interval.

session ID. In MQSeries for OS/390, the CICS-unique identifier that defines the communication link to be used by a message channel agent when moving messages from a transmission queue to a link.

shutdown. See *immediate shutdown*, *preemptive shutdown*, and *quiesced shutdown*.

signaling. In MQSeries for OS/390 and MQSeries for Windows 2.1, a feature that allows the operating system to notify a program when an expected message arrives on a queue.

single logging. A method of recording MQSeries for OS/390 activity where each change is recorded on one data set only. Contrast with *dual logging*.

single-phase backout. A method in which an action in progress must not be allowed to finish, and all changes that are part of that action must be undone.

single-phase commit. A method in which a program can commit updates to a queue without coordinating those updates with updates the program has made to resources controlled by another resource manager. Contrast with *two-phase commit*.

SIT. System initialization table.

storage class. In MQSeries for OS/390, a storage class defines the page set that is to hold the messages for a particular queue. The storage class is specified when the queue is defined.

store and forward. The temporary storing of packets, messages, or frames in a data network before they are retransmitted toward their destination.

subsystem. In OS/390, a group of modules that provides function that is dependent on OS/390. For example, MQSeries for OS/390 is an OS/390 subsystem.

supervisor call (SVC). An OS/390 instruction that interrupts a running program and passes control to the supervisor so that it can perform the specific service indicated by the instruction.

SVC. Supervisor call.

switch profile. In MQSeries for OS/390, a RACF profile used when MQSeries starts up or when a refresh security command is issued. Each switch profile that MQSeries detects turns off checking for the specified resource.

synchronous messaging. A method of communication between programs in which programs place messages on message queues. With synchronous messaging, the sending program waits for a reply to its message before resuming its own processing. Contrast with *asynchronous messaging*.

syncpoint. An intermediate or end point during processing of a transaction at which the transaction's protected resources are consistent. At a syncpoint, changes to the resources can safely be committed, or they can be backed out to the previous syncpoint.

sysplex. A multiple OS/390-system environment that allows multiple-console support (MCS) consoles to receive console messages and send operator commands across systems.

System Authorization Facility (SAF). An OS/390 facility through which MQSeries for OS/390 communicates with an external security manager such as RACF.

system.command.input queue. A local queue on which application programs can put MQSeries commands. The commands are retrieved from the queue by the command server, which validates them and passes them to the command processor to be run.

system control commands. Commands used to manipulate platform-specific entities such as buffer pools, storage classes, and page sets.

system initialization table (SIT). A table containing parameters used by CICS on start up.

T

target library high-level qualifier (thlqual). High-level qualifier for OS/390 target data set names.

task control block (TCB). An OS/390 control block used to communicate information about tasks within an address space that are connected to an OS/390 subsystem such as MQSeries for OS/390 or CICS.

task switching. The overlapping of I/O operations and processing between several tasks. In MQSeries for OS/390, the task switcher optimizes performance by allowing some MQI calls to be executed under subtasks rather than under the main CICS TCB.

TCB. Task control block.

temporary dynamic queue. A dynamic queue that is deleted when it is closed. Temporary dynamic queues are not recovered if the queue manager fails, so they can contain nonpersistent messages only. Contrast with *permanent dynamic queue*.

termination notification. A pending event that is activated when a CICS subsystem successfully connects to MQSeries for OS/390.

thlqual. Target library high-level qualifier.

thread. In MQSeries, the lowest level of parallel execution available on an operating system platform.

time-independent messaging. See *asynchronous messaging*.

TMI. Trigger monitor interface.

trace. In MQSeries, a facility for recording MQSeries activity. The destinations for trace entries can include GTF and the system management facility (SMF). See also *global trace* and *performance trace*.

transid. See *transaction identifier*.

transaction identifier. In CICS, a name that is specified when the transaction is defined, and that is used to invoke the transaction.

transmission program. See *message channel agent*.

transmission queue. A local queue on which prepared messages destined for a remote queue manager are temporarily stored.

trigger event. An event (such as a message arriving on a queue) that causes a queue manager to create a trigger message on an initiation queue.

triggering. In MQSeries, a facility allowing a queue manager to start an application automatically when predetermined conditions on a queue are satisfied.

trigger message. A message containing information about the program that a trigger monitor is to start.

trigger monitor. A continuously-running application serving one or more initiation queues. When a trigger message arrives on an initiation queue, the trigger monitor retrieves the message. It uses the information in the trigger message to start a process that serves the queue on which a trigger event occurred.

trigger monitor interface (TMI). The MQSeries interface to which customer- or vendor-written trigger monitor programs must conform. A part of the MQSeries Framework.

two-phase commit. A protocol for the coordination of changes to recoverable resources when more than one resource manager is used by a single transaction. Contrast with *single-phase commit*.

U

undo/redo record. A log record used in recovery. The redo part of the record describes a change to be made to an MQSeries object. The undo part describes how to back out the change if the work is not committed.

unit of recovery. A recoverable sequence of operations within a single resource manager. Contrast with *unit of work*.

unit of work. A recoverable sequence of operations performed by an application between two points of consistency. A unit of work begins when a transaction starts or after a user-requested syncpoint. It ends either at a user-requested syncpoint or at the end of a transaction. Contrast with *unit of recovery*.

utility. In MQSeries, a supplied set of programs that provide the system operator or system administrator with facilities in addition to those provided by the MQSeries commands. Some utilities invoke more than one function.

X

XCF. Cross Systems Coupling Facility.

Index

Special Characters

&ZSEL 89
% character in RACF profiles 401

A

abend

application option of SSM entry 174
CICS transaction disconnecting 145
starting after 195
states 280
U3042 177

abnormal termination 277, 280

access

if incorrect 465
restricting by using alias queues 408

access method services (AMS)

BSDS definition 306
commands 350
defining page sets 62
deleting damaged BSDS 346
new active log definition 321
renaming damaged BSDS 346
REPRO 332, 350

accounting

eye catcher string 388
introduction 363
message manager 388
rules for data collection 368
sample SMF records 390
SMF trace 366
starting automatically 72

ACS user-exit filter, archive log data sets 311

ACTCHL parameter of CSQ6CHIP 83

active log

copy to archive 20
CSQJU003, change log inventory utility 256
data set
copying 321
copying with AMS REPRO statement 332
off-loaded to archive log 302
VSAM linear 301
defining in BSDS 321
delays in off-loading 338
deleting from BSDS 322
description 299
dual logging 303
enlarging 322
input buffer size (INBUFF) 74
log print utility (CSQ1LOGP) 266
number of buffers per write 76

active log (*continued*)

off-loading 303
out of space 338
output buffer
number filled (WRTHRSH) 76
size (OUTBUFF) 75
printing (CSQ1LOGP) 266
recording existing in BSDS 322
recovery plan, problems 338
single or dual (TWOACTV) 75
space allocation
primary (PRIQTY) 79
secondary (SECQTY) 80
units (ALCUNIT) 77
status 320
stopped data set effect 342
truncation 304
writing 303

active log problem

delays in off-loading 338
dual logging lost 339
log stopped 342
out of space 338
read I/O errors 340
write I/O errors 339

ADAPS parameter of CSQ6CHIP 84

adapter

CICS 109—155
illustration 8
introduction 8, 22

adapter subtasks, number to use for channel initiator 84

adapters

IMS 159—161, 180

adapters and dispatchers, total number 86

address space

abend 192
canceling for MQSeries 196
CICS adapter 111
CICS, user ID 432
connection security 439
for MQSeries 22
IMS adapter 159
IMS control region 165
IMS, user ID 432
user ID 429, 430

administering

by writing programs 211
introduction 27

administration programs 211, 214

administrator commands 19

Index

- age, specifying for OTMA 72
- ALCUNIT parameter of CSQ6ARVP 77
- alert monitor 112
- alias queue 10
 - system default 55
- alias queues
 - command resource checking 422
 - restricting access using 408
 - security 407, 411
 - undelivered messages 411
- ALL, attribute of DISPLAY SECURITY 443
- ALTER commands, security 423
- alter queue attributes, security 423
- ALTER SECURITY command 441
- alternate user ID 431
- alternate user ID, distributed queuing 429
- alternate user security 396, 415
- AMS (access method services)
 - BSDS definition 306
 - commands 350
 - defining page sets 62
 - deleting damaged BSDS 346
 - new active log definition 321
 - renaming damaged BSDS 346
 - REPRO 332, 350
- AMS REPRO, backing up and recovering page sets 332
- APF authorization of load libraries 47
- API-crossing exit
 - defining 114
 - enable or disable 132
- API-resource security 395
 - quick reference 419
 - RESLEVEL 425
- APPC
 - LU name 84
 - LUADD 84
 - maximum number of current channels 85
 - restart interval after failure 84
- APPC, applying service 477
- APPCPMxx 84
- application
 - data 4
 - environments 4
 - in MQSeries 14
 - program 23, 299
 - batch adapter 23
 - CICS adapter 24
 - command format 124
 - CQKC DISPLAY 142
 - IMS adapter 25
 - issuing commands from 211
 - recovery 299
 - starting automatically 14
 - time-independent 3
 - types 14
- application access control 450
- application stubs, coexistence with earlier versions 98
- applid node name 121
- archive initialization parameters, setting 68
- archive log
 - ACS user-exit filter 311
 - adding information to BSDS (NEWLOG) 258, 323
 - BSDS copies 306
 - cataloging (CATALOG) 78
 - compacting (COMPACT) 78
 - create 20
 - CSQJU003, change log inventory utility 256
 - data set
 - name prefix 77
 - off-loading 302
 - password 261
 - password protection (PROTECT) 79
 - time stamp (TSTAMP) 80
 - deleting 317
 - deleting information from the BSDS 260, 323
 - device type (UNIT) 80
 - discarding records 317
 - dual copies 306
 - dynamic allocation of data sets 310
 - information in BSDS 305
 - input buffer size (INBUFF) 74
 - maximum number in BSDS (MAXARCH) 74
 - maximum number of volumes (MAXALLC) 74
 - mounting, WTOR (ARCWTOR) 78
 - output buffer size (OUTBUFF) 75
 - password, changing 323
 - printing (CSQ1LOGP) 266
 - quiesce time (QUIESCE) 80
 - recording in BSDS 323
 - recovery plan 343
 - retention period (ARCRETN) 77
 - route codes (ARCWRTC) 77
 - single or dual (TWOARCH) 75
 - space allocation
 - block size (BLKSIZE) 78
 - primary (PRIQTY) 79
 - secondary (SECQTY) 80
 - units (ALCUNIT) 77
- ARCHIVE LOG command
 - security 423
- archive log problem
 - allocation problems 343
 - insufficient DASD for off-load 344
 - read I/O errors during restart 344
 - write I/O errors during off-load 343
- ARCHIVE LOG, command 315
- archive parameter
 - default 76
 - setting 76
- ARCHIVE, utility function (CSQJU003) 261

- archiving 299, 310, 315
 - See *also* archive log
 - controlling, OFFLOAD parameter of CSQ6LOGP 75
 - log data sets 304, 310
 - SMS 311
 - ARCPFX1 parameter of CSQ6ARVP 77
 - ARCPFX2 parameter of CSQ6ARVP 77
 - ARCRETN parameter of CSQ6ARVP 77
 - ARCWRTC parameter of CSQ6ARVP 77
 - ARCWTOR parameter of CSQ6ARVP 78
 - ARM
 - activating a policy 296
 - couple data sets 294
 - defining a policy 295
 - introduction 294
 - network considerations 296
 - policy sample 295
 - registering with 296
 - ARM (automatic restart manager)
 - coexistence 93
 - introduction 26
 - LUADD for channel initiator 84
 - migration 93
 - subsystem name table 45
 - attributes of queues 9
 - audit, security 437
 - auditing RESLEVEL 437
 - authority, alternate user 396
 - auto-reconnect, CICS adapter 113
 - Automatic Restart Manager
 - See ARM
 - automatic restart manager (ARM)
 - coexistence 93
 - introduction 26
 - LUADD for channel initiator 84
 - migration 93
 - subsystem name table 45
 - automating starting of CKTI 446
 - availability, recovery planning 307
- B**
- back out 274
 - backing up
 - logs, frequency 308
 - object definitions 309
 - page sets 331, 332
 - queues 307
 - base function 33
 - batch
 - example security scenario 458
 - Batch adapter
 - See Batch/TSO adapter
 - batch application, improving portability 87
 - batch assembler, IVP 99
 - batch message program (BMP)
 - See BMP (batch message program)
 - batch utility
 - See MQSeries utility program (CSQUTIL)
 - batch, testing customization 99
 - Batch/TSO adapter 23
 - connection security 404
 - CSQBSTUB stub program 23
 - installing 87
 - maximum number of connections
 - background (IDBACK) 70
 - foreground (IDFORE) 70
 - total (CTHREAD) 69
 - OS/390 SNAP dump 87
 - RESLEVEL 425
 - security checking 425
 - user IDs, security checking 431
 - Batch/TSO RRS adapter
 - See RRS adapter
 - bibliography xvi
 - blank fields in operations and control panels 209
 - blank user IDs 436
 - BLKSIZE parameter of CSQ6ARVP 78
 - BLSCECTX SYS1.PARMLIB member 90
 - BMP (batch message program) 174
 - BookManager xxi
 - bootstrap data set (BSDS)
 - introduction 17
 - BSDS (bootstrap data set) 299
 - changing log inventory utility (CSQJU003) 323
 - defining 305
 - dual copies 305
 - errors 346
 - introduction 17
 - managing 315, 323
 - print log map utility (CSQJU004) 265
 - recovery 350
 - registers log data 305
 - time stamps 319
 - what it is for 305
 - BSDS problem
 - does not agree with log 349
 - error while opening 347
 - I/O error 346
 - out of synchronization 348
 - unequal time stamps 347
 - buffer
 - illustration 17
 - input buffer size (INBUFF) 74
 - maximum number 17
 - number filled before write to log 76
 - output buffer size (OUTBUFF) 75
 - output log 302
 - what they are 17
 - buffer manager
 - suppressing console messages 91

Index

- buffer manager statistics 376
 - interpretation 377
 - buffer pool
 - defining 17, 20, 54
 - illustration 17
 - performance implications 17
 - buffer pools
 - management 377
 - size 379
 - statistics 378
 - building messages 216
 - bypass of syncpoint 275
- ## C
- C and C++, testing customization 102
 - callable services, applying service 477
 - canceling MQSeries address space 196
 - CARTs 190
 - CATALOG parameter of CSQ6ARVP 78
 - catalog, archive log (CATALOG) 78
 - CCSID (coded character set identifier)
 - queue manager (QMCCSID) 72
 - CCSID, keyword of COMMAND function 239
 - change log inventory utility (CSQJU003) 256—264
 - adding new active log 321, 338
 - BSDS preparation 61
 - change BSDS 319, 321
 - changes for active logs 321
 - changes for archive logs 322
 - functions
 - ARCHIVE 261, 351
 - CHECKPT 263
 - conditional restart 262
 - CRESTART 262
 - DELETE 260
 - HIGHRBA 264
 - NEWLOG 258, 321, 323
 - setting checkpoint records 263
 - updating the highest written log RBA 264
 - invoking 256
 - log data set preparation 61
 - managing log data sets 305
 - multiple statement operation 257
 - time stamp in BSDS 348
 - CHANGE SUBSYS, command of IMS 169, 173
 - channel
 - controlling 20
 - introduction 13
 - maximum number active 83
 - maximum number current 84
 - maximum number of current LU 6.2 85
 - maximum number of current TCP/IP 85
 - queues 56
 - sample definitions 58
 - storage requirement 28
 - channel (*continued*)
 - suppressing console messages 91
 - system default 55
 - user ID 429
 - channel definitions (CICS mover) 480
 - channel file definition (CSQKCDF) 479
 - channel initiator
 - controlling 21
 - CSQ6CHIP 83
 - defining the procedure 50
 - installation verification program 102
 - maximum number of connections (CTHREAD) 69
 - migration considerations 94
 - restarting with ARM 296
 - sample definitions 60
 - tailoring the parameter module 82
 - channel initiator parameter module, invoking 83
 - channel initiator parameters
 - default 83
 - displaying settings 82
 - migration considerations 93
 - channel listener, controlling
 - channel reply information queue 11
 - channel sequence number queue 481
 - channel synchronization queue 11
 - channel-initiation queue 11
 - checkpoint
 - number of log records (LOGLOAD) 71
 - records 300, 301
 - checkpoint records, setting 263
 - CHECKPT, utility function (CSQJU003) 263
 - Chinese language feature 39, 227
 - CI (control interval), description 302
 - description 310
 - CICS
 - address space user ID, security checking 430
 - CICS mover
 - See distributed queuing using CICS ISC
 - completing the connection 121
 - connection ID 283
 - definition of term xv
 - example security scenario 458
 - in-doubt units of recovery 283
 - monitoring facility 367
 - MQSeries-CICS bridge 24
 - recovery considerations when using MQSeries 312
 - related publications xxiii
 - resources for distributed queuing 479
 - system administration and operation 24
 - system definition (CSD) data set 118
 - terminating 144
 - testing customization 102
 - units of recovery 286
 - user IDs, security checking 432
 - CICS adapter 57, 109—155
 - address space 111

- CICS adapter (*continued*)
 - administrative functions 109
 - alert monitor 112
 - API-crossing exit 114
 - application programming 24
 - auto-reconnect 113
 - CKTI task initiator 113
 - commands 123
 - components 111
 - connect program (CSQCQCON) 129
 - connection security 404
 - consistency with MQSeries 275
 - control panels 126—142
 - conventions 117
 - CSQ4INYG object sample 57
 - CSQCSTUB stub program 111, 121
 - customizing 122
 - deferred connection 112
 - disconnect program 131
 - displaying CICS tasks 135
 - displaying connection details 135
 - displaying status 142
 - forced shutdown 144, 145
 - initiation queue, defining 58
 - installing 118
 - introduction 23
 - maximum number of connections (CTHREAD) 69
 - MQI support 110
 - object sample 57
 - operation of
 - control functions 109
 - control panels 126
 - displaying current tasks 141
 - displaying instances of CKTI 140
 - lowercase queue names 128
 - modifying a connection 132
 - starting a connection 127
 - starting CKTI 136
 - stopping CKTI 138
 - orderly shutdown 144
 - overview 109
 - passing parameters 124
 - pending events 112
 - performance 115
 - connection statistics 116
 - multi-tasking 114
 - PLTPI program, writing 122
 - quiesced shutdown 144
 - recovery 283
 - RESLEVEL 425
 - resource definition 118
 - restart, what happens 291
 - restrictions 113
 - security 446
 - authorization 445
 - checking 425
 - for transactions 445
- CICS adapter (*continued*)
 - security (*continued*)
 - PLTPI 446
 - PLTSD 446
 - terminal user IDs 446
 - shutting down a connection 144
 - starting a connection 127
 - statistics 364
 - system definition 120
 - task initiation program (CSQCSSQ) 137
 - terminating 144
 - testing customization 102
 - transaction authority 445
 - transaction services
 - security support 445
 - syncpoint support 275
 - two-phase commit 116, 276
 - user IDs for security 425, 430, 446
 - VTAM node name, connection ID in system
 - initialization tables 121
- CICS bridge
 - customizing 152
 - definition 147
 - migration considerations 93
 - prerequisite APARs 152
 - security 447
 - starting 154
 - stopping 155
 - system configuration 148
 - tuning considerations 154
 - when to use 147
- CICS mover
 - See distributed queuing using CICS ISC
- CICS system definition data set (CSD)
 - updating for CICS 118
- CICS Transaction Server for OS/390
 - See CICS
- CKBM, security 445
- CKCN, security 445
- CKDL, security 445
- CKDP, security 445
- CKMQ, transient data queue 481
- CKQC
 - authorization 445
 - CICS adapter transaction 109
 - DISPLAY command 142
 - MODIFY command 133
 - security 445
 - START command 128
 - STARTCKTI command 137
 - STOP command 130
 - STOPCKTI command 138
- CKQQ, transient data queue 125
- CKRS, security 445
- CKRT, security 445

Index

- CKSD, security 445
- CKSG, MCA transaction 446
- CKSQ, security 445
- CKTI transaction 113
 - automating starting of 446
 - displaying 140
 - propagating user IDs 446
 - security 445, 446
 - starting 136
 - stopping 138, 139
- class of service 26
- client
 - channels 13
 - sample definitions 58
- client attachment feature 33
- client channel definition file, generating 239
- close options, dynamic queues 410
- cluster
 - coexistence 94
 - command queue 11
 - commands 21
 - introduction 6
 - joining 20
 - leaving 20
 - migration 94
 - queues 10, 57
 - queues required 11
 - repository queue 11
 - sample definitions 59
 - transmission queue 12
- cluster support
 - security considerations 455
- CMDUSER parameter of CSQ6SYSP 69
- COBOL, testing customization 102
- coded character set identifier (CCSID)
 - queue manager (QMCCSID) 72
- coded character set identifier, queue manager (QMCCSID) 72
- cold start 356
- command
 - administrator 19
 - for the CICS adapter 123
 - introduction 18
 - issuing
 - from initialization input data sets 52
 - methods of issuing 19
 - processor 12, 213
 - queue manager 8
 - queues required 12
 - resource checking summary table 423
 - resource security 397
 - resource security profiles 422
 - security 397
 - security profiles 421
 - summary table 19
 - system control, summary 20
- command and response tokens 190
- command prefix (CPF)
 - displaying existing 43
 - naming convention 42
 - valid characters 43
- command prefix string
 - See CPF (command prefix)
- command prefix strings
 - See CPF (command prefix)
- command queue, clustering 11
- command server 12, 213—226
 - controlling 20
 - restart 213
 - sending commands to 216
 - starting 213
 - stopping 213
- COMMAND, CSQUTIL function 238
 - MAKECLNT keyword 241
 - MAKEDEF keyword 240
- commands
 - See *also* MQSeries, commands
 - DISPLAY 364
 - examples of 220
 - in request messages 216
 - issuing 189, 190
 - from CSQUTIL 191, 238
 - from system-command input queue 211
 - from the OS/390 console 190
 - from TSO panels 191
 - no reply to 226
 - operator 191
 - remote queue manager 214
 - STOP QMGR 195
 - user messages
 - from DEFINE 221
 - from DEFINE QLOCAL 223
 - from DEFINE THREAD 222
 - from DELETE 221
 - from DISPLAY 221
 - from DISPLAY commands 222
- commit 275
- commit point
 - See point of consistency
- communication protocol, choosing 40
- COMPACT parameter of CSQ6ARVP 78
- compacting archive logs (COMPACT) 78
- conditional restart 262
- connection parameters, setting 68
- connection security 395, 439, 449
- connections
 - controlling CICS 122
 - controlling IMS 169
 - displaying details of
 - CICS 135
 - IMS 174
 - maximum number
 - Batch/TSO, background (IDBACK) 70

- connections (*continued*)
 - maximum number (*continued*)
 - Batch/TSO, foreground (IDFORE) 70
 - total (CTHREAD) 69
 - monitoring the activity on 174
 - profiles for security 404
 - starting from
 - CICS adapter control panel 127
 - CICS application program 129
 - CICS command line 128
 - IMS 170
 - PLTPI program 119
 - statistics for CICS adapter 116
 - stopping from
 - CICS adapter control panel 130
 - CICS application program 131
 - CICS command line 130
 - IMS 169
 - to IMS, monitoring activity 174—175
- consistency of data, abnormal termination 277
- consistency with batch, CICS, IMS, and TSO 275
- console messages, suppressing 91
- context security 396, 417
- continuous operation, recovery planning 307
- control functions, CICS adapter 109
- control interval
 - See CI (control interval), description
- control panels for the CICS adapter 126
- control region
 - See CTL (IMS control region)
- controlling application connections 122
- controls and options, security 394
- coordinator, recovery 275
- copy files
 - general-use programming interface 470
- COPY, CSQUTIL function 246
- copying
 - messages from a queue (COPY) 245
 - page sets
 - COPYPAGE function 233
 - RESETPAGE function 235
 - queues to a data set (COPY) 246
 - queues to a data set (SCOPY) 249
- COPYPAGE, CSQUTIL function 233
- Correld field, administration programs 218
- COUNT field, user messages 220
- couple data sets, ARM 294
- CPF (command prefix)
 - displaying existing 43
 - establishing 41
 - issuing commands 190
 - naming convention 42
 - registering 44
 - registration 44
 - running in a sysplex 44
 - scope 44
- CPF (command prefix) (*continued*)
 - valid characters 42, 43
- CRESTART, utility function (CSQJU003) 262
- CSA storage requirement 28
- CSD (CICS system definition data set) 118
- CSQ1LOGP
 - See log print utility (CSQ1LOGP)
- CSQ2020E message 184
- CSQ4BSDS, sample 346, 350
- CSQ4D100, customization 479
- CSQ4DISP, display sample 54, 59
- CSQ4DISQ, distributed queuing sample 54
- CSQ4DISQ, distributed queuing using CICS ISC
 - sample 59
- CSQ4INP1 sample initialization data set 54
- CSQ4INPX sample initialization data set 54
- CSQ4INSG sample initialization data set 54
- CSQ4INSX sample initialization data set 54
- CSQ4INSX system object sample 56
- CSQ4INYC sample initialization data set 54
- CSQ4INYD sample initialization data set 54
- CSQ4INYG sample initialization data set 54
- CSQ4IVP, installation verification program 54, 59
- CSQ4IVP1
 - RACF commands 100, 103
- CSQ4IVP1 installation verification program 99
- CSQ4IVPX 102
 - example output 105
- CSQ4MPFL information message suppression
 - sample 91
- CSQ4MSTR sample startup procedure 49
- CSQ4PAGE page set sample 62
- CSQ6ARVP macro 67, 76
- CSQ6ARVP, macro 310
- CSQ6CHIP 83
- CSQ6LOGP macro 67, 74
- CSQ6SYSP macro 67, 68
- CSQ6SYSP, macro 368
- CSQBDEFV 87
- CSQBSTUB for Batch/TSO adapter 23
- CSQBSTUB for IMS adapter 25
- CSQCAPX sample API-crossing exit program 114
- CSQCCODF sample PLTPI program 119
- CSQCDSC CICS adapter disconnect program 131
- CSQCQCON CICS adapter connect program 129
- CSQCRST CICS adapter reset program 133
- CSQCSSQ CICS adapter task initiation program 137, 139
- CSQCSTUB for CICS adapter 24, 111, 121
- CSQINP1
 - commands 52
 - sample data set 54
 - security 397, 431
- CSQINP2
 - commands 52
 - security 397, 431

Index

- CSQINP2 (*continued*)
 - updating 121
 - using 55
 - CSQINPX
 - security 397, 453
 - specifying a response time 53
 - CSQINPx data sets, issuing commands from 52
 - CSQJ004I message 339
 - CSQJ030E message 342
 - CSQJ100E message 347
 - CSQJ102E message 349
 - CSQJ103E message 343
 - CSQJ105E message 339
 - CSQJ106E message 340
 - CSQJ107E message 346
 - CSQJ108E message 346
 - CSQJ110E message 338
 - CSQJ111A message 338
 - CSQJ114I message 344
 - CSQJ115E message 343
 - CSQJ120E message 347
 - CSQJ122E message 348
 - CSQJ124E message 340
 - CSQJ126E message 346
 - CSQJ138E message 344
 - CSQJU003
 - See change log inventory utility (CSQJU003)
 - CSQJU004
 - See print log map utility (CSQJU004)
 - CSQKCDF, channel file definition 479
 - CSQOREXX 89
 - CSQP004I message 353
 - CSQP018I message 192
 - CSQP019I message 192
 - CSQQDEFV, subsystem definition table 161—166
 - CSQQDEFX, macro 165
 - CSQQSTUB for IMS adapter 25
 - CSQQSTUB, for IMS adapter 159
 - CSQQTRMN program
 - starting 177
 - CSQQTRMN transaction 160
 - stopping 178
 - CSQQxxx messages 358
 - CSQSNAP 87, 110, 159
 - CSQUTIL
 - See also MQSeries utility program (CSQUTIL)
 - RESLEVEL 425
 - security checking 425
 - CSQWDMP 90
 - CSQWDPRD 90
 - CSQXPARM
 - description 82
 - displaying settings 82
 - CSQZPARM
 - creating 67
 - displaying settings 67
 - CSQZPARM (*continued*)
 - specifying an alternate 194
 - CTHREAD parameter of CSQ6SYSP 69
 - CTL (IMS control region) 170, 174
 - CURRCHL parameter of CSQ6CHIP 84
 - customizing 33
 - before you start 37
 - CICS adapter 122
 - CICS bridge 152
 - IMS bridge 181
 - initialization input data sets 52
 - introduction 27, 37
 - IPL 37
 - overview 33
 - planning 33
 - security 444
 - tasks 37
 - testing 99
 - when migrating from previous versions 93
- ## D
- DASD 17, 302, 310
 - performance 377
 - Data Facility Hierarchical Storage Manager
 - See DFHSM (Data Facility Hierarchical Storage Manager)
 - data integrity 4
 - data manager statistics 375
 - data set
 - See also page set
 - multivolume 66
 - space management 15
 - data sets
 - See also active log
 - See also archive log
 - active log 321
 - archive log 310
 - bootstrap, creating 61
 - copying messages from queues 246
 - copying messages from queues (offline) 249
 - dump and restore 333
 - initialization 52
 - log, creating 61
 - page set I/O error 353
 - restart on losing 356
 - restoring messages from 254
 - datagram, message 5
 - dead-letter queue 12, 58
 - finding out its name 224
 - security 411
 - DEAD.QUEUE
 - See dead-letter queue
 - default
 - archive parameters 76
 - CCSID 72

- default (*continued*)
 - channel definition 13
 - channel initiator parameters 83
 - CSQ6ARVP macro 76
 - CSQ6LOGP macro 74
 - CSQ6SYSP macro 68
 - logging parameters 74
 - namelist definition 13
 - process definition 14
 - queue definitions 13
 - routing code 72
 - system queues 13
 - user ID 69
- DEFAULT storage class 57
- default transmission queue 58
- deferred connection for CICS 112
- DEFINE commands, security 423
- defining
 - buffer pools 17
 - CICS resources for MQSeries 118
 - MQSeries to IMS 162
 - storage class 15
 - subsystems 41
- defining queues 9
 - defaults 13
- definitions
 - storing 15
- DELETE commands, security 423
- DELETE, utility function (CSQJU003) 260
- deleting
 - active information log from BSDS 322
 - archive logs 317, 318
 - log information from BSDS 260
 - messages from a queue 252
- dependent region, IMS 174, 175
 - disconnecting from 175
 - user ID 432
- DEQUEUE TMEMBER, command of IMS 183
- descriptor, message 4
- DEST option, trace data destination 369
- destination resolution exit
 - sample 475
 - specifying name 71
- destination resolution exit, writing 473
- device type for logs (UNIT) 80
- DFHSM (Data Facility Hierarchical Storage Manager)
 - in backup and recovery 311—312
- DFS3611 message 358
- DFS555I message 357
- DFSMS
 - related publications xxiv
- DFSYDRU0 sample module 473
- DFSYPRX0 473
- disaster recovery 312
- discarded messages 219
- disconnecting
 - from CICS 130
 - from IMS 176
- dispatchers and adapters, total number 86
- dispatchers, number to use for channel initiator 84
- display
 - channel initiator parameters 82
 - CKQC transaction 142
 - system settings 67
- DISPLAY commands, security 423
- DISPLAY OASN command of IMS 173
- display sample 59
- DISPLAY THREAD 280
- displaying
 - function key settings 200
 - units of recovery in CICS 286
 - units of recovery in IMS 172, 288
- DISPS parameter of CSQ6CHIP 84
- distributed queuing
 - choosing facility 40
 - connection security 405
 - CSQINPX 53
 - defining the data sets 50
 - definition of term xv
 - initialization input data sets 53
 - installation verification program 102
 - LE runtime library 46
 - MCA user ID 433
 - queues required 11
 - RESLEVEL 427
 - sample definitions 58
 - SCEERUN 46
 - security 51
 - security checking 427
 - security considerations 453
 - setting the CCSID 72
 - testing customization 102
- distributed queuing using CICS ISC
 - channel definitions 480
 - channel file definition 479
 - channel sequence number queue 481
 - defining CICS resources 479, 481
 - defining queues, triggers, and processes 481
 - LU 6.2 connections 481
 - sample, CSQ4DISQ 59
 - security 482
 - transient data queue 481
- DRU exit
 - specifying name 71
- DRU exit sample
 - sample 475
- DRU exit, writing 473
- druexit name, specifying for OTMA 71
- dual BSDS (TWOBSDS) 75
- dual logging 300
 - establishing 309

Index

- dual logging (*continued*)
 - losing 339
 - specifying for active log (TWOACTV) 75
 - specifying for archive log (TWOARCH) 75
- dump formatting member 90
- dynamic calls, IMS 162
- dynamic expansion of page sets 65
- dynamic queue
 - specifying a name 10
 - template 10
- dynamic queues 5
 - close options 410
 - security 409

E

- early code library 46
- early code, multiple versions 98
- ECSA storage requirement 28
- editing namelists 208
- EMCS 190
- EMPTY, utility function (CSQUTIL) 252
- English language feature 39
- ENQUEUE names, CICS 117
- environments, for applications 4
- error symptoms
 - OS/390 error recovery program message 344
- errors, hardware 359
- euro currency support 95
- event
 - introduction 18
- event queues 12
- event reporting sample, CSQ4INSG 55
- event-driven processing 4
- example
 - ARM policy 295
 - output from CSQ4IVPX 105
 - security scenario 458
- example recovery scenarios
 - active log problems
 - delays in off-loading 338
 - dual logging lost 339
 - log stopped 342
 - out of space 338
 - read I/O errors 340
 - write I/O errors 339
 - archive log problems
 - allocation problems 343
 - insufficient DASD for off-load 344
 - read I/O errors during restart 344
 - write I/O errors during off-load 343
- BSDS problems
 - BSDS recovery 350
 - does not agree with log 349
 - error while opening 347
 - I/O error 346
 - out of synchronization 348

- example recovery scenarios (*continued*)
 - BSDS problems (*continued*)
 - unequal time stamps 347
 - hardware problems 359
 - IMS problems
 - application terminates 357
 - IMS not operational 357
 - unable to connect to MQSeries 358
 - page set problems
 - I/O error 353
 - page set full 353
- EXEC CICS LINK
 - COMMAREA option 122
 - INPUTMSG option 124
 - linking to the CICS adapter 122, 129
- exit program
 - CICS adapter 114
 - number of TCBs 70
 - time allowed per invocation 70
- EXITLIM parameter of CSQ6SYSP 70
- EXITTCB parameter of CSQ6SYSP 70
- expanding page sets 233
- extended console support 190
- Extended Recovery Facility (XRF) 312
- external security manager (ESM) 393
- eye catcher strings 382, 388
 - SMF type 115 records 382
 - SMF type 116 records 388

F

- F keys
 - See function keys
- FAILURE, keyword of COMMAND function 239
- features, installable 33
- finding archive log data sets to be deleted 318
- FORCE 279
 - parameter, table of effects 279
- FORCE keyword of FORMAT 231
- FORCE option of STOP QMGR command 280
- FORCE parameter or RESETPAGE 236
- format
 - type 115 SMF records 371
 - type 116 SMF records 385
- FORMAT, utility function (CSQUTIL) 231
- formatting dumps 90
- frequency backing up logs 308
- function keys
 - changing namelists 208
 - operations and control panels 201
 - showing 200
 - updating 89
 - using 200
- functions
 - See change log inventory utility (CSQJU003)
 - See MQSeries utility program (CSQUTIL)

functions, return codes from CSQUTIL 230

G

global trace
 initial setting 73
 start automatically 73
 glossary 491
 GMQADMIN, security class 400, 401
 GMQNLIST, security class 400
 GMQPROC, security class 400
 GMQQUEUE, security class 400
 group class, security 400
 group name, specifying for OTMA 71
 GRPLIST system initialization parameter 120

H

hardware errors 359
 header
 SMF type 115 record 371
 SMF type 116 record 385
 help
 CICS adapter 126
 operations and control panels 201
 HIGHRBA, utility function (CSQJU003) 264
 HTML (Hypertext Markup Language) xxi
 Hypertext Markup Language (HTML) xxi

I

I/O error
 marks active log as TRUNCATED 320
 occurrence 305
 queues 353
 ICHRIN03, started-task procedure table 456
 IDBACK parameter of CSQ6SYSP 70
 IDFORE parameter of CSQ6SYSP 71
 IEFSSNs, SYS1.PARMLIB member 41
 IFASMFDP, reporting program for SMF 366
 IMS
 abend U3042 177
 adapter 174
 application programming 25
 commands
 CHANGE SUBSYS 169, 173
 DEQUEUE TMEMBER 183
 DISPLAY OASN 173
 DISPLAY OASN SUBSYS 169
 DISPLAY SUBSYS 175
 START REGION 175
 START SUBSYS 169
 START TMEMBER 183
 STOP REGION 175
 STOP SUBSYS 169, 176
 STOP TMEMBER 183
 TRACE 169

IMS (continued)

 connection status 175
 connections to MQSeries 169
 control region 170
 controlling dependent region connections 174
 CSQQTRMN transaction 160
 definition of term xv
 disconnecting from dependent region 175
 dynamic call stub, linking 162
 in-doubt units of recovery 284, 288
 initializing 170
 log record 284
 MQSeries-IMS bridge 25
 recovery considerations when using MQSeries 312
 related problems 357
 related publications xxiv
 reset Tpipe 21
 resynchronizing the bridge 184
 second user ID, determining 432
 system administration and operation 25
 thread 171
 trigger monitor 160
 user IDs, security checking 432
 IMS adapter 161, 180
 address space user ID 395
 connection security 405
 connection status 175
 CSQBSTUB stub program 25
 CSQQDEFV, subsystem definition table 165
 CSQQDEFX, macro 165, 166
 CSQQSTUB stub program 25
 CSQQSTUB, stub program 159
 CSQQTRMN transaction 160
 defining MQSeries to it 165
 dependent regions of IMS 174
 displaying in-doubt units of recovery 172
 IMSID option 170
 installing 161
 language interface token (LIT) 166
 logical terminal (LTERM) 170
 maximum number of connections (CTHREAD) 69
 residual recovery entry (RRE) 173
 RESLEVEL 426
 restart, what happens 293
 second user ID 426, 432
 security checking 426
 SSM EXEC parameter 164
 SSM specification options 165
 starting CSQQTRMN 177
 stopping CSQQTRMN 178
 subsystem member entry in IMS.PROCLIB 162
 threads, displaying 172
 trigger monitor 160
 IMS bridge
 age, specifying for OTMA 72
 Commit mode, synchronization 184

Index

- IMS bridge (*continued*)
 - controlling queues 183
 - customizing 181
 - deleting messages 183
 - description 179
 - druexit name, specifying for OTMA 71
 - group name, specifying for OTMA 71
 - illustration 179
 - member name, specifying for OTMA 71
 - OTMA parameters 71
 - persistent messages 475
 - resynchronizing 184
 - security 449
 - starting 179
 - stopping 179
 - storage class 181
 - suppressing console messages 91
 - Tpipe name 72
 - IMS problem
 - application terminates 357
 - IMS not operational 357
 - unable to connect to MQSeries 358
 - IMS transactions, submitting 180
 - IMS.PROCLIB library 162, 170, 174
 - in-doubt units of recovery 283, 285, 291, 293
 - causing inconsistent state 280
 - resolution 289
 - resolving, in CICS 283
 - resolving, in IMS 284
 - resolving, in RRS 285
 - INBUFF parameter of CSQ6LOGP 74
 - include files
 - general-use programming interface 471
 - incorrect access 465
 - indexed queues, effect on restart time 282
 - information messages, suppressing 91
 - initialization input data sets
 - customizing 52
 - editing 53
 - formats 53
 - migration considerations 94
 - MQSeries-supplied samples 54
 - initiation queue 11
 - defining for CICS 58
 - INITPARM system initialization parameter 120
 - input buffer size (INBUFF) 74
 - installable features 33
 - installation verification program (IVP) 59
 - distributed queuing 102
 - sample output 105
 - queue manager 99
 - installing
 - See also* customizing
 - CICS adapter 118
 - IMS adapter 161
 - introduction 26
 - installing (*continued*)
 - security 456
 - Internet Gateway feature 34
 - interpreting
 - buffer manager statistics 377
 - log manager statistics 381
 - replies to messages 220
 - intersystem connection (ISC) links 113
 - INTERVAL, attribute of ALTER SECURITY 55, 441
 - introduction
 - introduction to MQSeries 3
 - investigating performance 365
 - IPCS job, formatting dumps 90
 - IPCS list, updating 90
 - IPCS VERBEXIT 90
 - IPL (initial program load)
 - when required for MQSeries 37
 - IRC and the CICS adapter 120
 - ISC LU 6.2 connection 481
 - ISPF
 - See also* operations and control panels
 - effect of split screen 71
 - installing panels permanently 88
 - menu, updating 89
 - operations and control panels, setting up 88
 - panel 21
 - showing keys (PFSHOW) 200
 - ISPLLIB, concatenation 88
 - ISPMLIB, concatenation 88
 - ISPPLIB, concatenation 88
 - ISPSLIB, concatenation 88
 - ISPTLIB, concatenation 88
 - issuing commands 189, 238
 - IUCV
 - migration considerations 94
 - IUCV interface to TCP/IP 85
 - IVP
 - See* installation verification program (IVP)
 - IVP (installation verification program)
 - distributed queuing 102
 - sample output 105
 - queue manager 99
- ## J
- Japanese language feature 39, 227
 - Japanese language letter 33
- ## K
- KEEPALIVE value, TCP/IP 85
- ## L
- Language Environment, applying service 477

- language interface token (LIT) 163
 - language letter 33
 - language, national 39
 - layout
 - type 115 SMF records 371
 - type 116 SMF records 385
 - libraries, after installation 34
 - link list, updating 46
 - listener
 - controlling 21
 - restarting with ARM 296
 - listener restart time 84
 - LIT (language interface token) 163
 - load balancing on page sets 327
 - load libraries, APF authorization of 47
 - load management 15
 - LOAD, utility function 254
 - local queue 10
 - system default 55
 - locating archive log data sets to be deleted 318
 - log
 - active 299
 - archive 299
 - changing log inventory utility (CSQJU003) 256
 - copy active to archive 20
 - determining inventory contents 319
 - dual logging 300
 - active 300, 306
 - archive 300, 306
 - synchronization 304
 - error recovery procedures 338
 - introduction 17
 - log print utility (CSQ1LOGP) 266
 - managing 299
 - number of buffers per write 76
 - print log map utility (CSQJU004) 265
 - records 284
 - logical 301
 - physical 301
 - types 300
 - recovering from problems
 - active log 338
 - archive log 343
 - recovery 323
 - structure 301
 - log data sets
 - creating 61
 - restart on losing 355
 - single or dual 74
 - storage required 61
 - log initialization parameters, setting 68
 - log manager statistics 380, 381
 - log print utility (CSQ1LOGP) 266—268
 - extract log records 266
 - invoking 266
 - output 268
 - log print utility (CSQ1LOGP) (*continued*)
 - print log records 266, 319
 - time stamp 319
 - what it does 266
 - log RBA value, modifying 256
 - log RBA, updating the highest written 264
 - log records
 - number between checkpoints 71
 - logging parameters
 - default 74
 - setting 74
 - logging process 302
 - logging, single and dual 74, 300
 - LOGLOAD parameter of CSQ6SYSP 71
 - logs and recovery 17
 - lowercase queue names
 - CICS adapter 128
 - operations and control panels 209
 - LSTRTMR parameter of CSQ6CHIP 84
 - LU 6.2
 - LU name 84
 - LUADD 84
 - maximum number of current channels 85
 - restart interval after failure 84
 - LU 6.2 and ARM 297
 - LU 6.2 connections (CICS mover) 481
 - LU name for outbound transmissions 84
 - LU62ARM parameter of CSQ6CHIP 84
 - LU62CHL parameter of CSQ6CHIP 85
 - LUNAME parameter of CSQ6CHIP 84
- ## M
- macros
 - CSQ6ARVP 67
 - CSQ6LOGP 67, 74
 - CSQ6SYSP 67
 - general-use programming interface 469
 - product-sensitive programming interface 471
 - maintaining consistency after errors 277
 - MAKECLNT, keyword of COMMAND function 239, 241
 - MAKEDEF, keyword of COMMAND function 238, 240
 - managing
 - BSDS 306, 321
 - MQSeries log 315
 - page sets 325
 - MAXALLC parameter of CSQ6LOGP 74
 - MAXARCH parameter of CSQ6LOGP 74
 - MAXFILEPROC 85
 - maximum number of uncommitted messages 54, 245
 - maximum page set extents 66
 - MAXSMSGS 54, 245
 - MCA user ID, distributed queuing 433
 - media recovery 359

Index

- member class, security 400
- member name, specifying for OTMA 71
- message
 - calculating the space required 63
 - channels 13
 - illustration 4
 - maximum length 5
 - retrieval algorithms 6
 - storing 15
 - types 5
 - what they are 4
- message channel 13
- message descriptor 4
- message manager accounting 388
- message manager statistics 374
- message processing program (MPP) 174
- message queue interface (MQI) 3
 - calls 8
- message queuing, what it is 3
- message routing code (ROUTCODE) 72
- message-driven processing 3
- messages
 - CICS adapter 125
 - discarded 219
 - incorporating MQSeries commands 216
 - information, suppressing 91
 - interpreting replies to MQSeries commands 220
 - maximum number of uncommitted 54, 245
 - on the system-command input queue 217
 - suppressing 91
 - undelivered, security 411
 - user 201, 212
 - violation, security 464
 - waiting for replies to 218
- MGCR 431
- MGCR and MGCRE 189
- migration
 - testing 99
- migration considerations
 - ARM 93
 - changes to installation process 93
 - channel initiator 94
 - channel initiator parameters 93
 - channel initiator security 94
 - CICS bridge 93
 - clusters 94
 - coexistence with earlier versions 98
 - euro currency support 95
 - initialization input data sets 94
 - IUCV 94
 - OpenEdition sockets 94
 - operations and control panels 95
 - queue objects 95
 - RRS 94
 - software levels 93
 - storage classes 94
 - migration considerations (*continued*)
 - system parameters 93
 - TCP/IP 94
- model queue 10
 - system default 55
- model queues
 - security 409
- modifying an MQSeries-CICS connection 132
- monitoring
 - CICS connection activity 135
 - DISPLAY commands 364
 - IMS connection activity 174
 - performance 363
 - resource usage 363
 - tools 363
- mounting, archive log (ARCWTOR) 78
- mover
 - CICS
 - See distributed queuing using CICS ISC
- moving queues 328
- MPP (message processing program) connection control 174
- MQADMIN, security class 400, 401
- MQCLOSE options, security 419
- MQCMDS, security class 400
- MQCONN, security class 400, 404
- MQGET
 - in administration programs 214
 - security 408
- MQI (message queue interface)
 - calls 8
 - support, CICS adapter 110
 - what it is 3
- MQI channel 13
- MQNLIST, security class 400
- MQOPEN/MQPUT1 options, security 406, 419
- MQPROC, security class 400
- MQPUT
 - in administration programs 214
 - security 408
- MQQUEUE, security class 400
- MQSC command
 - introduction 18
- MQSeries commands
 - ARCHIVE LOG 303, 315
 - DEFINE PSID 325
 - DISPLAY SECURITY 403
 - DISPLAY THREAD 286
 - issuing from TSO panels 191
 - REFRESH SECURITY 403
 - remote queue manager 214
 - RESOLVE INDOUBT 286
 - resource security profiles 422
 - security 423
 - security profiles 421
 - START TRACE 368

MQSeries commands (*continued*)
 STOP TRACE 368

MQSeries publications xvi

MQSeries subsystems
 running different versions 98

MQSeries utility program (CSQUTIL) 229—256
 COMMAND function 238
 COPY function 246
 COPYPAGE function 233
 EMPTY function 252
 FORMAT function 231
 introduction 229
 invoking 229
 issuing commands from 191
 LOAD function 254
 monitoring the progress of 230
 moving queues 328
 page set management 231, 328
 queue management functions 245
 RESETPAGE function 235
 return codes 230
 SCOPY function 249
 SDEFS function 243
 security 431
 syntax checking 230

MQSeries-IMS bridge
 See IMS bridge

Msgld field, administration programs 218

multi-region operation (MRO) restrictions 113

multi-tasking, CICS adapter 114

multi-volume archive log data sets 304

MVS
 See OS/390

N

name of LU to use 84

namelist 13
 system default 55

namelists 208
 security 396, 414

names of objects 7

national language feature 33

network considerations for ARM 296

network ID (NID)
 See NID (network ID)

NEWLOG, utility function (CSQJU003) 258, 321, 323

NID (network ID) 286, 289

NODEFINE storage class 57

nonpersistent messages 213

NSYSLX, value of 45

O

object
 channel 13

object (*continued*)
 namelist 13
 names 7
 process definition 14
 queue 10
 queue manager 8
 storage class 15
 types 7

object definition
 storing 15

objects
 backing up definitions 309
 defining 207
 operations and control panels 207
 supplied samples 55
 system default 55

off-loading
 active log 303
 description 302, 304
 diagram of process 303
 errors during 304, 320
 messages issued during 304
 process description 302
 relationship to other logging events 303

OFFLOAD parameter of CSQ6LOGP 75

OpenEdition 22

OpenEdition sockets
 migration considerations 94
 security 51

OpenEdition sockets interface to TCP/IP 85

opening the system-command input queue 214

operating
 basic operations 189
 introduction 27

operations and control panels
 changing function keys 89
 changing the subsystem ID 202
 coexistence with earlier versions 98
 example of 202
 function keys 201
 installing permanently 88
 introduction 21
 invoking 197
 libraries 88
 migration considerations 95
 performance 16
 queue manager default 200
 RESLEVEL 425
 rules for using 209
 security checking 425
 setting up 88
 system command objects 56
 user messages 201
 using 197
 working with object definitions 207

Index

- operator commands
 - CICS adapter 143
 - IMS adapter 169
 - issuing 189
 - operations and control panels 197
 - orderly shutdown
 - CICS adapter 144
 - MQSeries 279
 - OS/390
 - APF authorized libraries 47
 - console, connecting from 190
 - environment 22
 - issuing MQSeries commands 192
 - link list, updating 46
 - MQSeries considerations 22
 - parmlibs, updating 41
 - program properties table, updating 48
 - related publications xxiii
 - SNAP dump 87, 110, 159
 - subsystem name table, updating 41
 - WLM 26
 - workload manager 26
 - OS/390 Automatic Restart Manager
 - See ARM
 - OS/390 OpenEdition 22
 - OTMA 179
 - DRU exit sample 475
 - pre-routing exit sample 473
 - OTMA exit 473
 - OTMACON parameter of CSQ6SYSP 71
 - out of space on active log 338
 - OUTBUFF parameter of CSQ6LOGP 75
 - output buffer, logs (OUTBUFF) 75, 302
- P**
- page set
 - backing up 332
 - define 20
 - display current state 20
 - dynamic expansion 65
 - maximum limit 66
 - how messages are stored 64
 - illustration 16, 17
 - number used 65
 - problems 353
 - reducing the size 330
 - sample 62
 - storage management 15
 - what they are 15
 - page set identifier (PSID)
 - specifying 15
 - page set problem
 - I/O error 353
 - page set full 353
 - page set zero 15
 - managing 16
 - performance 16
 - storage requirements 63
 - page set zero, migration 95
 - page sets
 - adding 62, 325
 - AMS REPRO 332
 - backing up 308, 331
 - control records 301
 - copying 233, 235
 - COPYPAGE 233
 - creating a point of recovery 331
 - defining 62
 - dynamic expansion 62
 - expanding 233, 326
 - formatting 231
 - full 325, 353
 - initialization input data sets 54
 - load balancing 327
 - managing 325
 - performance 378
 - recovery 308, 333
 - RESETPAGE 235
 - resetting the log 235
 - storage requirements 63
 - utility functions 231
 - PAGES keyword of FORMAT 231
 - panels
 - blank fields in 209
 - coexistence with earlier versions 98
 - installing 89
 - issuing commands from 191
 - operations and control 197, 202
 - rules for using 209
 - PARM option, START QMGR command 194
 - parmlibs, updating 41
 - passwords
 - archive log data set 323
 - data sets 258
 - supply for archive log 261
 - PDF (Portable Document Format) xxi
 - pending events
 - deferred connection 112
 - termination notification 112, 121
 - performance
 - buffer pool management 377
 - buffer pools 17
 - compacting archive log 79
 - DASD operations 377
 - DISPLAY commands 364
 - effect of indexed queues 282
 - effect of MQSeries trace 369
 - monitoring 363
 - of CICS adapter 115
 - operations and control panels 16

- performance (*continued*)
 - page set zero 16
 - problems 364, 365
 - sample SMF records 382
 - SMF trace 366
 - snapshots 364
 - symptoms of reduced 365
 - Performance Reporter 367
 - permanent dynamic queues, problems 16
 - permanent queue 10
 - permanent queues 5
 - PF keys
 - See function keys
 - PFSHOW, ISPF command 200
 - PING CHANNEL command
 - security 423
 - PL/I, testing customization 102
 - PLTPI (program list table post initialization) 119
 - starting the CICS adapter 122
 - point of consistency
 - CICS 275
 - description 273
 - IMS 275
 - point of recovery 308
 - creating 331
 - Portable Document Format (PDF) xxi
 - PostScript format xxii
 - PPT (program properties table)
 - example 48
 - updating 48
 - pre-routing exit 473
 - predefined queues 5
 - previous versions
 - migrating from
 - coexistence with earlier versions 98
 - print log map utility (CSQJU004) 265
 - invoking 265
 - PRIQTY parameter of CSQ6ARVP 79
 - problem determination
 - IMS 357
 - performance 364
 - security 464
 - problems
 - operations and control panels 16
 - permanent dynamic queues 16
 - procedures
 - channel initiator 50
 - queue manager subsystem 49
 - process
 - definitions 14
 - security 395
 - security profile 413
 - system default 55
 - proclibs 49
 - profile, RACF 401
 - for alternate user security 415
 - profile, RACF (*continued*)
 - for command resources 422
 - for command security 421
 - for connection security
 - batch/TSO adapter 404
 - CICS adapter 404
 - distributed queuing 405
 - IMS adapter 405
 - RRS adapter 404
 - for context security 417
 - for namelists 414
 - for process security 413
 - for queue security 406
 - RESLEVEL 428
 - switch 401
 - used to protect MQSeries resources 404
 - program list table (PLT) 119
 - program properties table
 - See PPT (program properties table)
 - program, administration 211
 - PROTECT parameter of CSQ6ARVP 79
 - PSID (page set identifier)
 - specifying 15
 - publications
 - MQSeries xvi
 - related xxiii
- ## Q
- QIST, data manager statistics record 375
 - QJST, log manager statistics record 380
 - QMAC, message manager accounting record 388
 - QMCCSID (queue manager coded character set identifier) 72
 - QMCCSID parameter of CSQ6SYSP 72
 - QMST, message manager statistics record 374
 - QPST, buffer manager statistics record 376
 - queue
 - alias 10
 - attributes 9
 - channel 11
 - channel initiation 11
 - channel reply information 11
 - channel sequence number (CICS mover) 481
 - channel synchronization 11
 - cluster 10
 - cluster command 11
 - cluster repository 11
 - cluster transmission 12
 - dead-letter 12
 - defining 9
 - dynamic 5
 - event 12
 - illustration 8
 - initiation 11
 - local 10

Index

- queue (*continued*)
 - mapping to page sets 15
 - model 10
 - permanent 5, 10
 - predefined 5
 - remote 10
 - required for clusters 11
 - required for commands 12
 - required for distributed queuing 11
 - required for events 12
 - required for triggering 11
 - system default 13
 - system-command
 - input 12
 - temporary 10
 - transmission 11
 - used with MQSeries 10
 - what they are 5
 - where they reside 6
 - WLM-managed 26
 - queue management utility functions 245
 - queue manager
 - commands 8
 - display cluster information 21
 - illustration 8
 - installation verification program 99
 - relationship to queues 6
 - running multiple versions 98
 - starting 20, 192
 - stopping 20, 195
 - what it is 8
 - queue manager cluster
 - See cluster
 - queue manager coded character set identifier (QMCCSID) 72
 - queue manager event
 - introduction 18
 - queue object
 - types 10
 - using 10
 - queue objects
 - migration considerations 95
 - queues 117
 - alter attributes, security 423
 - backup 307
 - channel 56
 - cluster 57
 - copying 233, 246
 - copying (offline) 249
 - dead-letter 58
 - defining local 203
 - emptying 252
 - indexed, effect on restart time 282
 - LOAD function 254
 - moving them 328
 - profiles for security 406
 - queues (*continued*)
 - recommended, sample 57
 - recovery 307
 - reply-to model 214
 - reserved names 117
 - restoring messages 254
 - security 395, 406
 - supplied samples 55
 - system-administration 56
 - system-command 214
 - input 214
 - reply-to model 214
 - transmission, default 58
 - QUIESCE 195, 279, 280
 - option of STOP QMGR command 280
 - parameter, table of effects 279
 - stop mode 195
 - QUIESCE MODE, of ARCHIVE LOG utility
 - function 315
 - QUIESCE parameter of CSQ6ARVP 80
 - QWHS, message manager accounting record 388
- ## R
- RACF
 - See *also* profile, RACF
 - See *also* security
 - authority, dead-letter queue 412
 - authorization
 - ICHRIN03 456
 - STARTED class 456
 - started-task procedure table 456
 - to MQSeries data sets 457
 - commands for CSQ4IVP1 100, 103
 - profiles 401
 - related publications xxiv
 - security classes 400
 - RACF profiles 450
 - railroad diagrams, how to read 228
 - RBA (relative byte address)
 - description 300
 - range shown in messages 304
 - range specified in active log 321
 - RDO (resource definition online) 118
 - recommended queues sample, CSQ4INYG 57
 - record subtypes, SMF type 115 records 372
 - records
 - type 115 (SMF) 366
 - type 116 (SMF) 366
 - RECOVER BSDS command
 - security 423
 - recovery 335
 - active log problems 338
 - application program 299
 - basic operations 189
 - BSDS
 - errors 346

- recovery (*continued*)
 - BSDS (*continued*)
 - log inventory 317
 - CICS
 - manually recovering units of recovery 286
 - resolving in-doubt units of recovery 283
 - COPY 335
 - creating a point of 331
 - data with DFHSM 311
 - description 333
 - example scenarios 337
 - IMS
 - manually recovering units of recovery 288
 - resolving in-doubt units of recovery 172, 284
 - resynchronizing the bridge 184
 - introduction 17, 28
 - logs 323
 - MQSeries-related problems
 - active log problems 338
 - archive log problems 343—344
 - BSDS 346—352
 - page set problems 353
 - object definitions, backing up 309
 - page sets 332
 - point of 331
 - procedure, IMS units of recovery 288
 - restart 281
 - RRS
 - manually recovering units of recovery 290
 - resolving in-doubt units of recovery 285
 - single BSDS 350
 - starting 191—195
 - subsystem 299
 - system procedures 307
 - tokens 286
- redo records 301
- reduced performance, symptoms of 365
- reducing the size of a page set 330
- REFRESH SECURITY command 403
 - security 423
 - using 442
- region error options (REO) 163, 174
- registering with ARM 296
- relative byte address (RBA) 300
 - See also* RBA (relative byte address)
- remote queue
 - system default 55
- remote queues 10
 - command resource checking 422
 - security 410
- REMOTE storage class 57
- REO (region error options) 163, 174
- replies, examples 220
- reply message descriptor 219
- reply messages 5, 218
- reply-to queue
 - attributes 214
 - defining 214
 - opening 215
 - system-command 56
- report message 5
- repository queue, clustering 11
- REPRO command of access method services 332, 350
- request message 5, 217
- RESET CHANNEL command
 - security 423
- RESETPAGE, utility function (CSQUTIL) 235
- resetting page sets 235
- residual recovery entry (RRE)
 - See* RRE (residual recovery entry)
- RESLEVEL
 - auditing 430, 437
 - checking CICS user IDs 425
 - distributed queuing 427
 - IMS adapter 426
 - security profile 394
 - usage notes 430
 - user IDs associated with 428
 - using 425
- RESOLVE commands
 - security 423
- RESOLVE INDOUBT command
 - free locked resources 286
- resolving
 - in-doubt units of recovery 172
 - units of recovery 286, 290
- resource definition online (RDO) 118
- Resource Measurement Facility (RMF) 367
- resource protection 4, 395
- resource recovery services (RRS)
 - adapter 23
 - introduction 23
 - migration considerations 94
- resource security
 - alias queues 422
 - API 395, 419
 - commands 397, 422
 - remote queues 422
- resource-level security checking by CICS adapter 445
- RESOURCE_TYPE, CICS adapter 117
- RESPTIME, keyword of COMMAND function 239
- restart 279
 - See also* recovery
 - after abnormal termination 277
 - after losing data sets 356
 - after losing logs 355
 - CICS adapter 283, 291
 - cold start 356
 - conditional 262
 - distributed recovery environment 277

Index

- restart (*continued*)
 - effect of indexed queues 282
 - effect of lost connections 283
 - IMS 284
 - IMS adapter 293
 - MQSeries 284
 - phase 275
 - normal 281
 - OS/390 Automatic Restart Manager 294
 - parameter, table of effects 279
 - process 273, 279, 281
 - RRS adapter 285
 - user messages 292
 - with ARM 294
- RESTART option of STOP QMGR command 280
- restart timer, listener 84
- restoring
 - messages to a queue 245
- restricting access using alias queues 408, 411
- resynchronization of CICS and MQSeries
 - resources 283
- resynchronization of IMS and MQSeries resources 284
- resynchronization of RRS and MQSeries
 - resources 285
- retention period, archive logs (ARCRETN) 77, 317
- retrieval algorithms, for messages 6
- return codes, from utility functions 230
- RMF (Resource Measurement Facility) 367
- ROUTCDE parameter of CSQ6SYSP 72
- route codes, archive log (ARCWRTC) 77
- routing code, message (ROUTCDE) 72
- routing commands
 - in a sysplex 44
 - using CPFs 42
- RRE (residual recovery entry) 173
 - logged at IMS checkpoint 284
 - not resolved 284
- RRS
 - applying service 477
 - units of recovery 290
- RRS (resource recovery services)
 - adapter 23
 - introduction 23
 - migration considerations 94
- RRS adapter 23
 - connection security 404
 - installing 87
 - recovery 285
- rules for using the operations and control panels 209
- RVERIFY SECURITY command
 - security 423
 - using 442
- S**
- sample
 - ARM policy 295
 - channel definitions (CICS mover) 480
 - defining page sets 62
 - destination resolution exit 475
 - initialization input data set members 54
 - linking the IMS dynamic call stub 162
 - OTMA pre-routing exit 473
 - output from CSQ4IVPX 105
 - security scenario 458
 - SMF accounting record 390
 - SMF statistics records 382
 - startup procedure 49
- sample data set members 37
- SCOPY, CSQUTIL function 249
- SCSQxxxx contents 34
- SDEFS, CSQUTIL function 243
- second user ID, IMS adapter 426, 429, 432
- SECQTY parameter of CSQ6ARVP 80
- security
 - See also* profile, RACF
 - See also* RACF
 - alternate user 396
 - API quick-reference table 419
 - archive log 79
 - auditing considerations 437
 - automating starting of CKTI 446
 - blank user IDs 436
 - CICS adapter 445
 - transactions 445
 - user IDs 446
 - CICS bridge 447
 - CKSG user IDs 446
 - CKTI 446
 - clustering 455
 - command 397
 - summary table 423
 - command resource 397
 - connection 395, 404, 430
 - context 396
 - controlling 20
 - controls and options 394
 - customizing 444
 - data sets 456
 - default user ID 69, 430
 - displaying status 443
 - distributed queuing 453
 - distributed queuing (using CICS ISC) 482
 - example scenario 458
 - implementation 394
 - implementation checklist 439
 - IMS 449
 - installation tasks 51, 456
 - installation verification program
 - distributed queuing 103

- security (*continued*)
 - installation verification program (*continued*)
 - queue manager 100
 - INTERVAL attribute 55, 441
 - management 393, 441
 - migration considerations 94
 - MQCLOSE/MQOPEN/MQPUT1 options 419
 - namelist 396
 - OTMA 449
 - problem determination 464
 - process 395
 - profile, RESLEVEL 394, 425
 - propagating CKTI user IDs 446
 - queues 395
 - alias 407, 422
 - dead-letter 411
 - dynamic 410
 - model 409
 - profiles 406, 423
 - remote 410, 422
 - transmission 417, 419, 422
 - refreshes 442
 - subsystem 394
 - terminal user IDs 446
 - TIMEOUT attribute 55, 441
 - undelivered messages 411
 - universal access (UACC) levels 436
 - user ID timeouts 441
 - user IDs 430
 - using RACF classes 400
 - utilities 431
- Security Server
 - See also* RACF
 - related publications xxiv
- self-defining section
 - SMF type 115 records 371, 373
 - SMF type 116 records 385, 387
- service considerations 477
- service goal 26
- SETSSI command 41
- shutdown
 - See* terminating
- shutting down
 - CICS bridge 155
- Simplified Chinese language feature 39
- Simplified Chinese language letter 33
- single BSDS (TWOBSDS) 75
- single logging 300
 - establishing 309
 - specifying for active log (TWOACTV) 75
 - specifying for archive log (TWOARCH) 75
- single phase commit 275
- SIT (system initialization table) 121
 - GRPLIST parameter 120
 - INITPARM parameter 120
 - PLTPI parameter 120
- SMF (System Management Facility) 366
 - accounting record sample 390
 - buffers 366
 - CSQ6SYSP, specifying parameters 68
 - data records
 - type 115 371
 - type 116 385
 - gathering (STATIME) 73
 - processing type 115 records 372
 - processing type 116 records 386
 - recording trace data for 366
 - reporting data in (IFASMFDP) 366
 - self-defining section
 - type 115 records 371
 - type 116 records 385, 387
 - starting automatically (SMFSTAT) 72
 - statistics records sample 382
 - type 115 header 371
 - type 115 record layout 371
 - type 115 record subtypes 372
 - type 115 self-defining section 373
 - type 116 header 385
 - type 116 record layout 385
- SMFACCT parameter of CSQ6SYSP 72
- SMFSTAT parameter of CSQ6SYSP 72
- SMS (Storage Management Subsystem)
 - See* Storage Management Subsystem (SMS)
- SNAP dump
 - Batch/TSO adapter 87
 - CICS adapter 110
 - IMS adapter 159
- snapshots, performance 364
- sockets
 - security 51
- sockets interface to TCP/IP 85
- softcopy books xxi
- software levels 93
- space allocation
 - archive logs, block size (BLKSIZE) 78
 - logs, primary (PRIQTY) 79
 - logs, secondary (SECQTY) 80
 - units, logs (ALCUNIT) 77
- SPT (started-task procedure table) 456
- ssid.DEAD.QUEUE
 - See* dead-letter queue
- SSM (subsystem member)
 - contains control information 170
 - entry in IMS.PROCLIB 162
 - error options 174
 - EXEC parameter 164
 - specification options 165
 - specified on EXEC parameter 174
- START CMDSERV command 213
- START commands
 - security 423

Index

- start options for MQSeries 194
- START QMGR command
 - from OS/390 console 191
 - options 194
- START REGION, command of IMS 175
- START SUBSYS, command of IMS 169
- START TMEMBER, command of IMS 183
- START TRACE command 368
- start-up messages (MQSeries) 192
- STARTED RACF class 456
 - authorization to data sets 457
- started task procedure
 - authorization 456
 - creating for channel initiator 50
 - creating for queue manager 49
 - security 49
- starting
 - after an abend 195
 - CICS bridge 154
 - CICS-MQSeries connection
 - from a CICS program 129
 - from the command line 128
 - using the CICS adapter control panels 127
 - command server 213
 - IMS-MQSeries connection 170
 - MQSeries 191, 192, 195
 - MQSeries trace 368
 - OS/390 Automatic Restart Manager 294
 - with ARM 294
- startup procedure, CSQ4MSTR 49
- STATIME parameter of CSQ6SYSP 73
- statistics
 - buffer manager 376
 - buffer pool 378
 - CICS adapter 116, 364
 - data manager 375
 - eye catcher strings 382
 - gathering time interval 73
 - log manager 380, 381
 - message manager 374
 - sample SMF records 382
 - security 438
 - starting automatically 72
- STOP CMDSERV command 213
- STOP commands
 - security 423
- STOP QMGR command
 - MODE(FORCE) 195, 279
 - MODE(QUIESCE) 195, 279
 - MODE(RESTART) 195, 279
- STOP REGION, command of IMS 175
- STOP SUBSYS, command of IMS 169, 176
- STOP TMEMBER, command of IMS 183
- STOP TRACE command 368
- stopping
 - See also* terminating
- stopping (*continued*)
 - CICS bridge 155
- storage class
 - errors 58
 - illustration 16
 - migration considerations 94
 - sample 57
 - storage management 15
 - system default 55
 - when to change 58
- storage class sample, CSQ4INYG 57
- storage class, defining 15
- storage class, IMS bridge 181
- storage management 15
- Storage Management Subsystem (SMS) 26, 317
 - archive log data sets 311
- storage requirement
 - BSDS 61
 - CSA 28
 - messages 63
 - page set 63
 - trace table 73
- storage, archive planning 310
- stub program
 - CSQBRRSI, for Batch/TSO RRS adapter 23
 - CSQBRRSI, for RRS adapter 23
 - CSQBRSTB, for Batch/TSO RRS adapter 23
 - CSQBRSTB, for RRS adapter 23
 - CSQBSTUB for Batch/TSO adapter 23
 - CSQBSTUB for IMS adapter 25
 - CSQCSTUB for CICS adapter 24
 - CSQQSTUB for IMS adapter 25
- stub programs
 - coexistence with earlier versions 98
 - CSQCSTUB for CICS adapter 111, 121
 - CSQQSTUB, for IMS adapter 159
- subsystem ID, changing 202
- subsystem member (SSM)
 - See* SSM (subsystem member)
- subsystem name table, updating 41
- subsystem recovery 299
 - See also* recovery
- subsystem security 394
 - See also* security
- subsystems, defining
- successful collection, accounting data 368
- suppressing information messages 91
- switches, security 401
- syncpoint
 - See* point of consistency
- syncpoint protocols 275
- syncpoint-bypassing, CICS adapter 116
- syntax diagrams, how to read 228
- SYS1.PARMLIB member
 - APPCPMxx 84

- SYS1.PARMLIB members
 - IEFSSNss 41
 - SYSEXEC, concatenation 88
 - sysplex
 - routing commands 44
 - SYSPROC, concatenation 88
 - system administration
 - CICS 24
 - IMS 25
 - MQSC commands 19
 - MQSeries commands 189, 191
 - using application programs 211
 - system command sample, CSQ4INSG 55
 - system control command
 - summary 20
 - system control commands
 - for starting MQSeries 191
 - system default
 - objects 55
 - queues 13
 - sample, CSQ4INPX 60
 - sample, CSQ4INSG 55, 57
 - sample, CSQ4INYC 59
 - sample, CSQ4INYG 58
 - system initialization table (SIT)
 - See SIT (system initialization table)
 - system linkage index, reviewing the number 45
 - System Management Facility (SMF)
 - See SMF (System Management Facility)
 - system monitoring, DISPLAY commands 364
 - system object sample, CSQ4INSG 55
 - system parameter module
 - CSQZPARM 194
 - displaying settings 67
 - invoking 67
 - tailoring 67
 - system parameters
 - migration considerations 93
 - setting 68
 - system security 456
 - SYSTEM storage class 57
 - migration considerations 94
 - system-administration
 - objects 56
 - system-command
 - input queue 12, 56, 191
 - default attributes 56
 - defining 214
 - opening 214
 - putting messages on 217
 - objects 56
 - reply-to model queue 12, 56, 214
 - SYSTEM.ADMIN.CHANNEL.EVENT queue 12
 - SYSTEM.ADMIN.PERFM.EVENT queue 12
 - SYSTEM.ADMIN.QMGR queue 12
 - SYSTEM.CHANNEL.COMMAND queue 481
 - SYSTEM.CHANNEL.INITQ queue 11
 - SYSTEM.CHANNEL.REPLY.INFO queue 11
 - SYSTEM.CHANNEL.SEQNO queue 481
 - SYSTEM.CHANNEL.SYNCQ queue 11
 - SYSTEM.CLUSTER.COMMAND.QUEUE 12
 - SYSTEM.CLUSTER.REPOSITORY.QUEUE 11
 - SYSTEM.CLUSTER.TRANSMIT.QUEUE 12
 - SYSTEM.COMMAND.INPUT queue 12
 - SYSTEM.COMMAND.REPLY.MODEL queue 12
- ## T
- target library high-level qualifier (thlqual) 34
 - task initiator (CKTI)
 - See CKTI transaction
 - tasks, displaying CICS 141
 - TCB
 - CICS adapter 114
 - number for exit programs 70
 - TCP/IP
 - address space name 85
 - interface type 85
 - KEEPALIVE value 85
 - MAXFILEPROC 85
 - maximum number of current channels 85
 - migration considerations 94
 - number of dispatchers 84
 - restart interval after failure 84
 - settings 86
 - TCP/IP and ARM 297
 - TCP/IP, applying service 477
 - TCPCHL parameter of CSQ6CHIP 85
 - TCPKEEP parameter of CSQ6CHIP 85
 - TCPNAME parameter of CSQ6CHIP 85
 - TCPTYPE parameter of CSQ6CHIP 85
 - temporary queues
 - See dynamic queues
 - terminal user IDs, CICS adapter 446
 - terminating
 - MQSeries 195, 279
 - MQSeries-CICS connection 144
 - from a CICS program 131
 - from the CICS adapter control panels 130
 - from the CICS command line 130
 - MQSeries-IMS connection 176
 - using QUIESCE, FORCE, and RESTART, table of effects 279
 - termination notification, CICS adapter 112
 - terminology used in this book 491
 - testing your queue manager 99
 - TGTQMGR, keyword of COMMAND function 239
 - thlqual
 - definition of term xv
 - thlqual (target library high-level qualifier) 34

Index

- thlqual.SCSQxxxx contents 34
 - thread cross reference 280
 - thread, display information 20
 - threads
 - attachment in IMS 171
 - CICS adapter termination 144, 279
 - displaying, IMS adapter 172
 - ID for CICS transactions 122
 - IMS termination 176
 - maximum number (CTHREAD) 69
 - stopping MQSeries 195
 - time stamps 282, 319
 - archive log (TSTAMP) 80
 - from BSDS 319
 - unequal in BSDS 347
 - time-independent applications 3
 - TIMEOUT, security attribute 55, 441
 - Tpipe
 - name 72
 - reset 21
 - trace
 - controlling 20
 - controlling MQSeries 368
 - effect on performance 369
 - specifying destinations 369
 - specifying keywords 368
 - starting automatically (TRACSTR) 73
 - trace table size (TRACTBL) 73
 - TRACE, command of IMS 169
 - trace, size of data space for channel initiator 86
 - trace, start automatically for channel initiator 86
 - tracing parameters, setting 68
 - TRACSTR parameter of CSQ6SYSP 73
 - TRACTBL parameter of CSQ6SYSP 73
 - transaction pipe
 - See Tpipe
 - transient data queue (TDQ), CKMQ 481
 - transient data queue (TDQ), CKQQ 125
 - transmission queue 11
 - See also security
 - default 58
 - transmission queue, clustering 12
 - TRAXSTR parameter of CSQ6CHIP 86
 - TRAXTBL parameter of CSQ6CHIP 86
 - trigger monitor 14
 - See also CKTI transaction
 - See also CSQQTRMN transaction
 - triggering
 - queues required 11
 - truncation, active log 304
 - TSO
 - See also Batch/TSO adapter
 - formatting dumps 90
 - issuing MQSeries commands from 191
 - starting MQSeries from 191
 - TSO applications
 - improving portability 87
 - TSTAMP parameter of CSQ6ARVP 80
 - tuning
 - CICS bridge 154
 - tuning MQSeries 363
 - tuning, introduction 27
 - two-phase commit 275
 - TWOACTV parameter of CSQ6LOGP 75
 - TWOARCH parameter of CSQ6LOGP 75
 - TWOBSDS parameter of CSQ6LOGP 75
 - type 115 SMF records 366
 - type 116 SMF records 366
 - types of objects 7
- ## U
- U.S. English language letter 33
 - U3042 abend (IMS) 177
 - unauthorized access, protecting from 393
 - uncommitted messages, maximum number 20, 54, 245
 - undelivered messages, security 411
 - undelivered-message queue
 - See dead-letter queue
 - undo records 301
 - unit of recovery
 - maximum number of messages in 54, 245
 - unit of work (UOW) 291
 - UNIT parameter of CSQ6ARVP 80
 - units of recovery 273
 - CICS
 - in-doubt resolution 283
 - recovering manually 286
 - IMS
 - in-doubt resolution 172, 284
 - recovering manually 288
 - in-doubt
 - displaying in IMS 172
 - how they are resolved 283
 - recovering in IMS 172
 - log records 301
 - RRS
 - in-doubt resolution 285
 - recovering manually 290
 - universal access (UACC) levels 436
 - UNIX system services sockets interface to TCP/IP
 - See OpenEdition sockets interface to TCP/IP
 - UOW (unit of work) 291
 - updating
 - CSQINP2 121
 - OS/390 link list 46
 - OS/390 parmlibs 41
 - OS/390 subsystem name table 41
 - RACF security 393

- upgrade considerations 477
- upgrading from previous versions
 - coexistence with earlier versions 98
- US English language features 227
- user exits 114
- user ID
 - maximum age in OTMA 72
- user ID security
 - alternate user 428
 - Batch/TSO adapter
 - connection 404, 425, 430
 - RESLEVEL 425, 428
 - blank 436
 - checking 431
 - CICS adapter 445
 - address space 432
 - connection 404, 425, 430
 - RESLEVEL 425, 428
 - task 432
 - transactions 432
 - CKTI 446
 - CSQUTIL 425
 - default 69
 - distributed queuing 429
 - connection 405, 427
 - RESLEVEL 427
 - IMS adapter
 - address space 432
 - connection 405, 426, 430
 - RESLEVEL 426, 428
 - second user ID 426, 429, 432
 - number checked 425
 - operations and control panels 425
 - RESLEVEL profile 394, 425, 428
 - reverification 442
 - RRS adapter
 - connection 404
 - timeouts 441
- user messages 212
 - at start up 192
 - COUNT field 220
 - displaying from panels 201
 - from MQSeries commands, replies 220
- utilities
 - CSQ1LOGP
 - See log print utility (CSQ1LOGP)
 - CSQJU003
 - See change log inventory utility (CSQJU003)
 - CSQJU004
 - See print log map utility (CSQJU004)
 - CSQUTIL
 - See MQSeries utility program (CSQUTIL)
 - functions
 - ARCHIVE 261
 - CHECKPT 263
 - COMMAND 238
 - conditional restart 262
 - utilities (*continued*)
 - functions (*continued*)
 - COPY 246
 - COPYPAGE 233
 - CRESTART 262
 - DELETE 260
 - EMPTY 252
 - FORMAT 231
 - HIGHRBA 264
 - LOAD 254
 - NEWLOG 258
 - RESETPAGE 235
 - SCOPY 249
 - SDEFS 243
 - setting checkpoint records 263
 - updating the highest written log RBA 264
 - summary table 227
 - time stamp 319
 - unit of recovery, maximum number of messages 245
 - utility program
 - See MQSeries utility program (CSQUTIL)
 - Utoken 431

V

- VERBEXIT, IPCS 90
- violation messages, security 464
- virtual storage access method (VSAM)
 - See VSAM (virtual storage access method)
- volume dump and restore 333
- volume serial number 306
- VSAM (virtual storage access method) 258, 300, 302
 - control interval, block size 310

W

- WAIT option, CICS adapter 117
- waiting for replies to messages 218
- Windows Help xxii
- WLM 26
- WLMTIME parameter of CSQ6SYSP 73
- work, units of 286
- Workflow 26
- workload manager 26
 - queue scan interval 73
- writing
 - programs to administer MQSeries 211
 - to the active log 303
 - to the archive log 303
- WRTHRS parameter of CSQ6LOGP 76
- WTOR, MQSeries-related 195

Index

X

XCF

 group name, specifying for OTMA 71

 member name, specifying for OTMA 71

XRF (Extended Recovery Facility) 312

Sending your comments to IBM

System Management Guide

SC34-5374-00

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book. Please limit your comments to the information in this book and the way in which the information is presented.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

You can send your comments to IBM in any of the following ways:

- By mail, use the Readers' Comment Form.
- By fax:
 - From outside the U.K., after your international access code use 44 1962 870229
 - From within the U.K., use 01962 870229
- Electronically, use the appropriate network ID:
 - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
 - IBMLink: HURSLEY(IDRCF)
 - Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

- The publication number and title
- The page number or topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Readers' Comments

System Management Guide

SC34-5374-00

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

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

Be sure to print your name and address below if you would like a reply.

Name

Address

Company or Organization

Telephone

Email



You can send your comments POST FREE on this form from any one of these countries:

Australia	Finland	Iceland	Netherlands	Singapore	United States
Belgium	France	Israel	New Zealand	Spain	of America
Bermuda	Germany	Italy	Norway	Sweden	
Cyprus	Greece	Luxembourg	Portugal	Switzerland	
Denmark	Hong Kong	Monaco	Republic of Ireland	United Arab Emirates	

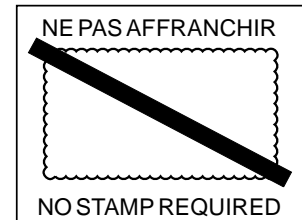
Cut along this line

If your country is not listed here, your local IBM representative will be pleased to forward your comments to us. Or you can pay the postage and send the form direct to IBM (this includes mailing in the U.K.).

2 Fold along this line

By air mail
Par avion

IBRS/CCR NUMBER: PHQ - D/1348/SO



**REPONSE PAYEE
GRANDE-BRETAGNE**

IBM United Kingdom Laboratories
Information Development Department (MP095)
Hursley Park,
WINCHESTER, Hants
SO21 2ZZ United Kingdom

3 Fold along this line

From: Name _____
Company or Organization _____
Address _____

EMAIL _____
Telephone _____

Cut along this line

4 Fasten here with adhesive tape



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC34-5374-00

