

MQSeries® for Tandem NonStop Kernel



System Management Guide

Version 2 Release 2.0.1

MQSeries® for Tandem NonStop Kernel



System Management Guide

Version 2 Release 2.0.1

Note!

Before using this information and the product it supports, ensure you read the general information under Appendix O, "Notices" on page 415.

Second Edition (February 1999)

This edition applies to the following product:

- IBM® MQSeries for Tandem NonStop Kernel, Version 2 Release 2.0.1

and to any subsequent releases and modifications until otherwise indicated in new editions.

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 1995, 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	xi
Who this book is for	xi
What you need to know to understand this book	xi
How to use this book	xi
Using the appendixes	xii
MQSeries publications	xii
MQSeries cross-platform publications	xii
MQSeries platform-specific publications	xv
MQSeries Level 1 product publications	xvii
Softcopy books	xvii
MQSeries information available on the Internet	xviii
Related publications	xix
Information about MQSeries on the Internet	xix

Part 1. Guidance	1
Chapter 1. Introduction	3
MQSeries and message queuing	3
Messages and queues	3
Objects	5
MQSeries queues	7
Process definitions	10
Channels	10
System default objects	10
Administration	10
Clients and servers	11
Extending queue manager facilities	12
Security	12
MQSeries for Tandem NSK and related products	13
Highlights of MQSeries for Tandem NSK V2.2.0.1	15
Chapter 2. Installing MQSeries for Tandem NSK Version 2.2.0.1	19
Preparing for installation	20
Installation procedure	21
TACL environment variables	26
Configuration of other NonStop Kernel resources	27
Files that exist after installation (native and nonnative)	27
Verifying your installation	32
Troubleshooting	34
Upgrading a Version 2.2 queue manager	34
Chapter 3. Customizing your system	35
Configuring an authorization service component	36
Enabling users to access MQSeries	36
Enabling communications support	37
Implementing data conversion	37
Defining the default and system objects	37
Specifying volumes for queue manager objects	38
Sharing queues using the name service	39

	Configuring a queue manager	39
	Specifying the location of the machine-wide INI file	39
	Specifying the location of the MQSeries executables	39
	Configuring MQSS Servers	40
	Specifying retrieval options for queues	40
	Configuring MQSeries to work with RDF	41
	Chapter 4. Using administration command sets	43
	Control commands	43
	MQSC commands	44
	PCF commands	44
	Comparing command sets	45
	TS/MP (Pathway) administration	48
	Chapter 5. Managing queue managers	59
	Getting started	59
	Guidelines for creating a queue manager	59
	Volume structure	70
	Working with queue managers	72
	Managing the command server for remote administration	77
	Using the Message Queue Management (MQM) facility	78
	Chapter 6. Administering local MQSeries objects	99
	Supporting application programs that use the MQI	99
	Issuing MQSC commands for administration	99
	Running MQSCs from text files	102
	Troubleshooting MQSC	106
	Working with local queues	108
	Displaying default object attributes	109
	Copying a local queue definition	110
	Changing local queue attributes	111
	Changing the volume of a local queue	111
	Changing the recalculation, update, and retrieval options for a local queue	112
	Reassignment of objects to MQSS Servers	113
	Clearing a local queue	113
	Deleting a local queue	114
	Browsing queues	114
	Working with alias queues	117
	Defining an alias queue	118
	Using other commands with queue aliases	119
	Working with model queues	119
	Defining a model queue	119
	Using other commands with model queues	120
	Managing objects for triggering	120
	Defining an application queue for triggering	121
	Defining an initiation queue	122
	Creating a process definition	122
	Displaying your process definition	123
	Chapter 7. Administering remote MQSeries objects	125
	Understanding channels and remote queuing	125
	Administering a remote queue manager	126
	If you have problems using MQSC remotely	131
	Creating a local definition of a remote queue	132

An alternative way of putting messages on a remote queue	133
Using other commands with remote queues	134
Creating a transmission queue	134
Using remote queue definitions as aliases	135
Chapter 8. Implementing security control	137
Understanding user IDs in the MQM user group	137
Protecting MQSeries resources	138
Understanding the Object Authority Manager (OAM)	138
Using the Object Authority Manager (OAM) commands	141
Access authorizations	144
Display authority command	144
Object Authority Manager (OAM) guidelines	144
Understanding the authorization specification tables	148
Understanding authorization files	153
Chapter 9. MQSeries dead-letter queue handler	157
Invoking the DLQ handler	157
DLQ handler rules table	158
Rules table conventions	163
How the rules table is processed	165
Example DLQ handler rules table	166
Chapter 10. Instrumentation and EMS events	169
MQSeries instrumentation events	169
Event Management Service (EMS) events	171
Chapter 11. Understanding transactional support and messaging	175
Using the NonStop TM/MP (Transaction Manager)	175
Configuration requirements for TM/MP and MQSeries for Tandem NSK	176
Chapter 12. Recovery and restart	179
Fault tolerance and recovery	179
Backing up and restoring	180
Recovery and restart of MQSS Servers	180
Chapter 13. Configuration files	183
What are configuration files?	183
MQSeries configuration file (MQSINI)	183
Queue manager configuration file (QMINI)	185
Editing configuration files	193
Chapter 14. Problem determination	195
Making a preliminary check	195
Common programming errors	198
What to do next	199
Application design considerations	202
Incorrect output	203
Error logs	207
Dead-letter queues	210
Configuration files and problem determination	210
Using MQSeries trace	210
First Failure Support Technology™ (FFST)	212

Part 2. Reference	215
Chapter 15. The MQSeries control commands	217
Control commands summary	217
Using names	218
How to read syntax diagrams	218
altmqfls (Alter queue file attributes)	220
altmqusr (Alter MQSeries user information)	223
cleanrdf (Perform RDF housekeeping)	225
cnv1520 (Convert V1.5.1 definitions to V2.2.0.1)	226
cnvclchl (Convert client channel definitions)	228
cnvmgs (Convert V1.5.1 messages to V2.2.0.1)	229
crtmqcvx (Data conversion)	232
crtmqm (Create queue manager)	234
dlmqm (Delete queue manager)	238
dspmqaut (Display authority)	240
dspmqcsv (Display command server)	244
dspmqfls (Display MQSeries file attributes)	245
dspmqtrc (Display MQSeries formatted trace output)	248
dspmqusr (Display MQSeries user information)	249
endmqcsv (End command server)	251
endmqm (End queue manager)	253
endmqtrc (End MQSeries trace)	255
instmqm (Install MQSeries for Tandem NSK)	256
runmqchi (Run channel initiator)	257
runmqchl (Run channel)	258
runmqdlq (Run dead-letter queue handler)	259
runmqlsr (Run listener)	260
runmqsc (Run MQSeries commands)	261
runmqtrm (Start trigger monitor)	264
setmqaut (Set/reset authority)	265
strmqcsv (Start command server)	271
strmqm (Start queue manager)	272
strmqtrc (Start MQSeries trace)	273
upgmqm (Upgrade V2.2 queue manager)	276

Part 3. Appendixes	277
Appendix A. MQSeries for Tandem NSK at a glance	279
Program and part number	279
Hardware requirements	279
Software requirements	279
Security	279
Maintenance functions	280
Compatibility	280
License management	280
Language selection	281
Message persistence	281
Internationalization	281
Appendix B. PAK file installation examples	283
An example PAK file installation (nonnative installation)	283
An example PAK file installation (native installation)	287
An example PAK file installation (UPGRADE installation)	290
Appendix C. System defaults	293
Appendix D. Stopping and removing queue managers manually	295
Stopping a queue manager manually	295
Removing queue managers manually	295
Appendix E. MQSC supported by MQSeries for Tandem NSK	297
Attributes of MQSC	298
CLEAR QLOCAL considerations	299
Appendix F. Application Programming Reference	301
Elementary data types – TAL programming language	301
Structure data types – TAL programming language	301
MQI calls – TAL programming language	307
Attributes of MQSeries objects	313
Reason codes	313
Appendix G. Building and running applications	315
Writing applications	315
Compiling and binding applications	318
Running applications	319
Appendix H. MQSeries for Tandem NSK sample programs	321
Building C sample programs	322
Building COBOL sample programs	323
Building TAL sample programs	324
Building and using data-conversion exit functions	324
Building and using channel exit functions	326
Appendix I. User exits	329
Channel exit programs	329
Data-conversion exit programs	330

Contents

Appendix J. Setting up communications	333
SNA channels	333
TCP/IP channels	334
Communications examples	335
Appendix K. MQSeries clients	349
Client support	349
Appendix L. Programmable System Management	351
Instrumentation events	351
Programmable command formats (PCFs)	354
Installable services	358
Appendix M. EMS event template used by MQSeries for Tandem NSK	359
Appendix N. Messages	363
Message format	363
Structure of messages	363
MQSeries messages	363
Appendix O. Notices	415
Trademarks	416
<hr/>	
Part 4. Glossary and index	417
Glossary of terms and abbreviations	419
Index	427

Figures

1.	Installing MQSeries for Tandem NSK	22
2.	Example PATHWAY configuration	51
3.	The MQSeries for Tandem NSK MQM Main Menu	78
4.	The Queue Manager Menu panel	79
5.	The QUEUE MANAGER TRACE MENU	80
6.	The Search Criteria panel (queue)	81
7.	The Queue Menu	82
8.	The Create Queue panel	83
9.	The Create Local Queue panel	84
10.	The Create Remote Queue panel	84
11.	The Copy Queue panel	85
12.	The Display/Modify Local Queue panel	86
13.	The Monitor Local Queues panel	87
14.	The Search Criteria panel (channel)	88
15.	The Channel Menu	88
16.	The Display/Modify Sender Channel panel (1)	89
17.	The Display/Modify Sender Channel panel (2)	90
18.	The Create Channel panel	90
19.	The Create Sender Channel panel	91
20.	The Create Receiver Channel panel	92
21.	The Create Server Connection Channel panel	92
22.	The Monitor Channels panel	93
23.	The Channel Status panel	94
24.	The Start/Stop Channel panel	94
25.	The Reset Channel panel	95
26.	The Resolve Channel panel	96
27.	The Copy Channel panel	97
28.	Example output for QMGR ALL	101
29.	Extract from the MQSC command file, mymqscin	104
30.	Extract from the MQSC report file, mymqscou	105
31.	Remote administration	127
32.	Setting up channels and queues for remote administration	128
33.	Example control data	159
34.	Example rule	160
35.	Example MQSeries configuration file (MQSINI)	184
36.	Example queue manager configuration file (QMINI)	188
37.	Sample trace	211
38.	Sample First Failure Symptom Report	212
39.	An example PAK file installation (nonnative installation)	283
40.	An example PAK file installation (native installation)	287
41.	An example PAK file installation (UPGRADE install)	290

Tables

1.	Commands for queue manager administration	45
2.	Commands for command server administration	46
3.	Commands for queue administration	46
4.	Commands for process administration	46
5.	Commands for channel administration	47
6.	Other control commands	47
7.	Security authorization needed for MQI calls	149
8.	MQSC commands and security authorization needed	151
9.	PCF commands and security authorization needed	152
10.	How to read syntax diagrams	219
11.	Security authorities from the dspmqaut command	241
12.	Specifying authorizations for different object types	268
13.	Objects included in amqscoma	293
14.	Using the correct version of the MQI library	318
15.	Event types supported by MQSeries for Tandem NSK	351
16.	MQMD structure of an event message	352
17.	Event header structure (MQCFH)	353
18.	PCF message descriptor	354
19.	PCF header	355
20.	PCF string parameter	355
21.	PCF integer list	356
22.	PCF integer	356
23.	PCF string list	356
24.	PCF commands supported by MQSeries for Tandem NSK	357

About this book

MQSeries for Tandem NonStop Kernel, Version 2 Release 2.0.1—referred to in this book as MQSeries for Tandem NSK V2.2.0.1 or MQSeries, as the context permits—is part of the MQSeries family of products. These products provide application programming services that enable application programs to communicate with each other using *message queues*. This form of communication is referred to as *commercial messaging*. The applications involved can exist on different nodes on a wide variety of machine and operating system types. They use a common application programming interface, called the Message Queuing Interface or MQI, so that programs developed on one platform can readily be transferred to another.

This book describes the system administration aspects of MQSeries for Tandem NSK, Version 2 Release 2.0.1, and the services it provides to support commercial messaging in a Tandem NSK environment. This includes managing the queues that applications use to receive their messages, and ensuring that applications have access to the queues that they require.

Who this book is for

Primarily, this book is for system administrators, and system programmers who manage the configuration and administration tasks for MQSeries. It is also useful to application programmers who must have some understanding of MQSeries administration tasks.

What you need to know to understand this book

To use this book, you should have a good understanding of the Tandem NSK operating system and associated utilities. You do not need to have worked with message queuing products before, but you should have an understanding of the basic concepts of message queuing.

How to use this book

The body of this book:

- Introduces MQSeries
- Describes how to install and configure MQSeries for Tandem NSK
- Describes day-to-day management of an MQSeries for Tandem NSK system, addressing topics such as administration of local and remote MQSeries objects, security, transactional support, and problem determination

Using the appendixes

The appendixes provide reference material. Some include information that will be incorporated in other MQSeries books at the next opportunity. Those appendixes are:

- Appendix E, “MQSC supported by MQSeries for Tandem NSK” on page 297.
- Appendix F, “Application Programming Reference” on page 301.
- Appendix G, “Building and running applications” on page 315.
- Appendix H, “MQSeries for Tandem NSK sample programs” on page 321.
- Appendix I, “User exits” on page 329.
- Appendix J, “Setting up communications” on page 333.
- Appendix K, “MQSeries clients” on page 349.
- Appendix L, “Programmable System Management” on page 351.

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 Digital UNIX V2.2.1
- 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 V2.2.0.1
- 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 xvii. For a functional comparison of the List A and List B 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 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 classes for Java are 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 publications

MQSeries for HP-UX

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

MQSeries for OS/2

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 2.0.1 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

- *SNAX/APC Planning and Configuration Manual*, (Tandem Part No. 098289)
SNAX/APC provides LU 6.2 support for the Tandem implementation of SNA. This guide explains how to install and configure SNAX/APC.
- *SCF Reference Manual for SNAX/APC*, (Tandem Part No. 064525)
SNAX/APC provides LU 6.2 support for the Tandem implementation of SNA. This guide explains the Subsystem Control Facility (SCF) interactive interface that lets operators and network managers configure and control SNAX/APC.
- *Pathway System Management Guide*, (Tandem Part No. 096881)
This guide presents guidelines for configuring and controlling Pathway transaction processing systems.
- *Introduction to NonStop Transaction Manager/MP (TM/MP)*, (Tandem Part No. 085812)
This guide describes how to use the TMF subsystem to protect your business transactions and the integrity of your databases.
- *Introduction to Tandem Networking and Data Communications*, (Tandem Part No. 093148)
This guide provides an overview of Tandem networking and data communications concepts, tasks, products, and manuals.
- *Intersystem Communications Environment (ICE) Installation Guide*, (Version 2 Release 2, December 1995, or later edition)
This guide describes how to install ICE and configure the ICE start-up parameters. (ICE provides LU 6.2 support for Insessions's implementation of SNA.)
- *Intersystem Communications Environment (ICE) Administrator's Guide*, (Version 2 Release 2, January 1996, or later edition)
This guide describes how to configure and operate ICE, its interfaces, and its utilities.

Information about MQSeries on the Internet

MQSeries URL

The URL of the MQSeries product family home page is:

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

Part 1. Guidance

Chapter 1. Introduction

This chapter introduces MQSeries for Tandem NSK, Version 2 Release 2.0.1 (MQSeries for Tandem NSK, V2.2.0.1) from an administrator's perspective. It describes the basic concepts of MQSeries and messaging. It contains these sections:

- "MQSeries and message queuing"
- "Messages and queues"
- "Objects" on page 5
- "MQSeries queues" on page 7
- "Process definitions" on page 10
- "Channels" on page 10
- "System default objects" on page 10
- "Administration" on page 10
- "Clients and servers" on page 11
- "Extending queue manager facilities" on page 12
- "Security" on page 12
- "MQSeries for Tandem NSK and related products" on page 13
- "Highlights of MQSeries for Tandem NSK V2.2.0.1" on page 15

MQSeries and message queuing

MQSeries for Tandem NSK, V2.2.0.1 lets Tandem NSK applications use message queuing to participate in message-driven processing. Your applications can communicate across different platforms by using MQSeries function.

MQSeries for Tandem NSK, V2.2.0.1 implements a common application programming interface called the *Message Queue Interface* (MQI). The MQI is explained in detail in the *MQSeries Application Programming Reference* manual.

Time-independent applications

With message queuing, the exchange of messages between a sending and receiving application is time independent. Time independence lets the sending application continue processing without having to wait for the receiving application to acknowledge the receipt of the message. The receiving application does not need to be running when the sending application sends the message. The message can be retrieved after the receiving application starts.

Message-driven processing

Applications can be automatically started by messages arriving on a queue using a mechanism called *triggering*. You can stop the applications when messages are processed.

Messages and queues

Messages and queues are the basic components of a message queuing system.

What are messages?

Data that is transferred by the MQSeries system is in the form of a message. A message is exchanged between cooperating distributed applications (or between different parts of the same application). The applications can be running on homogeneous platforms or on heterogeneous platforms.

MQSeries messages comprise two parts: the *application data* and a *message descriptor*. The content and structure of the application data are defined by the applications that use them. The message descriptor identifies the message and contains other control information, such as 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 Tandem NSK. For a complete description of the message descriptor, see the *MQSeries Application Programming Reference*

Message lengths

In MQSeries for Tandem NSK, the maximum message length is 4 MB (where, 1 MB equals 1 048 576 bytes). The message length can 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 either the sending or receiving application.
- The amount of storage available for the message.

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

What are queues?

Messages are exchanged between applications via queues, which use calls from the Message Queuing Interface (MQI). Queues belong to a *queue manager*. A queue manager puts received messages on the appropriate queues.

For example, an application can put a message on a queue, and another application can retrieve the message from the same queue. The sending application opens the queue for PUT operations by making an MQOPEN call, and then issues an MQPUT call to put the message onto that queue. The receiving application opens the same queue for GET, and retrieves the message from the queue by issuing an MQGET call.

Message queues are classified as local or remote. A local queue is any queue located on the same message queuing system. A local queue that is used to hold messages to be transmitted to another system is called a transmission queue. An alias queue is a logical naming capability that lets an alias queue name be resolved to another local or remote queue.

For more information about MQI calls, see the *MQSeries Application Programming Reference* manual.

Predefined and dynamic queues

You can create predefined and dynamic queues as follows:

Predefined queue

Use an MQSC command. For example, the MQSC command DEFINE QLOCAL lets you create a predefined local queue. A predefined local queue is permanent and exists independently of the applications using it. A predefined queue is not altered if you restart MQSeries for Tandem NSK.

Dynamic queue

Use an MQSC command. For example, the MQSC command DEFINE QMODEL. The attributes of a model queue (for example, the maximum number of messages that can be stored) are inherited by any dynamic queue that is created from a model queue. The queue created is based on a template queue definition, which is the model queue. A model queue has an attribute that specifies whether the dynamic queue is to be permanent or temporary. A permanent queue is not altered when you restart an application or the queue manager. A temporary queue can be lost or damaged upon restart.

For more information about MQSeries commands (MQSC), see the *MQSeries Command Reference* book.

Retrieving messages from queues

In MQSeries for Tandem NSK, authorized applications can retrieve messages from a queue using these methods:

- First-in-first-out (FIFO) is a queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time.
- Message priority, as defined in the message descriptor. Messages that have the same priority are retrieved on a FIFO basis.
- A program request for a specific message.

A combination of the MQGET, MQOPEN, and the queue object attributes determines the method used.

Objects

Many of the tasks described in this guide involve manipulating MQSeries *objects*. There are four types of object, which you can manipulate as follows:

- Queue manager; see “Queue manager” on page 6.
- Queues; see “MQSeries queues” on page 7.
- Process definition; see “Process definitions” on page 10.
- Channel; see “Channels” on page 10.

Object names

Each instance of a queue manager has an object name. This object name must be unique within the network of queue managers for proper identification of the target queue manager to which a message is sent.

The object name must be unique within a queue manager and object type. For example, you can have a queue and a process with the same name; however, you cannot have two queues with the same name.

Objects

An object name can have a maximum of 48 characters, with the exception of *channels*. Channel objects can have a maximum of 20 characters. For more information about names see “Using names” on page 218.

Managing objects

MQSeries provides facilities for creating, altering, displaying, and deleting objects. These include:

- MQSC commands (MQSC), which can be entered from the keyboard or read from a file
- MQM (screen-based interface)
- Programmable Command Format (PCF) commands, which a program can use.
- Control commands, which you can enter interactively from the operating-system command line.

For more information, see Chapter 4, “Using administration command sets” on page 43.

Object attributes

The properties of an object are defined by its object attributes. You can specify or change some object attributes, but only view others. For example, the maximum message length that a queue can accommodate is defined by its *MaxMsgLength* attribute. You can specify this object attribute when you create a queue. The *DefinitionType* attribute specifies how the queue was created. You can only display the *DefinitionType* attribute.

In MQSeries, there are two ways of referring to an object attribute:

- Using its PCF name, for example, *MaxMsgLength*. The PCF name is the formal name of an attribute.
- Using its MQSC name, for example, MAXMSGL. This book uses the MQSC name rather than the formal PCF name.

Queue manager

A queue manager provides message queuing services to applications. It ensures that:

- Object attributes are changed according to the commands received.
- Special events, such as trigger events or instrumentation events, are generated when the appropriate conditions are met.
- Messages are put on the correct queue, as requested by the application making the MQPUT call. The application is informed if this task is not accomplished, and you are provided with the appropriate reason code.

Each queue belongs to a single queue manager and is referred to as a *local queue* to that queue manager. The queue manager to which an application is connected is the local queue manager for that application. For the application, the queues that belong to its local queue manager are called local queues. A *remote queue* is a queue that belongs to another queue manager. A remote queue manager can exist on a remote system across the network or it can exist on the same system as the local queue manager. MQSeries for Tandem NSK supports multiple queue managers on the same system.

MQI calls

A queue manager object can be used for various MQI calls. For example, you can inquire about object attributes using the MQINQ MQI call.

Note: Messages are always put on queue objects, not on queue manager objects. You cannot put a message on a queue manager object.

MQSeries queues

Queues are defined to MQSeries for Tandem NSK using the MQSC DEFINE commands, the Message Queue Management (MQM) facility of MQSeries for Tandem NSK, or the PCF command Create Queue. These commands specify the type of queue and its object attributes. For example, a local queue has object attributes that specify when the applications reference that queue in MQI calls. Examples of object attributes 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 among 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

For further information:

- About MQSC, see the *MQSeries Command Reference* book
- About MQM, see “Using the Message Queue Management (MQM) facility” on page 78
- About PCF commands, see the *MQSeries Programmable System Management* book

Using queue objects

MQSeries for Tandem NSK has the following four types of queue object:

1. A *local queue object* is any queue that resides on the same message queuing system as the application.
2. A *remote queue object* is any queue residing on another message queuing system. This queue must be defined as a local queue to that queue manager. The information that you specify when you define a remote queue object lets the local queue manager find the remote queue manager, which allows any messages destined for the remote queue to go to the correct queue manager.

You must also define a transmission queue and channels between the queue managers, before applications can send messages to a queue on another 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 process lets you change the queues that an application uses without changing the application in any way. You change the alias queue definition to reflect the name of the new queue to which the alias resolves.

MQSeries queues

An alias queue is not a queue, but an object that you can use to access another queue.

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 issues an MQOPEN request specifying a queue name that is the name of a model queue. The dynamic queue that is created is a local queue whose attributes are taken from the model queue definition. The dynamic queue name can be specified by the application, or the queue manager can generate the name and return it to the application.

Dynamic queues defined in this way can be temporary queues, which can be lost or damaged by restarts, or permanent queues, which are not altered by restarts.

Local queues used by MQSeries

MQSeries uses various local queues for specific purposes related to its operation. You *must* define them before MQSeries can use them. You can create all default objects for a queue manager by running the supplied command file amqscoma, which is in subvolume \$SYSTEM.ZMQSSMPL by default.

Application queues

A queue that is used by an application (through the MQI) is referred to as an *application queue*. This queue can be a local queue on the queue manager to which an application is connected, or it can be a remote queue that is owned by another queue manager.

Applications can put messages on local or remote queues. However, they can get messages from a local queue only.

Initiation queues

Initiation queues are queues that are used in triggering. A queue manager puts a trigger message on an initiation queue when a trigger event occurs. A trigger event is a logical combination of conditions that is detected by a queue manager. For example, a trigger event can be generated when the number of messages on a queue reaches a predefined depth. This event causes the queue manager to put a trigger message on a specified initiation queue. This trigger message is retrieved by a *trigger monitor*, a special application that monitors an initiation queue. 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.

See "Managing objects for triggering" on page 120. For more information about triggering, see the *MQSeries Application Programming Guide*.

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.

These queues are also used in remote administration. See “Administering a remote queue manager” on page 126. For information about the use of transmission queues in distributed queuing, see the *MQSeries Intercommunication* book.

Dead-letter queues

A *dead-letter queue* stores messages that cannot be routed to their correct destinations. For example, this event occurs when the destination queue is full. The supplied dead-letter queue is called SYSTEM.DEAD.LETTER.QUEUE. These queues are also referred to as undelivered-message queues on other platforms.

For distributed queuing, you should define a dead-letter queue on each active queue manager.

Command queues

The command queue, named SYSTEM.ADMIN.COMMAND.QUEUE, is a local queue to which suitably authorized applications can send MQSeries for Tandem NSK commands for processing. These commands are then retrieved by an MQSeries component called the command server. The command server validates the commands, passes valid commands to the queue manager for processing, and returns any responses to the appropriate reply-to queue.

Reply-to queues

When an application sends a request message, the application that receives the message can send a reply message to the sending application. This message is put on a queue, called a reply-to queue, which is normally a local queue to the sending application. The name of the reply-to queue is specified by the sending application as part of the message descriptor.

Event queues

MQSeries for Tandem NSK supports instrumentation events, which can be used to monitor queue managers independently of MQI applications. Instrumentation events can be generated in several ways, for example:

- An application attempting to put a message on a queue that is not available or does not exist
- A queue becoming full
- A channel being started

When an instrumentation event occurs, the queue manager puts an event message on an event queue. This message can then be read by a monitoring application that can inform an administrator or initiate remedial action if the event indicates a problem.

Note: Trigger events are different from instrumentation events in that trigger events are not caused by the same conditions, and do not generate event messages.

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

Process definitions

A *process definition object* defines an application that is to be started in response to a trigger event on an MQSeries for Tandem NSK queue manager. See “Initiation queues” on page 8 for more information.

The process definition attributes include the application ID, the application type, and data specific to the application.

Use the MQSC command DEFINE PROCESS or the PCF command Create Process to create a process definition.

Channels

Channels are objects that provide a communication path from one queue manager to another. Channels are used in distributed message queuing to move messages from one queue manager to another. Channels shield applications from the underlying communications protocols. The queue managers can exist on the same or different platforms. For queue managers to communicate with one another, you must define one channel object at the queue manager that is to send messages, and another channel object at the queue manager that is to receive them.

For information on channels and how to use them, see the *MQSeries Intercommunication* book, and also “Preparing channels and transmission queues for remote administration” on page 127.

System default objects

The *system default objects* are a set of object definitions that you can create for each queue manager, using the command file amqscoma, which is supplied with MQSeries. You can copy and modify any of these object definitions for use in applications at your installation. Default object names have the stem SYSTEM.DEF; for example, the default local queue is SYSTEM.DEFAULT.LOCAL.QUEUE; the default receiver channel is SYSTEM.DEF.RECEIVER. You cannot rename these objects; default objects of these names are required.

When you define an object, any attributes that you do not specify explicitly are copied from the appropriate default object. For example, if you define a local queue, the attributes you do not specify are taken from the default queue SYSTEM.DEFAULT.LOCAL.QUEUE.

Administration

In MQSeries, you execute administration tasks by issuing *commands*. Four command sets are provided. Which set you use depends on the tasks you want to perform and how you want to perform them. The command sets are described in Chapter 4, “Using administration command sets” on page 43. Administration tasks include:

- Starting and stopping queue managers.
- Creating objects, particularly queues, for applications.

- Working with channels to create communication paths to queue managers on other (remote) systems. This process is explained in detail in the *MQSeries Intercommunication* book.

Local and remote administration

Local administration entails executing administration tasks on a queue manager you have defined on your local system. In MQSeries, this process is known as local administration because no channels are involved, that is, the communication is managed by the operating system.

MQSeries supports administration from a single point using *remote administration*. This process lets you issue commands from your local system, which are processed on another system. You do not have to log on to that system; however, you need to have the appropriate channels defined. The queue manager and command server on the target system must be running. For example, you can issue a remote command to change a queue definition on a remote queue manager.

Various commands cannot be issued in this way, in particular, creating or starting queue managers and starting command servers. To perform this type of task, you must either log on to the remote system and then issue commands, or create a process that can issue the commands for you.

Clients and servers

MQSeries for Tandem NSK supports client-server configurations for MQI applications. There are no MQSeries for Tandem NSK clients, only an MQSeries for Tandem NSK server; however, clients on other platforms can connect to the MQSeries for Tandem NSK server.

An *MQI client* is part of the MQSeries product that is installed on a machine to accept MQI calls from applications and pass them to an MQI server machine. There they are processed by a queue manager. Typically, the client and server reside on different machines but they can also exist on the same machine.

An *MQI server* is a queue manager that provides queuing services to one or more clients. All the MQSeries objects (for example, queues) exist only on the queue manager system, that is, on the MQI server machine. A server can support normal, local MQI applications as well.

For more information, see the *MQSeries Intercommunication* book and the *MQSeries Clients* book.

MQI applications in a client-server environment

When linked to a server, MQI client applications can issue MQI calls in the same way as local applications. The client application issues the MQCONN call to connect to a specified queue manager. Any additional MQI calls that specify the connection handle returned from the connect request are then processed by this queue manager. You must link your applications to the appropriate client libraries. See the *MQSeries Application Programming Guide* for further information. No MQI client is currently provided for Tandem NSK; however, since Tandem NSK is an MQI Server, it accepts connections from any MQSeries MQI client running on other platforms.

Extending queue manager facilities

The facilities provided by a queue manager can be extended by:

- User exits
- Installable services

User exits

User exits let you insert your own programming code into a queue manager function. Two types of user exit are supported:

- *Channel exits*, which change the way that channels operate.
- *Data conversion exits*, which can be used by application programs to convert data from one format to another.

For more information about these exits, see Appendix I, “User exits” on page 329.

Installable services

Installable services are more extensive than user exits in that they have a formalized Application Programming Interface (API) with multiple entry points.

An implementation of an installable service is called a *service component*. You can use the components supplied with the product, or you can write your own component to perform the functions that you require. Currently, the following installable services are provided:

The *authorization service*, which lets you build your own security facility. The default service component that implements the service is the Object Authority Manager (OAM), which is supplied with the product. By default, the OAM is enabled. You can use the authorization service interface to create other components to replace or augment the OAM.

The *name service*, which allows queue managers to share queues. You must write your own component to carry out this task, which enables a queue manager to determine the owner of a queue.

See the *MQSeries Programmable System Management* manual for more information.

Security

Authorization for using MQI calls and commands and for accessing objects is provided by the Object Authority Manager (OAM), which by default is enabled. Access to MQSeries entities is controlled through MQSeries for Tandem NSK principals and user groups, and the OAM. The principal and group names that the OAM supports are resolved to Tandem NSK user and group names. In V2.2.0.1, all users of MQSeries must have a principal name that maps to a Tandem NSK user name. This is required regardless of whether the OAM is enabled. A command-line interface is provided to allow you to add and delete principals, and to grant and revoke authorizations, as required.

In addition, NSK security facilities can be used to control access to MQSeries commands and database files. If SAFEGUARD is installed, MQSeries is compatible with, and can take advantage of, some of the extended facilities that it provides. For more information, see Chapter 8, “Implementing security control” on page 137.

MQSeries for Tandem NSK and related products

The following products are required for the operation of MQSeries for Tandem NSK:

- NonStop TM/MP (TMF)
- NonStop TS/MP (PATHWAY)
- SNA LU 6.2 or TCP/IP
- ENSCRIBE
- EMS

EMS, ENSCRIBE, NonStop TM/MP (TMF), and NonStop TS/MP (PATHWAY) are included with the Tandem NSK operating system.

For SNA LU 6.2 function, either SNAX/APC or Insession ICE can be used.

Migration from MQSeries for Tandem NSK V1.5.1

If you are a user of MQSeries for Tandem NSK V1.5.1, you can convert your existing MQSeries configuration files and messages to work with MQSeries for Tandem NSK V2.2.0.1 using the following two conversion utilities:

CNV1520 Converts MQSeries for Tandem NSK Version 1.5.1 queue and channel definitions into MQSC scripts.

CNVMSG Transfers messages from MQSeries for Tandem NSK Version 1.5.1 message queues to Version 2.2.0.1 message queues after the queue definitions have been established using CNV1520.

Both utilities reside in the ZMQSEX subvolume.

Migrating applications

To migrate your MQSeries for Tandem NSK V1.5.1 applications, you must recompile and rebind them with V2.2.0.1 header files and libraries. Stubs have been provided for MQI calls that are not present or required in MQSeries for Tandem NSK V2.2.0.1, so code changes relating to MQSeries (other than including the correct header files) are not required. However, MQSeries for Tandem NSK V2.2.0.1 requires that you compile C programs with the WIDE model. MQSeries for Tandem NSK V1.5.1 required LARGE: if your programs contain code that relies on LARGE data representation, the code may have to be changed before it functions correctly under the WIDE model.

Migration from MQSeries for Tandem NSK V2.2

To migrate from V2.2 to V2.2.0.1 of MQSeries for Tandem NSK:

- Select the UPGRADE installation option when you install V2.2.0.1. Installation of MQSeries for Tandem NSK V2.2.0.1 is described in Chapter 2, “Installing MQSeries for Tandem NSK Version 2.2.0.1” on page 19.

Installation prerequisites

- When installation is complete, upgrade individual queue managers using the **upgmqm** command. For more information about **upgmqm**, see “Upgrading a Version 2.2 queue manager” on page 34 and “upgmqm (Upgrade V2.2 queue manager)” on page 276.

Migrating applications

To migrate MQSeries for Tandem NSK V2.2 applications to V2.2.0.1, recompile and rebind them with V2.2.0.1 header files and libraries.

Software requirements

Minimum software requirements are:

- Tandem NSK operating system version D3x, D4x, G02, or later G0x with TMF and PATHWAY
- Either:
 - TCP/IP, installed and correctly configured
 - or
 - SNAX/APC (over SNAX/XF or SNAX/APN), or Insession ICE, as appropriate, installed and correctly configured
- At least one of the following language compilers, installed and correctly configured: C, COBOL-85, or TAL

Hardware requirements

Specific hardware in support of user-selected network transport protocols must be available.

You are also recommended to have mirrored data disks with specified space requirements for TMF audit space as well as the MQSeries database.

Compilation and binding issues

MQSeries for Tandem NSK, V2.2.0.1 is built using the Common Runtime Environment (CRE) to link all objects. This method imposes the following requirements on users of versions of the MQI prior to Version 2.2.0.1:

1. All pre-D30 COBOL and C object code must be recompiled with the D30 (or later) compiler to integrate the CRE linkage.
2. All pre-D30 TAL object code must be recompiled with a D30 (or later) compiler and you must ensure that the TAL program is compliant with the special programming considerations specified in the *Common Run-time Environment Programmer's Guide*. More detailed information on each of these programming considerations is provided in the *TAL Programmer's Guide*.
3. For object code produced with native compilers on D40, a separate binding is provided.
4. C programs must use the WIDE memory model (32-bit integers).
5. COBOL programs must conform to the requirements of the CRE.
6. In TAL programs, all integers passed to the MQI functions must be 32 bits (or be cast to 32 bit with the \$INT32() macro).

Highlights of MQSeries for Tandem NSK V2.2.0.1

MQSeries for Tandem NSK V2.2.0.1 provides:

- An MQSS Server
- A native version of MQSeries for the D4x and G0x environments
- RDF compatibility enhancements
- Full MQSeries principal support

These new functions are introduced in the remainder of this chapter.

MQSeries Status Server (MQSS Server)

The MQSeries Status Server (MQSS Server) provides a valuable performance and scalability enhancement over MQSeries for Tandem NSK Version 2 Release 2. The MQSS Server maintains the status of all queue and process objects configured for a queue manager in memory, using a Tandem NonStop process pair architecture. Releases of MQSeries for Tandem NSK prior to V2.2.0.1 maintained the status data in a disk file, which required update for every MQSeries operation.

The status data for an MQSeries object includes not only the CURDEPTH, IPPROCS, and OPPROCS attributes, but also internal data used for controlling and coordinating concurrent MQSeries user and administration operations for that object. The MQSS Server provides reliable access to the data for the queue manager, with the performance benefits of using memory and interprocess messaging rather than TM/MP-protected disk file updates.

The default MQSS Server

Every queue manager has a default MQSS Server, which is configured when the queue manager is created. A new, mandatory, command-line parameter to the **crtmqm** command is required to specify the process name for the default MQSS Server. The process name must be unique across the system. The **crtmqm** command adds a new TS/MP server class definition for the default MQSS Server into the queue manager's PATHWAY. Whenever a queue or process object is defined, it is initially configured to use the default MQSS Server for maintenance of status data.

Defining additional MQSS Servers

Additional MQSS Servers may be configured after queue manager creation. Objects that are heavily accessed may be assigned to dedicated MQSS Servers, rather than to the default MQSS Server. This technique provides improved scalability, which is required for large or high-capacity queue managers. Any queue or process object may be assigned to any MQSS Server defined in the queue manager. An MQSS Server is responsible for one or many objects. Therefore, any distribution of objects among the configured MQSS Servers in a queue manager may be accomplished. As with any TS/MP server class, MQSS Servers may be distributed across the system, specifying any valid combination of primary and backup CPUs.

Functions of the MQSS Server

- The MQSS Server corrects the queue depth (CURDEPTH) attribute of local queues, a function performed by the **cleanqm** utility in Version 2 Release 2. You can specify when the queue depth is to be recalculated on a queue-by-queue basis. This provides a more robust, and much more configurable facility. The trade-off between CURDEPTH accuracy and the resource overhead of recalculation can be accurately tuned, depending on the usage of each queue.
- The MQSS Server is also responsible for maintaining an accurate count of the number and type of MQSeries opens against each object for which it is responsible. On a queue-by-queue basis, the MQSS Server can be configured to update the Object Catalog database with the current status data, either when specific events occur or periodically.

MQSeries imposes no restriction on the number of MQSS Servers that may be configured for a queue manager, or on the number of objects that may be configured to use a specific MQSS Server.

- As a further performance enhancement for message throughput, the special queue indices that support retrieval of messages by *Msgld* and *Correlld* can be removed. If an application never needs to use these retrieval mechanisms (that is, it needs to use only the FIFO or priority-based FIFO mechanisms) the indices may be removed, giving a significant increase in performance for MQPUT and MQGET operations. These options may still be configured on a queue-by-queue basis. See “Specifying retrieval options for queues” on page 40 for more information.

Native version of MQSeries (D4x and G0x)

A native version of MQSeries is provided for Tandem NSK D4x and G0x. This has been built using the Tandem native compilers and installs MQSeries as a private SRL. As a result, the footprint of the MQSeries executables is much reduced, and the performance improvements of native applications are gained.

In all functional respects, the behavior of the native version is identical to that of the nonnative version and, except where explicit reference is made, the documentation for the two versions is identical.

Areas of difference are:

1. Building of samples and other MQI applications
2. Inclusion of exits and pluggable services.

RDF compatibility enhancements

MQSeries has been enhanced to be more easily integrated with the Remote Database Duplication Facility (RDF) environment.

In versions of MQSeries for Tandem NSK prior to V2.2.0.1, audited files were purged (when a queue was deleted, for example) and, although the files were removed from the primary system, RDF did not remove them from the backup system. This caused problems for RDF if the removed files were recreated on the primary system (if a queue that was deleted was recreated, for example), but had not been removed from the backup system: RDF experienced file-system errors when trying to recreate the files on the backup system.

In addition, several databases in the queue manager were not audited in releases of MQSeries for Tandem NSK prior to V2.2.0.1. RDF does not address nonaudited files, making the maintenance of an up-to-date configuration on the backup system difficult.

MQSeries for Tandem NSK V2.2.0.1 now recognizes a new environment variable (PARAM) that indicates that it is operating in an RDF environment. If the environment variable is set, the queue manager no longer purges audited files as the objects that need them are deleted from MQSeries. Instead, the files are emptied, any references to them are removed from other databases, and their disk space is deallocated. An entry is made in a new database that lists the names of the files that have been logically deleted. This new database file is processed by a new utility, invoked by the **cleanrdf** control command, which performs the clean-up of the logically deleted files on both the primary and backup sites.

The **cleanrdf** command runs from the command line or as a batch job scheduled via Tandem's NetBatch product. An additional feature of **cleanrdf** is that it copies the contents of all other nonaudited databases (there are still a few remaining) to the backup site to ensure that each time **cleanrdf** is run, there are up-to-date versions of all databases (audited and nonaudited) on the backup site.

Finally, all databases that are used in normal operation and that are not static have been made audited in V2.2.0.1.

Integration with RDF can be achieved only if the backup is configured to have a one-to-one mapping of volume names with the primary site. That is, the backup-site volume names must be exactly the same as the primary site's volume names. This configuration is highly recommended by Tandem.

The **dltmqm** operation cannot be supported by integration with RDF. Deleting queue managers remains a manual operation on the backup site to be performed by operations staff.

Full MQSeries principal support

In MQSeries for Tandem NSK V2.2, the object authority manager (OAM) based all its authorizations on the group to which a particular user ID belonged. This group name was also used as the MQSeries principal, which flows to remote platforms for security checking.

MQSeries for Tandem NSK V2.2.0.1 introduces the MQSeries principal to the Tandem environment, facilitating full interoperability with other MQSeries platforms. A principal database provides mappings between MQSeries principals and Tandem user IDs. You manage the contents of the principal database using two new control commands, **altmqusr** and **dspmqusr**, which are described in Chapter 15, "The MQSeries control commands" on page 217.

Note: Even queue managers with the OAM disabled require entries in the principal database for users who require access to the queue manager.

V2.2.0.1 highlights

Chapter 2. Installing MQSeries for Tandem NSK Version 2.2.0.1

This chapter explains how to install MQSeries for Tandem NSK V2.2.0.1 and how to verify that your installation is successful. It contains the following topics:

- “Preparing for installation” on page 20
- “Installation procedure” on page 21
- “TACL environment variables” on page 26
- “Configuration of other NonStop Kernel resources” on page 27
- “Files that exist after installation (native and nonnative)” on page 27
- “Verifying your installation” on page 32
- “Troubleshooting” on page 34
- “Upgrading a Version 2.2 queue manager” on page 34

The installation of the product consists of the following steps:

1. Preparing and planning
2. Loading the installation utility software from tape
3. Running the installation utility
4. Verifying the installation

The default installation subvolumes (or ISVs — where the software is initially loaded from tape) are:

<code>\$\$SYSTEM.ZMQSCONV</code>	Data conversion tables
<code>\$\$SYSTEM.ZMQSEXE</code>	Product executables
<code>\$\$SYSTEM.ZMQSLIB</code>	Libraries and header files
<code>\$\$SYSTEM.ZMQSSMPL</code>	Sample code
<code>\$\$SYSTEM.ZMQSSYS</code>	Product configuration files

Preparing for installation

This section guides you through some of the steps you must perform before you install MQSeries for Tandem NSK V2.2.0.1.

Before you can install MQSeries for Tandem NSK V2.2.0.1 you must:

- Create the user ID MQM.MANAGER (in the MQM user group) to use for the installation. The user ID MQM.MANAGER is the user ID:
 - Under which all queue managers are created and run
 - Under which all product executables (rather than applications) are run
 - By which all product data files and databases are owned
- Decide which national language to use for the installation from the supported national languages. The national language is determined on a system-wide basis at the time of installation. All queue managers on a system use the same national language.
- Decide whether you are migrating from MQSeries for Tandem NSK V2.2 or are installing MQSeries for Tandem NSK V2.2.0.1 from scratch.
- Decide whether you want to install and use native-mode executables for the queue manager, if you are running NSK D4x or G0x. Native-mode executables use fewer resources than nonnative, resulting in better performance in most environments.
- Determine the location of the installation subvolumes (ISVs) if different from the default location (\$SYSTEM).
- Verify that the disk space available on the installation volume is sufficient.
- If you are migrating to MQSeries for Tandem NSK V2.2.0.1 from an earlier release, ensure that all queue managers are stopped and back up all MQSeries files.

Disk storage

These are the approximate storage requirements:

- Base code and runtime: 170 MB (native installation)
- Base code and runtime: 100 MB (nonnative installation)

For information about physical file size of MQSeries message queues, see “Default physical file size for queues” on page 109.

TMF audit trail

For each queue manager and for each MQSeries application that uses a queue manager, there needs to be an allowance for the space used in the TMF audit trail volume. For more information, see Chapter 11, “Understanding transactional support and messaging” on page 175.

Installation procedure

To install MQSeries for Tandem NSKV2.2.0.1:

1. Logon as user MQM.MANAGER.
2. Run the RESTORE command to restore the installation utility from tape into the installation subvolume. For example:

```
RESTORE <tape device>, $*.install.instmqm, MAP NAMES
$*.*.* to $vol.subvol.*, NOUNLOAD, LISTALL, MYID
```

where \$vol.subvol is the volume and subvolume where you want to restore and use the **instmqm** utility.

After the RESTORE command is complete, verify that there are no errors and that **instmqm** is correctly restored.

3. Run the installation utility by entering **instmqm** at the TACL prompt. (For a description of the **instmqm** command, see “instmqm (Install MQSeries for Tandem NSK)” on page 256.) The installation utility loads the remaining software from tape. For Tandem NSK D4x and G0x, the native version of MQSeries may be installed instead of the nonnative version.

On tape, the files are structured as follows:

\$*.ZRELCONV.*	Data conversion tables
\$*.ZRELSYS.*	Common product configuration files
\$*.INSTALL.*	Product installation files
\$*.ZD30*.*	D3x-specific files (nonnative)
\$*.ZD40*.*	D4x- and G0x-specific files (native)

Figure 1 on page 22 shows the sequence of prompts (with example responses) that appear during the installation process, beginning with the RESTORE command. For a PAK file installation example, see Appendix B, “PAK file installation examples” on page 283. For an UPGRADE installation example, see Figure 41 on page 290.

Installation procedure

```
| $DEV2 INSTALL 447> restore $tape, $*.install.instmqm, map names $*.*.* to &
| $DEV2 INSTALL 447> &$dev2.install.*,nounload,listall,myid
| File Mode RESTORE Program - T9074ACR (26MAY95)
| Copyright Tandem Computers Incorporated 1981-1994
| Drives: ($TAPE)
| System: \HAWK Operating System: G02 Tape Version: 3
| Backup options: NO AUDITED, BLOCKSIZE 28, NO IGNORE, OPEN, PARTONLY OFF,
| INDEXES IMPLICIT
| Restore time: 15Dec98 14:42 Backup time: 9Dec98 18:47 Page: 1

| Tape: 1 Code EOF Last modif Owner RWEP Type Rec Block
|
| $DEV2.INSTALL
| INSTMQM 100 425984 4Dec98 18:08 44,1 NNNN

| Summary Information
|
| Files restored = 1 Files not restored = 0

| $DEV2 INSTALL 266> instmqm

| IBM MQSeries for Tandem NSK, Version 2.2.0.1
| Installation and License update program.

| @(#) Licensed Materials - Property of IBM 83H8731,5697-A17 (C) Copyright IBM Co
| rp. 1993, 1997 All Rights Reserved US Government Users Restricted Rights - Use,
| duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

| Product installation selected...
| You may stop the installation by entering
| "quit" at any prompt.
| Where there is a default for a parameter, you may
| select it by pressing the Enter key on its own.

| Phase 1: Collection of license information

| License information
| Enter the system type that you are licensed for.
| The following system types are recognized:
| CLX/R
| CLX800
| K100
| K200
| CYCLONE
| K1000
| K2000
| S7000
| K10000
| K20000
| S70000
| There is no default value for this parameter.
```

| *Figure 1 (Part 1 of 4). Installing MQSeries for Tandem NSK*

```
| Please enter your selection:  K2000

| Enter the number of CPUs that you are licensed for.
| The valid range for this parameter is 2 to 16.
| There is no default value for this parameter.

| Please enter the number:      4

| Will you be installing from tape or from an archive file?
| Enter TAPE or ARCHIVE.
| The default value for this parameter is "TAPE"

| Please enter the selection:

| Enter either a local or remote tape device name.
| The device name entered will be validated by opening it.
| If the device cannot be opened you will be given the
| opportunity to correct the name.
| There is no default value for this parameter.

| Please enter the tape device:  $TAPE

| Enter the name of your spooler process.
| The name entered will be validated by opening it.
| If the spooler cannot be opened you will be given the
| opportunity to correct the name.
| The default value for this parameter is "$S"

| Please enter the spooler name:

| Select the type of installation to be performed.
| The following options are available:
| SCRATCH      - a from scratch installation
| UPGRADE      - an upgrade from the latest service
|                level of MQSeries V2.2.0.0
| The default value for this parameter is "SCRATCH"

| Please enter the type of installation:

| Enter the volume that you will use for installation.
| Enter the volume name in the format "$VVVVVVV".
| The default value for this parameter is "$SYSTEM"

| Please enter the volume:      $DEV2

| Enter the default volume that you want Queue Managers to
| be created on.
| Note that the default Queue Manager volume may be changed
| at any time after installation by editing the MQSINI file.
| Enter the volume name in the format "$VVVVVVV".
| The default value for this parameter is "$DEV2"
```

| *Figure 1 (Part 2 of 4). Installing MQSeries for Tandem NSK*

Installation procedure

```
| Please enter the volume:
|
| Enter the subvolume on $DEV2 that you will use for executables.
| Enter the subvolume name in the format "VVVVVVVV".
| The default value for this parameter is "ZMQSEXE"
|
| Please enter the subvolume:
|
| Select the language to be used for administration messages.
| The following languages are available:
| ENUS    - US English
| ESES    - Spanish
| The default value for this parameter is "ENUS"
|
| Please enter the language:
|
| You are installing on D3x-series NSK
| You do not have the option of installing native mode executables.
| License verified.
|
| You have selected the following parameters for installation:
| Tape device for installation:      $TAPE
| Spooler name:                     $S
| Volume for installation:          $DEV2
| Default Queue Manager volume:     $DEV2
| Subvolume for executables:        ZMQSEXE
| Language for messages:            ENUS
| You are installing accelerated (TNS) executables.
| This is not an upgrade to a prior V2.2.0.0 installation.
| Beginning to restore files to $DEV2.
| Verify that the installation media is present and online
| in device $TAPE. Enter "YES" when ready.
| Ready to restore? (yes or quit):   yes
|
| Restoring product to $DEV2...
|
| Summary Information
|
| Files restored = 1274  Files not restored = 0
```

| *Figure 1 (Part 3 of 4). Installing MQSeries for Tandem NSK*

```
| Summary Information
|
| Files restored = 12  Files not restored = 0
|
| Summary Information
|
| Files restored = 2  Files not restored = 0
| Finished restoring files.
| If the summary information indicates a potential error,
| review the spooler jobs named #instmqm, and if necessary, repeat
| the installation.
| Securing files...
| Finished securing files.
| Creating MQSINI file...
| Finished creating MQSINI file.
| Creating message file...
| Finished creating message file.
| Installation complete.
```

| *Figure 1 (Part 4 of 4). Installing MQSeries for Tandem NSK*

Notes:

1. As shown in Figure 1 on page 22, the default installation volume is \$SYSTEM. That value (\$SYSTEM) is used in examples throughout this book. You should replace the value \$SYSTEM with the identifier of your own installation volume wherever appropriate.
2. After a fresh install of MQSeries or a cold load of the Tandem NSK system, MQSeries executables might take longer to run than expected when they are first invoked. This is because the Tandem NSK operating system goes through a “fixup” phase, during which it ensures that all external declarations are resolved.
3. Installation of MQSeries for Tandem NSK V2.2.0.1 can take significantly longer than installation of MQSeries for Tandem NSK V2.2, because there are three separate tape RESTORE operations. Please be sure to wait until **instmqm** has completed. If you interrupt **instmqm**, the results are unpredictable.

TACL environment variables

The following environment variables, or PARAMs, are recognized by MQSeries for Tandem NSK.

MQDEFAULTPREFIX

The name of the volume containing the installed subvolume, ZMQSSYS. This PARAM must be correctly defined in all environments.

For example:

```
PARAM MQDEFAULTPREFIX $data00
```

MQEMSEVENTS

This PARAM enables MQSeries EMS events. For example, to switch on all EMS events for MQSeries, you set the PARAM MQEMSEVENTS as follows:

```
PARAM MQEMSEVENTS 127
```

For more information about MQEMSEVENTS, see “Event Management Service (EMS) events” on page 171.

MQMACHINIFILE

The location of the MQSINI file for the installation. The default value is *MQDEFAULTPREFIX.ZMQSSYS.MQSINI*. This PARAM is required only if a nondefault location is required.

For example:

```
PARAM MQMACHINIFILE $data00.altinst.mqsini
```

MQRDF

If this PARAM is set ON, MQSeries changes the behavior of the delete operation to work with RDF for audited files. If this PARAM is not defined, or is set to anything other than ON, the MQSeries delete operation functions normally. If used, this PARAM must be set in the TACL environment of any user that runs administrative programs, and in the environment of all TS/MP server classes configured in the queue manager.

MQRDFFUPPROCESSNAME

This PARAM is interpreted only by the **cleanrdf** utility. It is used to specify a Guardian process name that will be assigned to the FUP server process that the **cleanrdf** utility creates. If this PARAM is not defined, the FUP server process name is assigned by the operating system.

MQRDFFUPPROGNAME

This PARAM is interpreted only by the **cleanrdf** utility. It is used to specify the fully qualified name of the FUP executable file to be used by the utility. The default value is <default system name>.\$SYSTEM.SYS00.FUP.

MQSNOAUT

If this PARAM is set to 1 when **crtmqm** is run, the new queue manager is created with the OAM disabled.

For example:

```
PARAM MQSNOAUT 1
```

SAVE-ENVIRONMENT ON

Required when running application programs to ensure the Common Run-Time Environment (CRE) passes PARAMs from the environment to the application program.

For example:

```
PARAM SAVE-ENVIRONMENT ON
```

If this PARAM is not set, applications receive return code 2058, indicating a queue manager name error.

The MQDEFAULTPREFIX PARAM must be present in the environment of all programs. The TS/MP (Pathway) configuration established automatically by the **crtmqm** command ensures that these PARAMs are set correctly for any queue manager server processes. Users of MQSeries applications and control commands must ensure that the ACLs and TS/MP configurations used also specify these variables.

You are recommended to include the PARAM statements in your TACLSTM files so that, when you log on, these PARAMs are created correctly, and any programs run from the TACL inherit the correct values. The following environment variable should also be modified to allow location of MQSeries executables:

```
#SET #PMSEARCH $SYSTEM.ZMQSEXE [#PMSEARCH]
```

Configuration of other NonStop Kernel resources

TM/MP (TMF)

The configuration of the TM/MP (TMF) product is critical to the correct operation of MQSeries queue managers. All volumes that hold queue manager data files must be TMF audited. The TMF subsystem must be configured with sufficient audit-trail space for the operation of all queue managers covered by that audit trail.

TS/MP (PATHWAY)

The Transaction Services TS/MP (PATHWAY) product must be available. Each MQSeries queue manager runs under its own PATHWAY configuration.

Communications

SNAX, ICE, or TCP/IP must be installed and configured appropriately to use MQSeries queue managers to communicate between systems (known as distributed queuing).

RDF (Remote Database Duplication Facility)

If the queue manager is to be used in an RDF environment, RDF must be configured to operate on all volumes used for the queue manager. Take particular care if queues have been moved to alternative volumes. RDF must also be configured with identically named volumes on both the primary and backup sites.

Files that exist after installation (native and nonnative)

After loading from tape, the following files exist in the ISVs:

Product executables (ZMQSEXE)

ALTMQFLS	Alter queue file attributes.
ALTMQUSR	Alter or add MQSeries user.
CLEANRDF	RDF housekeeping utility.
CNV1520	Convert V1.5.1 queues and channels for use with V2.2.0.1.
CNVCLCHL	Convert client channel definition table.
CNVMSGs	Convert V1.5.1 messages for use with V2.2.0.1.

CRTMQCVX	Data conversion utility.
CRTMQM	Create queue manager command.
DLTMQM	Delete queue manager command.
DSPMQAUT	Display MQSeries authorities command.
DSPMQCSV	Display command server.
DSPMQFLS	Display MQSeries files.
DSPMQTRC	Display trace.
DSPMQUSR	Display information about MQSeries users.
ENDMQCSV	End command server.
ENDMQM	Stop queue manager command.
ENDMQTRC	End trace.
INSTMQM	Install MQSeries for Tandem NSK.
MQCMDSVR	Command server.
MQEC	EC.
MQECBOSS	Execution Controller (EC) boss.
MLQMAG	Local queue manager agent (LQMA).
MLU6RES	MCA SNA Responder.
MQMCACAL	Message channel agent (MCA) caller.
MQSSSVR	MQSS server
MQMSVR	MQM server.
MQTCPRES	MCA TCP/IP Responder.
POBJCOD	MQM screen requestor code.
POBJDIR	MQM screen requestor directory file.
RUNMQCHI	Run channel initiator.
RUNMQCHL	Start an MQSeries channel.
RUNMQDLQ	Dead-letter queue (DLQ) handler.
RUNMQLSR	Run a TCP/IP listener.
RUNMQSC	MQSC command processor.
RUNMQTRM	Run trigger monitor.
SETMQAUT	Set MQSeries authorities command.
STRMQCSV	Start command server.
STRMQM	Start queue manager command.
STRMQTRC	Start trace.
UPGMQM	Upgrade a V2.2 queue manager.

Product configuration files (ZMQSSYS)

MQSINI	Machine-wide initialization database.
SMQSTMPL	EMS template file.
ZMQSC	EMS event template file - C.
ZMQSCOB	EMS event template file - COBOL.
ZMQSDDL	EMS event template file - Data Definition Language schema.
ZMQSPAS	EMS event template file - Pascal.
ZMQSTACL	EMS event template file - TACL.
ZMQSTAL	EMS event template file - TAL.
ZMQSTMPL	EMS event template file - Generic template.
MEMOPTF	Service information.
MSG	Installed copy of message file in selected language.
MSGENUS	American English message file.
MSGESES	Spanish message file.
CCSID	CCSID file.
UCCSID	Internal form of the CCSID file.
MQERRLG1	Early error log.
MLMD	License file.
MQSYSLOG	Global index for FFSTs.

AMQKAPPD	This book (GC33-1893-01) in Adobe Acrobat Portable Document Format (PDF).
AMQKAPPS	This book (GC33-1893-01) in PostScript format.
README	Late-breaking product information.

Libraries and header files (ZMQSLIB)

MQMLIBC (not native)	C MQI library.
MQMLIBT (not native)	TAL and COBOL MQI library.
MQMLIBNT (native only)	Static TAL and COBOL MQI link library
MQMLIBNC (native only)	Static C MQI link library.
MQSRLLIB (native only)	The executable MQSeries private SRL.
MQSRLLINK (native only)	The relinkable MQSeries private SRL.
NMLDEXES (native only)	TACL macro to relink the MQSeries executables with a new MQSeries private SRL.
CMQCH	Standard C header file for MQI interface.
CMQCFCH	MQCFH, MQCXP structures.
CMQXCH	MQCD structure.
CMQZCH	Installable services structure.
CMQXCDH	MQXDX structure.
CMQCTAL	TAL header file for the MQI.
CMQCOBOL	COBOL copy file for the MQI.
MQSVMHTH	Sample data conversion macro for use with crtmqcvx.

Sample code (ZMQSSMPL)

Miscellaneous samples

AMQSCOMA	MQSC command file for creating default objects.
AMQSCOS0	MQSC command file for creating sample objects.
AMQTRC	Sample MQSeries trace format file.
PARTIT	Sample TACL macro to partition queue files across disk volumes.
README	File containing late-breaking product news.
MEMOPTF	File containing information about service levels and maintenance.

Sample programs (COBOL)

AMQ0ECH
 AMQ0ECHO
 AMQ0GBR
 AMQ0GBR0
 AMQ0GET

ISV contents

	AMQ0GET0
	AMQ0INQ
	AMQ0INQ0
	AMQ0PUT
	AMQ0PUT0
	AMQ0REQ
	AMQ0REQ0
	AMQ0SET
	AMQ0SET0
	AMQREAD
	AMQREAD0
	AMQWRIT
	AMQWRIT0
	Sample programs (C)
	AMQSBCG
	AMQSBCG0
	AMQSECH
	AMQSECHA
	AMQSGBR
	AMQSGBR0
	AMQSGET
	AMQSGET0
	AMQSINQ
	AMQSINQA
	AMQSIPC
	AMQSIPC0
	AMQSPUT
	AMQSPUT0
	AMQSREQ
	AMQSREQ0
	AMQSSET
	AMQSSETA
	AMQSTRG
	AMQSTRG0
	Sample programs (TAL)
	ZMQREAD

	ZMQREADT	
	ZMQWRITE	
	ZMQWRITT	
	Sample exits (C)	
	AMQSVCHN	Sample C channel exit.
	AMQSVFCN	Sample C data-conversion exit.
	C sample scripts and macros (nonnative)	
	BINDALL	Script to bind all C samples using BSAMP macro.
	BSAMP	Macro to bind C samples (nonnative).
	BUILDC	Script to compile and bind all C samples (nonnative).
	COMPALL	Script to compile all C samples (nonnative).
	CSAMP	Macro to compile C samples (nonnative).
	BDXALL	Script to bind data-conversion exits into MQSeries executables.
	BCHXALL	Script to bind channel exits into MQSeries executables.
	BEXITE	Macro to bind an exit library into an executable.
	AXCELSMP	Accelerate C sample programs.
	C sample scripts and macros (native)	
	BUILDNMC	Script to compile and bind all C samples (native).
	NLDALL	Script to link all C samples using NMLDSAMP macro.
	NMCALL	Script to compile all C samples using NMCSAMP macro.
	NMCPSRL	Macro to compile code that can be included in the MQSeries private SRL.
	NMCSAMP	Macro to compile C samples.
	NMLDPSRL	Link a user MQI application with the MQSeries private SRL.
	NMLDSAMP	Macro to bind a C sample application with the static MQI library.
	NMLDUSRL	Macro to create a relinkable object library that can be included in the MQSeries private SRL.
	COBOL sample scripts and macros	
	BUILDCOB	Script to compile and bind all COBOL samples.
	BCBSMPLS	Script to bind COBOL sample programs using BCOBSAMP.
	BCOBSAMP	Macro to bind COBOL sample programs.

Verifying your installation

	CCBSMPLS	Script to compile all COBOL samples using macro COBSAMP.
	COBSAMP	Macro to compile COBOL samples.
	TAL sample scripts and macros	
	BTALSAMP	Macro to bind TAL samples.
	TALSAMP	Macro to compile TAL samples.
	BTLSMPLS	Script to bind all TAL samples.
	BUILDTAL	Script to compile and bind all TAL samples.
	CTLSMPLS	Script to compile all TAL samples.

Data conversion tables (ZMQSCONV)

See the *MQSeries Application Programming Reference* book for information about conversions supported by MQSeries for Tandem NSK.

Verifying your installation

When you have installed MQSeries for Tandem NSK V2.2.0.1 and its samples components, you should verify that the installation has completed successfully. The following steps explain how to verify your installation using the MQSC command file amqscoma. The commands in this file initialize your MQSeries queue manager and set up the default objects that your queue manager requires. The objects that amqscoma creates for you are listed in Appendix C, "System defaults" on page 293.

When you have completed the verification, you should delete the queue manager to leave a 'clean' system, that is, a system with no objects, including queue managers, defined.

Note: Deleting the queue manager does not delete the installation. You can use this procedure even if you have previously performed it.

Follow these steps to verify your installation

The following instructions use the name QMNAME for the queue manager. When creating your own queue manager, replace each occurrence of QMNAME with the appropriate name. A queue manager name must be unique within your network. The commands in these steps are case sensitive.

1. Create a queue manager called QMNAME using the **crtmqm** command. For example:

```
crtmqm -n $QMNA -o $TRMG.#A -s $QMSS QMNAME
```

The **crtmqm** command requires the process names of a PATHMON process and a default MQSS server process to use for the queue manager. These process names must be unique on the system.

It also requires the name of a home terminal, which must be paused.

You must enter the following options before the name of the queue manager:

– n PATHMON process name

- o home terminal (must be paused)
- s MQSS Server process name

For a detailed description of the **crtmqm** command and options see “crtmqm (Create queue manager)” on page 234

2. Start the queue manager using the **strmqm** command. For example:

```
strmqm QMNAME
```

The **strmqm** command returns control when the queue manager has started and is ready to accept connect requests.

3. Create the default objects for this queue manager. For example:

```
runmqsc -i $SYSTEM.ZMQSSMPL.AMQSCOMA -o defobjou QMNAME
```

The file `amqscoma` contains a sequence of MQSC commands that define the system default objects for the queue manager `QMNAME`. Note that you must specify a fully qualified file name for `amqscoma` if you are not in the subvolume `ZMQSSMPL`. The output from the MQSC commands is sent to a report file `defobjou`. Examine the last two lines of the output file to verify that all commands were processed without error. If errors have occurred, you should examine the rest of this file, checking the confirmation messages for each MQSC command. For example:

```
AMQ8006 MQSeries queue created
```

If no errors are indicated, all commands were successful and you have verified that your installation was successful.

You might want to modify a copy of `amqscoma` to meet your own requirements for system defaults.

4. Stop the queue manager using the command:

```
endmqm QMNAME
```

5. Delete the queue manager using the command:

```
dltmqm QMNAME
```

This command deletes the queue manager and its associated objects including the system default objects that you created.

Troubleshooting

If your installation was not successful or commands (including those run from `amqscoma`) failed to run properly, consider the following:

- *Did you enter the commands correctly?*

Try running one or more of the commands again. These commands and most parameters are case sensitive. If you create a queue manager with an uppercase name, you must specify the name in uppercase on any commands referring to that queue manager. For example, if you create a queue manager called `QMNAME`, you cannot use `'qmname'` or `'QMname'`.

- *Do you have enough disk space or memory to run the verification?*

Check any error messages for an indication. If error message `AMQ7065` Insufficient space on disk is returned, use the `dsap` utility to display the free space on the target volume. If there is insufficient free space, choose a different volume on which to create the queue manager, or free some space on the existing volume.

- *Do the required subvolumes for the installed product exist?*

If they do not exist, attempt to reinstall.

- *Do you have the required authority to run the commands?*

Check that you are still logged in as a member of the `MQM` group.

- *Is the home terminal you specified on the `crtmqm` command correct?*

- *Is the home terminal paused?*

- *Are the `PATHMON` and default `MQSS` server process names you specified for the `crtmqm` command unique in your system?*

- *Is `TM/MP (TMF)` running?*

Upgrading a Version 2.2 queue manager

To upgrade a V2.2 queue manager, use the `upgmqm` command. The utility invoked by `upgmqm` sends progress messages to the terminal from which it is started. When it has completed, the queue manager is ready to use with V2.2.0.1. Attributes new in V2.2.0.1 are set to their default values. You can alter these in the usual way.

For the syntax of the `upgmqm` command, see “`upgmqm (Upgrade V2.2 queue manager)`” on page 276.

Chapter 3. Customizing your system

This chapter identifies some of the tasks involved in customizing a queue manager to meet your requirements.

Do I need to customize?

When you have installed the product, you can use it without having to customize it in any way. The default configuration provides all the facilities you need to build a working system that can participate in message queuing with other MQSeries systems.

However, you must have set up the required Tandem NSK user and group IDs.

When do I customize?

Some customization tasks must be performed *before* you create a queue manager; others require you to stop and restart the queue manager. Check each task in turn, to see when you need to perform it.

What are configuration files?

There are two types of configuration file. One contains information about the way your MQSeries system is set up or configured; this file is created when MQSeries is installed. The other contains information about the attributes of an individual queue manager. This file is generated when a queue manager is created.

This chapter specifies which of these files to modify for each relevant configuration task. For more information about the files themselves, see Chapter 13, "Configuration files" on page 183.

What do I do now?

Check each item in the following list to see whether any of the things that you can customize applies to you:

- "Configuring an authorization service component" on page 36
- "Enabling users to access MQSeries" on page 36
- "Enabling communications support" on page 37
- "Implementing data conversion" on page 37
- "Defining the default and system objects" on page 37
- "Specifying volumes for queue manager objects" on page 38
- "Sharing queues using the name service" on page 39
- "Configuring a queue manager" on page 39
- "Specifying the location of the machine-wide INI file" on page 39
- "Specifying the location of the MQSeries executables" on page 39
- "Configuring MQSS Servers" on page 40
- "Specifying retrieval options for queues" on page 40
- "Configuring MQSeries to work with RDF" on page 41

The terms in this list are explained in the following sections.

Configuring an authorization service component

- *This task is not required on your first pass through this book.*
- *By default, authority checking is switched on.*

The authorization service supports authority checking on commands and MQI calls for the user ID associated with the command or call. The names of the authorization service and the component that implements the service are specified in the queue manager configuration file (QMINI).

By default, the active authorization service component is the Object Authority Manager (OAM), which is supplied with MQSeries for Tandem NSK.

Changing the authorization service component

You can edit the configuration file for a specific queue manager to:

- Remove the OAM and therefore all security checking.
- Replace the OAM with a user-written authorization service component.
- Add a user-written authorization service component to augment the OAM.

These tasks are not required, unless you have specific security requirements that cannot be accommodated by the OAM.

For more information about the queue manager configuration file, see “Queue manager configuration file (QMINI)” on page 185. For information about writing your own authorization service component, refer to the *MQSeries Programmable System Management* manual.

Note: You can change the configuration file QMINI after you have created and started the queue manager to which it relates. This has no effect until the queue manager is stopped and restarted. However, you should not create or change objects when the authorization service is off and then turn authorization back on again. If you do, you may compromise the security of your system.

Enabling users to access MQSeries

- *This task is required before users other than the creator of the queue manager can access that queue manager's objects, regardless of whether the OAM is enabled.*

For each queue manager you create, you **must** create MQSeries principals corresponding to each user that will access the queue manager. This can be done before the queue manager is started. See “Creating MQSeries principals” on page 73 for more information.

Enabling communications support

- *This task is required before you can communicate with other queue managers.*

You must specify the name of the communications protocol and other parameters that are to be used for communication with other queue managers. This includes the LAN protocol name, which must be one of the following:

- SNA LU 6.2
- TCP/IP

By default, these protocols are enabled. For further information about setting up communications, see Appendix J, “Setting up communications” on page 333 and the *MQSeries Intercommunication* book.

Implementing data conversion

- *This task is not normally required on your first pass through this book.*
- *You do not need data conversion to communicate between similar nodes.*

If you are using MQSeries with systems that have different encodings, you might need to use a data conversion exit. The conversion of a message is based on the message format (MQFMT), specified in the message descriptor. All IBM message formats are converted automatically. However, user-defined formats are not converted automatically, so that even ASCII-to-EBCDIC conversion must be done using a data-conversion exit (one per format).

You can use the supplied conversion exit utility if you wish to communicate with queue managers using MQI calls or remote commands, where the systems involved have formats other than those supported by MQSeries. The conversion exit utility lets you create the required conversions as C source code. Refer to the *MQSeries Application Programming Guide* for more information. You can leave this task until run time. However, if you do, you may not be able to communicate between the two different machines until then.

Supported code sets

MQSeries for Tandem NSK supports the code sets identified in “Internationalization” on page 281.

Defining the default and system objects

- *This task is required, but is part of the standard administration procedures. See Chapter 5, “Managing queue managers” on page 59.*

MQSeries for Tandem NSK provides an MQSC command file that you can use to set up the default and system objects. Typically, when you define an object, you do not define all the possible attributes. Those you do not specify are inherited from the corresponding default object. The supplied command file `amqscoma`, when used with the `runmqsc` command, creates a set of default and system objects. See “Creating the default and system objects” on page 74 for information about running this sample.

If you change the attributes of the default object, any objects of the same type you create inherit the new values. Do not attempt to change these attributes if you are not familiar with the various command sets provided with MQSeries for Tandem NSK.

Modifying the amqscoma command file

You should consider modifying the command file amqscoma if, for example:

- You have a large number of objects to create with the same, nondefault values.
- You have some specific requirements or limitations on the size of some resources.

To modify amqscoma, make a backup copy, make the required changes, then use the new version of the file to create the default objects.

Specifying volumes for queue manager objects

- *This task is not normally required on your first pass through this book.*
- *By default, the volumes are already set.*

Each queue manager is created on a specific volume. The queue manager subvolumes are all created on this volume. The default volume for the creation of queue managers is set in the AllQueueManagers stanza of the MQSINI file. The entry QMDefaultVolume is the default volume for the creation of queue managers for the installation.

The **crtmqm** command can override the default volume. For more information, see “crtmqm (Create queue manager)” on page 234.

Once a queue manager has been created on a volume, there are no facilities provided for moving an entire queue manager to a different volume, without deleting and recreating.

There are limited facilities for altering the volume on which a queue is stored, on a queue-by-queue basis. See the description of the **altmqfls** command in “altmqfls (Alter queue file attributes)” on page 220 for more information. However, there are no facilities for moving other queue manager objects: even if all queues are moved to alternative volumes, the queue manager’s “home” location is unaltered.

In addition to the facility for moving queues, the Tandem NSK ENSCRIBE file system can partition queue files across volumes. Partitioning can be used to overcome disk-space problems on specific volumes, or to distribute the physical I/O activity across multiple volumes.

The partitioning used on the queue files is “intelligent.” That is, the key used for partitioning purposes is constructed dynamically, thereby distributing the messages randomly across the available partitions. An example TAQL macro that partitions a queue file across two disk volumes is supplied with MQSeries for Tandem NSK. The partitioning macro, called **partit** in the samples subvolume ZMQSSMPL, can be adapted to support more than two disk volumes.

Note that there are some restrictions on the use of partitioning for queues with nondefault message-retrieval settings. For more information, see “Specifying retrieval options for queues” on page 40.

Specifying a default volume for queue managers

You can specify a default volume, so that when you create a new queue manager its volume is taken from the default. The default volume is specified in the `QMDefaultVolume` stanza in the `MQSINI` file. Unless you have changed it, the default volume is as specified at install time.

Sharing queues using the name service

- *This task is not normally required on your first pass through this book.*
- *You must stop and restart the queue manager to perform this task.*

The name service is an installable service that enables an application to access a queue on another queue manager as if it were a local queue. For MQI requests, applications can then treat this queue like a local queue, without being aware of its exact location.

The service name and the component to be invoked for that service are specified by stanzas in the `QMINI` configuration file. By default, this service is not active. For information about configuration file stanzas, see Chapter 13, “Configuration files” on page 183. For information about writing your own name service, see the *MQSeries Programmable System Management* manual.

Configuring a queue manager

- *This task is required, but is part of the standard administration procedures. See Chapter 5, “Managing queue managers” on page 59.*

When you create a queue manager, using the `crtmqm` command, you can specify certain properties for that queue manager. For example, you can specify the name of the dead-letter queue, and the default transmission queue.

Once you have created a queue manager, you might need to modify its properties. For more information, see “Guidelines for creating a queue manager” on page 59 and Chapter 13, “Configuration files” on page 183.

Specifying the location of the machine-wide INI file

The `PARAM MQMACHINIFILE`, if defined, overrides the default location of the `MQSINI` file for all administration commands.

Specifying the location of the MQSeries executables

Edit the stanza `MQExePath` in the `MQSINI` file for your installation to change the location of the executables for the installation. `MQExePath` is updated automatically during installation of MQSeries for Tandem NSK.

Configuring MQSS Servers

- *This task is not normally required on your first pass through this book.*

Each queue manager has a default MQSS Server. The standard administration procedures include the naming of the default MQSS Server process pair, and there is no need to configure additional MQSS Servers in order to use MQSeries. However, you can modify the initial TS/MP configuration of the default MQSS Server processes to change the CPUs in which the processes run. You do this using the standard TS/MP Pathcom utility while the queue manager is stopped.

As your capacity or performance needs dictate, you can also add additional MQSS Servers to a queue manager and distribute existing queues among several MQSS Servers. This approach might help in situations where a heavy processing load must be supported by several queues, or when queue managers support a large number of objects.

You create additional MQSS Servers by modifying the TS/MP configuration for the queue manager using the Pathcom utility. Objects are assigned to specific MQSS Servers using the **altmqfls** control command.

You can also use the **altmqfls** command to control when the responsible MQSS Server recalculates the queue depth for a queue, and updates the object catalog with the latest status information. However, there is no need to change the default configuration. These options give good control over the correction of the CURDEPTH attribute for local queues, and specifically the trade-off between the resource usage associated with the recalculation and the accuracy of the depth count.

Specifying retrieval options for queues

- *This task is not normally required on your first pass through this book.*

When a local queue is created (using **runmqsc**, the MQM panels, or PCF commands) it is always configured so that the MQGET call can get or browse messages using all the standard MQSeries retrieval criteria (FIFO, *Msgld*, *Correlld*, and the combination of *Msgld* and *Correlld*).

If retrieval by *Msgld*, *Correlld*, or the combination of *Msgld* and *Correlld* is not required for specific queues, **altmqfls** can be used to remove those retrieval methods from a local or model queue. The removal of unused retrieval methods can give a significant performance improvement because it results in reduced I/O to the disk for MQPUT and MQGET operations. The default retrieval mechanism (FIFO or FIFO by priority) is always available and cannot be removed using **altmqfls**. The **altmqfls** command can also be used to reenable retrieval mechanisms that have been previously disabled.

If an application attempts to use a retrieval method that is not supported for a queue, one of two new MQRC_ reason codes (MQRC_MQG_ID_ERROR or MQRC_CORREL_ID_ERROR) is returned to the application indicating that the requested retrieval mechanism is not supported.

Note: A queue that has had all message-retrieval options disabled (that is, a queue that has no alternate key files) **cannot** be partitioned across multiple volumes.

Configuring MQSeries to work with RDF

- *This task is not normally required on your first pass through this book.*

To make best use of RDF, you are recommended to customize some of the default configuration settings, as follows:

1. Ensure that RDF is set up with identical volume names on both the primary and backup sites.
2. Ensure that the backup site volume on which the queue manager will be created does not have files in the subvolumes in which the new queue manager will be created.
3. Create the queue manager using **crtmqm**.
4. Change the TS/MP configuration for the queue manager manually (using PATHCOM) to include the definition of the MQRDF PARAM in all server classes (for example, MQS-EC00).
5. Add the MQRDF PARAM to the TACLSTM files of all users who will be performing administrative operations on the queue manager (normally, those users in the MQM group only).
6. Start the queue manager using **strmqm**.
7. Create the default system objects using **runmqsc**.
8. Run **cleanrdf** for the first time to establish a baseline configuration for the nonaudited database files for the queue manager.
9. Verify the basic operation as follows:
 - a. Create a local queue using **runmqsc**.
 - b. Use a sample program to put a few messages on the queue.
 - c. Use a sample program to retrieve the messages.
 - d. Delete the queue using **runmqsc**.
 - e. Run **cleanrdf**.
 - f. Recreate the queue using the same name.

During these steps, monitor the RDF backup site. No EMS events indicating file-system errors should be generated.

Note: RDF operation is not instantaneous, so cannot guarantee no message loss.

Chapter 4. Using administration command sets

This chapter describes the commands you can use for performing system administration tasks on MQSeries objects. Administration tasks include creating, starting, altering, viewing, stopping, and deleting queue managers, queues, processes, and channels. To perform these tasks, you must select the appropriate command.

MQSeries for Tandem NSK V2.2.0.1 provides the following administration command sets for performing administrative tasks:

- MQSC (MQSeries commands)
- PCF (Programmable Command Format) commands
- Control commands

In addition:

- Some TS/MP (Pathway) commands are used for administration purposes.
- The MQM (Message Queue Management) facility supports some administration tasks. The MQM is described in “Using the Message Queue Management (MQM) facility” on page 78.

This chapter introduces the MQSC, PCF, and control command sets, and provides a summary of the functions supported by each command set in “Comparing command sets” on page 45. How to use TS/MP commands is described in “TS/MP (Pathway) administration” on page 48.

Control commands

The following types of control command are available:

- Commands for creating, starting, stopping, and deleting queue managers
- Commands for starting, stopping, and displaying command servers
- Utility commands associated with, for example, running MQSC commands, managing access to MQSeries objects, starting and stopping an MQSeries trace, and running trigger monitors

Using control commands

You run control commands from the Tandem TACL prompt. Command names are not case sensitive. (Note, however, that queue manager names *are* case sensitive.) For example:

```
runmqsc
```

Chapter 15, “The MQSeries control commands” on page 217 explains the syntax and purpose of each command.

MQSC commands

You can use the MQSC commands to manage queue manager objects including the queue manager, channels, queues, and process definitions. For example, you can define, alter, display, and delete a specified queue using MQSC commands.

When you display a queue, using the DISPLAY QUEUE command, you display the queue *attributes*. For example, the MAXMSGL attribute specifies the maximum length of a message that can be put on the queue. The command does not show you the messages on the queue. These commands are summarized in “Comparing command sets” on page 45. For detailed information about each MQSC command, see the *MQSeries Command Reference*.

Running MQSC commands

You can run MQSC interactively by invoking the control command **runmqsc** from the Tandem TACL prompt or running a script when a local queue manager is running.

You can run the **runmqsc** command itself in three modes, depending on the flags set on the command:

- *Verification mode*, where the MQSC commands are verified on a local queue manager, but are not run.
- *Direct mode*, where the MQSC commands are run on a local queue manager.
- *Indirect mode*, where the MQSC commands are run on a remote queue manager.

For more information about using the MQSC facility and text files, see “Entering MQSC interactively” on page 100. For more information about the **runmqsc** command, see “runmqsc (Run MQSeries commands)” on page 261.

PCF commands

PCF commands let you program administrative tasks into your applications or an administration program. PCF commands cover the same range of functions that are provided by the MQSC facility. You can write a program to issue PCF commands to any queue manager in the network from a single node. You can also centralize and automate administration tasks.

Each PCF command is a data structure that is embedded in the application data part of an MQSeries message. Each command is sent to the target queue manager using the MQI function MQPUT. The command server on the queue manager receiving the message interprets it as a command message and runs the command. To get the replies, the application issues an MQGET call and the reply data is returned as a data structure in the application data part of the MQSeries message. The application can then process the reply and act accordingly.

Note: Unlike MQSC commands, PCF commands and their replies are not in a text format that you can read.

You must specify the following items to create a PCF command message:

Message descriptor

This is a standard MQSeries message descriptor, in which:

Message type (*MsgType*) is MQMT_REQUEST.
 Message format (*Format*) is MQFMT_ADMIN.

Application data

Contains the PCF message including the PCF header, in which:

The PCF message type (*Type*) specifies MQCFT_COMMAND.
 The command identifier specifies the command, for example, *ChangeQueue* (MQCMD_CHANGE_Q).

For a complete description of the PCF data structures and how to implement them, see the *MQSeries Programmable System Management* book.

Attributes in MQSC and PCFs

Object attributes specified in MQSC are in uppercase (for example, RQMNAME), although they are not case sensitive. These attribute names are limited to eight characters (for example, QDPHIEV). Object attributes in PCF are shown in italics, and are not limited to eight characters. The PCF equivalent of RQMNAME is *RemoteQMgrName* and of QDPHIEV is *QDepthHighEvent*.

Escape PCFs

Escape PCFs are PCF commands that contain MQSC commands within the message text. You can use PCFs to send commands to a remote queue manager. For more information about using escape PCFs, see the *MQSeries Programmable System Management* book.

Comparing command sets

Table 1 through Table 6 on page 47 compare the facilities available from the different administration command sets.

Note: Only those MQSC commands that apply to MQSeries for Tandem NSK are shown.

PCF	MQSC	Control
Change Queue Manager	ALTER QMGR	–
(Create queue manager)*	–	crtmqm
(Delete queue manager)*	–	dltmqm
Inquire Queue Manager	DISPLAY QMGR	–
(Stop queue manager)*	–	endmqm
Ping Queue Manager	PING QMGR	–
(Start queue manager)*	–	strmqm
Note: * Not available as PCF commands.		

Comparing command sets

<i>Table 2. Commands for command server administration</i>	
Description	Control
Display command server	dspmqcsv
Stop command server	endmqcsv
Start command server	strmqcsv
Note: As an alternative to the control commands, you may use PATHCOM commands, as described in "TS/MP (Pathway) administration" on page 48. There are no MQSC or PCF equivalents of commands in this group.	

<i>Table 3. Commands for queue administration</i>	
PCF	MQSC
Change Queue	ALTER QLOCAL ALTER QALIAS ALTER QMODEL ALTER QREMOTE
Clear Queue	CLEAR QLOCAL
Copy Queue	DEFINE QLOCAL(x) LIKE(y) DEFINE QALIAS(x) LIKE(y) DEFINE QMODEL(x) LIKE(y) DEFINE QREMOTE(x) LIKE(y)
Create Queue	DEFINE QLOCAL DEFINE QALIAS DEFINE QMODEL DEFINE QREMOTE
Delete Queue	DELETE QLOCAL DELETE QALIAS DELETE QMODEL DELETE QREMOTE
Inquire Queue	DISPLAY QUEUE
Inquire Queue Names	DISPLAY QUEUE
Note: There are no control commands for these functions.	

<i>Table 4. Commands for process administration</i>	
PCF	MQSC
Change Process	ALTER PROCESS
Copy Process	DEFINE PROCESS(x) LIKE(y)
Create Process	DEFINE PROCESS
Delete Process	DELETE PROCESS
Inquire Process	DISPLAY PROCESS
Inquire Process Names	DISPLAY PROCESS
Note: There are no control commands for these functions.	

Table 5. Commands for channel administration

PCF	MQSC	Control
Change Channel	ALTER CHANNEL	–
Copy Channel	DEFINE CHANNEL(x) LIKE(y)	–
Create Channel	DEFINE CHANNEL	–
Delete Channel	DELETE CHANNEL	–
Inquire Channel	DISPLAY CHANNEL	–
Inquire Channel Names	DISPLAY CHANNEL	–
Inquire Channel Status	DISPLAY CHSTATUS	–
Ping Channel	PING CHANNEL	–
Reset Channel	RESET CHANNEL	–
Resolve Channel	RESOLVE CHANNEL	–
Start Channel	START CHANNEL	runmqchl
Start Channel Initiator	–	runmqchi
Start Channel Listener	–	runmqlsr
Stop Channel	STOP CHANNEL	–

Note: In MQSeries for Tandem NSK, use TS/MP or the control command **runmqlsr** to start TCP/IP channel listeners. For more information, see “Specifying and controlling TCP/IP listeners” on page 48 and “runmqlsr (Run listener)” on page 260.

Table 6. Other control commands

Description	Control
Alter queue volume, recalculation, update, or retrieval options for an object	altmqfls
Add, delete, or alter MQSeries principals	altmqusr
RDF housekeeping utility	cleanrdf
Convert V1.5.1 queues and channels to V2.2.0.1	cnv1520
Convert client channel definition table	cnvclchl
Convert V1.5.1 messages to V2.2.0.1	cnvmsgsgs
Create MQSeries conversion exit	crtmqcvx
Display authority	dspmqaout
Display files used by objects; recalculation, update, and retrieval options configured for an object	dspmqfls
Display MQSeries formatted trace output	dspmqtrc
Display principals	dspmqusr
End MQSeries trace	endmqtrc
Install MQSeries for Tandem NSK	instmqm
Run dead-letter queue handler	runmqdlq
Run MQSC commands	runmqsc
Run trigger monitor	runmqtrm
Set or reset authority	setmqaut
Start MQSeries trace	strmqtrc
Upgrade V2.2 queue manager	upgmqm

Note: As an alternative to the control command **runmqtrm**, you may use PATHCOM commands, as described in “TS/MP (Pathway) administration” on page 48. There are no MQSC or PCF equivalents of commands in this group.

TS/MP (Pathway) administration

Most operations on the queue manager are accomplished by running MQSeries control commands from TACL. Some operations, however, require the use of PATHCOM to operate directly on the TS/MP server classes. Also, because of system-configuration changes, you might need to perform some administration actions on the TS/MP configuration itself.

This section summarizes these activities.

Specifying and controlling TCP/IP listeners

To start TCP/IP listeners, you can use the MQSeries control command **runmqclsr** (described in “runmqclsr (Run listener)” on page 260), or you can use the PATHCOM commands **THAW SERVER** and **START SERVER**. To stop TCP/IP listeners, use the PATHCOM commands **FREEZE SERVER** and **STOP SERVER**. Use the PATHCOM command **STATUS SERVER** to display the number of TCP/IP listeners running, and their process names.

By default, each queue manager has one listener that is in server class MQS-TCPLIS00. Use the PATHCOM command **ADD SERVER** to create additional TCP/IP listener server classes to service more than one TCP/IP port. Each TCP/IP listener should be configured in its own TS/MP server class for maximum flexibility. If you add TCP/IP listeners, you must also add TCP/IP port definitions to the queue manager initialization file (QMINI), as described in “TCP/IP ports listened on by the queue manager” on page 66. The first listener to be started uses the first listener port defined in QMINI, the second listener uses the second listener port, and so on. For an example of the QMINI entries required to support multiple listeners, see “Configuring QMINI to support multiple TCP/IP listeners” on page 347.

Controlling the command server

The command server is created as the TS/MP server class MQS-CMDSERV00. As an alternative to the control commands **strmqcsvg**, **endmqcsvg**, and **dspmqcsvg**, you can use the PATHCOM commands **THAW SERVER**, **START SERVER**, **FREEZE SERVER**, **STOP SERVER**, and **STATUS SERVER**.

Specifying and controlling channel initiators

The default channel initiator is created as the TS/MP server class MQS-CHANINIT00. As an alternative to using the **runmqchi** control command (described in “runmqchi (Run channel initiator)” on page 257), you can use the PATHCOM commands **THAW SERVER**, **START SERVER**, **FREEZE SERVER**, **STOP SERVER**, and **STATUS SERVER** to control and display the status of the channel initiator. The default channel initiator processes the default channel initiation queue, SYSTEM.CHANNEL.INITQ.

Changing the default initiation queue for the channel initiator

If you want to use an initiation queue other than the default (SYSTEM.CHANNEL.INITQ), you must change the Pathway configuration. You can perform this task while the queue manager is running, but the channel initiator server class itself must be stopped. In Pathcom, issue the following command against the queue manager's Pathway configuration:

```
ALTER SERVER MQS-CHANINIT00, STARTUP "-q<init-queue>"
```

where *<init-queue>* is the name of the alternative initiation queue. You can then start the channel initiator and exit Pathcom.

Specifying and controlling trigger monitors

A single default trigger monitor is created as the TS/MP server class MQS-TRIGMON00. You can use the PATHCOM commands THAW SERVER, START SERVER, FREEZE SERVER, STOP SERVER, and STATUS SERVER to administer this server class. If you need additional trigger monitors, you can configure them as additional server classes, using MQS-TRIGMON00 as a template. You are recommended to use separate server class objects for maximum flexibility. You do not have to use TS/MP to control trigger monitors. For example, you can run the trigger monitor from TACL using the control command **runmqtrm**.

The default trigger monitor processes the default initiation queue, SYSTEM.DEFAULT.INITIATION.QUEUE. You can change this by adding or changing the STARTUP message for the server class that holds the trigger monitor. You need to do this if more than one trigger monitor is configured for the queue manager. Use the PATHCOM ALTER SERVER command to add or change the STARTUP attribute.

Specifying the distribution of processes across CPUs

The key to the distribution of work among CPUs is the CPU assigned to each EC in the queue manager. Each EC creates and manages a set of agent processes in its own CPU only. Consequently, if the EC processes are distributed among the CPUs of the system, the agent processes are similarly distributed.

By default, the EC processes (each a separate server class) are distributed as evenly as possible among the available CPUs on the system. There is no built-in limit to the number of EC processes in a queue manager: the number required depends entirely on the load to be handled by the queue manager. By default, there is one EC process in the queue manager.

The default EC server class is called MQS-EC00. Specify the **-e** flag of the **crtmqm** command to create a queue manager with more than one EC. The number of EC processes may be changed after the queue manager has been created by adding or deleting EC process server classes, and making a corresponding modification to the ExpectedNumECs entry in the ECBoss stanza in the QMINI file.

Each EC process *must* be in its own server class. Use the MQS-EC00 server class as a template if you need to create additional EC processes manually.

The default assignment of CPUs to EC processes, or any other server class, may be changed using the PATHCOM ALTER SERVER command on the CPU attribute.

The default MQSS Server is automatically created by **crtmqm** in the server class MQS-STATUS00. By default, the only CPU assignment made is CPU 0. The CPU assignment for the MQS-STATUS00 server can be changed using the PATHCOM ALTER SERVER command. You can specify a specific backup CPU for the MQSS Server by providing two CPU numbers separated by a colon, for example CPUS(2:12). In this case, PATHMON creates the primary in CPU 2 and the backup in CPU 12. If a specific backup CPU is not provided, the Tandem NSK operating system decides where to create the backup.

Addition of new MQSS Server processes

To add additional MQSS Servers to a queue manager, create a server class using the default MQSS Server class MQS-STATUS00 as a template.

The name of any new MQSS Server class should begin with the character string MQS-STATUS. If the server class names do not follow this naming convention, **strmqm** will not start them automatically on queue manager startup, and access to any objects that are configured for these MQSS Server classes will be disabled.

If additional MQSS Servers are configured, they each need unique process names. You are also recommended to configure them to run in different CPUs in order to benefit from the scalability that the MQSS Server architecture provides.

Specifying the refresh frequency of MQM monitor panels

The MQMQMREFRESHINT Pathway parameter for MQS-MQMSVR00 determines the frequency with which monitor screens for channels and queues are refreshed. The default frequency is every 30 seconds. To change the frequency to every 10 seconds, for example, enter:

```
alter server mqs-mqmsvr00, param mqmqmrefreshint 10
```

from the pathway for your queue manager.

PATHWAY configuration for a queue manager

Here is an example PATHWAY configuration for a queue manager. This example was generated by issuing a sequence of INFO commands on the objects in a default queue manager configuration.

```

PATHMON
  BACKUPCPU 3
  DUMP OFF

PATHWAY
  MAXASSIGNS 20           [CURRENTLY 0]
  MAXDEFINES 100         [CURRENTLY 0]
  MAXEXTERNALTCPS 0      [CURRENTLY 0]
  MAXLINKMONS 16         [CURRENTLY 2]
  MAXPARAMS 100          [CURRENTLY 8]
  MAXPATHCOMS 10         [CURRENTLY 1]
  MAXPROGRAMS 100        [CURRENTLY 1]
  MAXSERVERCLASSES 100   [CURRENTLY 8]
  MAXSERVERPROCESSES 100 [CURRENTLY 8]
  MAXSPI 10              [CURRENTLY 0]
  MAXSTARTUPS 100        [CURRENTLY 0]
  MAXTCPS 100            [CURRENTLY 1]
  MAXTELLQUEUE 4
  MAXTELLS 32            [CURRENTLY 0]
  MAXTERMS 100           [CURRENTLY 0]
  MAXTMFRESTARTS 5
  OWNER \RAPTOR.44,1
  SECURITY "N"

```

Figure 2 (Part 1 of 6). Example PATHWAY configuration

```
TCP MQS-TCP-01
  AUTORESTART 0
  CHECK-DIRECTORY OFF
  CODEAREALEN 80000
  CPUS 0:1
  DEBUG OFF
  DUMP OFF
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  INSPECT OFF
  MAXINPUTMSGLEN 6000
  MAXINPUTMSGS 0
  MAXPATHWAYS 0
  MAXREPLY 32000
  MAXSERVERCLASSES 1
  MAXSERVERPROCESSES 10
  MAXTERMDATA 500000
  MAXTERMS 10
  NONSTOP 0
  POWERONRECOVERY ON
  PRI 175
  PROGRAM \RAPTOR.$SYSTEM.SYSTEM.PATHTCP2
  SERVERPOOL 32000
  STATS OFF
  TCLPROG \RAPTOR.$DEV.ZMQSEXE.POBJ
  TERMBUF 1500
  TERMPPOOL 10000

PROGRAM MQMC
  ERROR-ABORT OFF
  OWNER \RAPTOR.44,1
  SECURITY "N"
  TCP MQS-TCP-01
  TMF ON
  TYPE T16-6520 (BREAK OFF,ECHO ON,EXCLUSIVE OFF,INITIAL MAINC,IOPROTOCOL
0,MAXINPUTMSGS 0,TRAILINGBLANKS ON)
```

Figure 2 (Part 2 of 6). Example PATHWAY configuration

```

SERVER MQS-CHANINIT00
  PROCESSTYPE GUARDIAN
  AUTORESTART 10
  CPUS (0)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT \RAPTOR.$ZTN0.#PTY001C
  OWNER \RAPTOR.44,1
  PARAM MQQUEMGRNAME "p101"
  PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
  PARAM MQDEFAULTPREFIX "$DEV"
  PRI 175
  PROGRAM \RAPTOR.$DEV.ZMQSEXE.RUNMQCHI
  SECURITY "N"
  TMF ON
  VOLUME \RAPTOR.$DATA1.P101D

SERVER MQS-CMDSERV00
  PROCESSTYPE GUARDIAN
  AUTORESTART 10
  CPUS (0)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT \RAPTOR.$ZTN0.#PTY001C
  OWNER \RAPTOR.44,1
  PARAM MQQUEMGRNAME "p101"
  PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
  PARAM MQDEFAULTPREFIX "$DEV"
  PRI 175
  PROGRAM \RAPTOR.$DEV.ZMQSEXE.MQCMDSVR
  SECURITY "N"
  TMF ON
  VOLUME \RAPTOR.$DATA1.P101D

```

Figure 2 (Part 3 of 6). Example PATHWAY configuration

```

SERVER MQS-EC00
  PROCESSTYPE GUARDIAN
  AUTORESTART 10
  CPUS (0)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT \RAPTOR.$ZTN0.#PTY001C
  OWNER \RAPTOR.44,1
  PARAM MQQUEMGRNAME "p101"
  PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
  PARAM MQDEFAULTPREFIX "$DEV"
  PRI 175
  PROGRAM \RAPTOR.$DEV.ZMQSEXE.MQEC
  SECURITY "N"
  TMF ON
  VOLUME \RAPTOR.$DATA1.P101D

SERVER MQS-ECBOSS
  PROCESSTYPE GUARDIAN
  AUTORESTART 10
  CPUS (0)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT \RAPTOR.$ZTN0.#PTY001C
  OWNER \RAPTOR.44,1
  PARAM MQQUEMGRNAME "p101"
  PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
  PARAM MQDEFAULTPREFIX "$DEV"
  PRI 175
  PROGRAM \RAPTOR.$DEV.ZMQSEXE.MQECBOSS
  SECURITY "N"
  TMF ON
  VOLUME \RAPTOR.$DATA1.P101D

```

Figure 2 (Part 4 of 6). Example PATHWAY configuration

```

SERVER MQS-MQMSVR00
  PROCESSTYPE GUARDIAN
  AUTORESTART 0
  CPUS (0)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT \RAPTOR.$ZTN0.#PTY001C
  OWNER \RAPTOR.44,1
  PARAM MQQUEMGRNAME "p101"
  PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
  PARAM MQDEFAULTPREFIX "$DEV"
  PARAM MQMPAGESTORETRIEVE "20"
  PRI 175
  PROGRAM \RAPTOR.$DEV.ZMQSEXE.MQMSVR
  SECURITY "N"
  TMF ON
  VOLUME \RAPTOR.$DATA1.P101D

SERVER MQS-STATUS00
  PROCESSTYPE GUARDIAN
  AUTORESTART 0
  CPUS (0)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT \RAPTOR.$ZTN0.#PTY001C
  OWNER \RAPTOR.44,1
  PARAM MQQUEMGRNAME "p101"
  PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
  PARAM MQDEFAULTPREFIX "$DEV"
  PRI 176
  PROCESS $P01S
  PROGRAM \RAPTOR.$DEV.ZMQSEXE.MQSSVR
  SECURITY "N"
  TMF ON
  VOLUME \RAPTOR.$DATA1.P101D

```

Figure 2 (Part 5 of 6). Example PATHWAY configuration

```

SERVER MQS-TCPLIS00
  PROCESSTYPE GUARDIAN
  AUTORESTART 10
  CPUS (0)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT \RAPTOR.$ZTN0.#PTY001C
  OWNER \RAPTOR.44,1
  PARAM MQQUEMGRNAME "p101"
  PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
  PARAM MQDEFAULTPREFIX "$DEV"
  PRI 175
  PROGRAM \RAPTOR.$DEV.ZMQSEXE.RUNMQLSR
  SECURITY "N"
  TMF ON
  VOLUME \RAPTOR.$DATA1.P101D

SERVER MQS-TRIGMON00
  PROCESSTYPE GUARDIAN
  AUTORESTART 10
  CPUS (0)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT \RAPTOR.$ZTN0.#PTY001C
  OWNER \RAPTOR.44,1
  PARAM MQQUEMGRNAME "p101"
  PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
  PARAM MQDEFAULTPREFIX "$DEV"
  PRI 175
  PROGRAM \RAPTOR.$DEV.ZMQSEXE.RUNMQTRM
  SECURITY "N"
  TMF ON
  VOLUME \RAPTOR.$DATA1.P101D

```

Figure 2 (Part 6 of 6). Example PATHWAY configuration

Changing the parameters of Pathway server classes

To alter the parameters of Pathway server classes:

1. Stop the queue manager by issuing the **endmqm** command. This also stops the Pathmon process.
2. Go to the subvolume *queue managerD*, which contains the PATHCTL file. For example:


```
>VOLUME $DATA2.MT01D
```
3. Start a Pathmon with the same name as the queue manager's Pathmon and with the NOWAIT option. For example:


```
>PATHMON /NAME $MT01, NOWAIT/
```
4. Start a Pathcom against the new Pathmon. For example:


```
>PATHCOM $MT01
```
5. Load the existing Pathway configuration for the queue manager by issuing the following command from the PATHCOM prompt:


```
>START PATHWAY COOL
```
6. Make the required changes using PATHCOM commands.
7. Shut down the Pathway system by issuing the following command:


```
>SHUTDOWN2
```
8. Start the queue manager using the **strmqm** command.

Adding user-defined server classes to an MQSeries pathway

You can add your own server class definitions to the MQSeries Pathway configuration using Pathcom. However, this is not recommended: servers *must* be well behaved, or **endmqm** does not function correctly. Note also that user-defined server class definitions are lost when a queue manager is deleted. To minimize inconvenience, you are recommended to create a reusable script.

Chapter 5. Managing queue managers

This chapter describes all aspects of the management of MQSeries queue managers.

The following sections are in this chapter:

- “Getting started”
- “Guidelines for creating a queue manager”
- “Volume structure” on page 70
- “Working with queue managers” on page 72
- “Managing the command server for remote administration” on page 77

Getting started

Before you use messages and queues, you must create at least one queue manager. Once you complete the installation process, you can use the MQSeries control commands to create a queue manager, create MQSeries principals, and start the queue manager.

You can then use MQSC commands to create the required default objects and system objects. Default objects form the basis of any object definitions that you make; system objects are required for queue manager operation. You must create these objects for each queue manager you create. The supplied command file `amqscoma`, when used with the `runmqsc` command, creates a set of default and system objects. See “Creating the default and system objects” on page 74 for information about running this sample.

See Chapter 4, “Using administration command sets” on page 43 for more information about commands that you can use with MQSeries for Tandem NSK, and the different methods of invoking them.

Guidelines for creating a queue manager

A queue manager manages the resources associated with it, such as the queues that it owns. A queue manager provides queuing services to applications for Message Queuing Interface (MQI) calls and commands to create, modify, display, and delete MQSeries objects. You create a queue manager using the `crtmqm` command. Here is a list of items to consider when creating a queue manager:

- Specify a unique queue manager name.
- Limit the number of queue managers.
- Specify a default queue manager.
- Specify a dead-letter queue.
- Specify a default transmission queue.
- Back up configuration files after creating a queue manager.

The tasks in this list are explained in the following sections.

- Specify the `PATHMON` process name.
- Specify the home terminal name.
- Distribute the processing load across CPUs.

Specifying a unique queue manager name

When you create a queue manager, you must ensure that no other queue manager has the same name in your network. Queue manager names are not checked at create time, and nonunique names prevent you from creating channels for distributed queuing.

You can ensure uniqueness by prefixing each queue manager name with its own node name. For example, if a node is called `accounts`, you can name your queue manager `accounts.saturn.queue.manager`, where `saturn` identifies a particular queue manager and `queue.manager` is an extension you can give to all queue managers. Alternatively, you can omit this extension; however, `accounts.saturn` and `accounts.saturn.queue.manager` are *different* queue manager names.

Note: Queue manager names in control commands are case sensitive. For example, you can create two queue managers with the names `jupiter.queue.manager` and `JUPITER.queue.manager`.

Limiting the number of queue managers

In MQSeries for Tandem NSK, you can create as many queue managers as resources allow. However, because each queue manager requires its own resources, it is often more efficient to have one queue manager with 100 queues than ten queue managers with ten queues each. Many nodes can be run with a single queue manager; however, larger servers can run with multiple queue managers. There can be special requirements of either performance, or functionality that would require multiple queue managers.

Specifying the default queue manager

Each node should have a default queue manager, though it is possible to configure MQSeries on a node without one.

To create a default queue manager, specify the `-q` flag on the `crtmqm` command. For a detailed description of this command and its parameters, see “`crtmqm` (Create queue manager)” on page 234.

What is a default queue manager?

The default queue manager is the queue manager that applications connect to if they do not specify a queue manager name in an MQCONN call. It is also the queue manager that processes MQSC commands when you invoke the `runmqsc` command without specifying a queue manager name.

How do you specify a default queue manager?

You include the `-q` flag on the `crtmqm` command to specify that the queue manager you are creating. This is the default queue manager. Omit this flag if you do not want to create a default queue manager.

Specifying a queue manager as the default *replaces* any existing default queue manager specification for the node.

What happens if I make another queue manager the default?

If you change the default queue manager it can affect other users or applications. The change has no effect on currently-connected applications because they can use the handle from their original connect call in any further MQI calls. This handle ensures that the calls are directed to the same queue manager. Any applications connecting after the change connect to the new default queue manager.

Specifying a dead-letter queue

The dead-letter queue is a local queue where messages are put if they cannot be routed to their correct destination.

Attention: You should have a dead-letter queue on each queue manager in your network. Failure to do so can result in application programs errors, which causes channels to be closed and causes replies to administration commands to fail. For example, if an application attempts to put a message on a queue on another queue manager, but the wrong queue name is given, the channel is stopped, and the message remains on the transmission queue. Other applications cannot then use this channel for their messages.

The channels are not affected if the queue managers have dead-letter queues. The undelivered message is put on the dead-letter queue at the receiving end, leaving the channel and its transmission queue available.

Therefore, when you create a queue manager you should use the `-u` flag to specify the name of the dead-letter queue. You can also use an MQSC command to alter the attributes of a queue manager and specify the dead-letter queue to be used. See “Altering queue manager attributes” on page 102 for an example of an MQSC ALTER command.

A sample dead-letter queue definition is provided with the supplied sample `amqscoma`. The queue is called `SYSTEM.DEAD.LETTER.QUEUE`. See “Creating the default and system objects” on page 74 for information about running this sample. When you find messages on a dead-letter queue, you can use the dead-letter queue handler, which is supplied with MQSeries, to process these messages. See Chapter 9, “MQSeries dead-letter queue handler” on page 157 for further information about the dead-letter queue handler, and how to reduce the number of messages that might otherwise be placed on the dead-letter queue.

Specifying a default transmission queue

A transmission queue is a local queue on which messages in transit to a remote queue manager are queued pending transmission. The default transmission queue is the queue that is used when no transmission queue is explicitly defined. Each queue manager can be assigned a default transmission queue.

When you create a queue manager you should use the `-d` flag to specify the name of the default transmission queue. The `-d` flag does not actually create the queue, which you have to create at a later time. See “Working with local queues” on page 108 for more information.

Backing up configuration files after creating a queue manager

There are two configuration files to back up, `MQSINI` and `QMINI`:

1. The MQSeries configuration file (`MQSINI`) is created when you install MQSeries. This file contains a list of queue managers that is updated each time you create or delete a queue manager. There is one `MQSINI` file per installation. By default, `MQSINI` is located in `$SYSTEM.ZMQSSYS`.
2. A queue manager configuration file (`QMINI`) is automatically created when you create a new queue manager. This file contains configuration parameters for the queue manager.

Creating queue managers

You should make a backup of these files. If you create another queue manager that causes problems, you can reinstate the backups when you have removed the source of the problem. You should back up your configuration files each time you create a new queue manager.

For more information about configuration files, see Chapter 13, “Configuration files” on page 183.

Configurable queue-manager properties

Many of the properties of a queue manager can be modified when the queue manager is created. Some properties can also be modified after the queue manager is created, though you are usually required to stop and restart the queue manager before the changes can take effect.

The remainder of this section describes some queue-manager properties that you might want to change.

Home volume of the queue manager

This is the volume where all databases, including queues, are created. (However, individual queues may be moved to a different volume after creation using the **altmqfls** control command, as described in “altmqfls (Alter queue file attributes)” on page 220.)

The default value is taken from the `QMDefaultVolume` entry of the `AllQueueManagers` stanza in the `MQSINI` file. It is overridden by the `-p DefaultPrefix` parameter of the **crtmqm** command, if specified.

The home volume can be specified *only* when a queue manager is created. It cannot be changed after creation.

Number of EC processes per queue manager

By default, there is one EC process per queue manager. You specify the number of EC processes for a queue manager on the `-e NumECs` parameter of the **crtmqm** command.

Each EC is responsible for a subset of the server processes that perform messaging and queuing for applications and channels in the same CPU as the EC itself. You are recommended to have 1 EC per CPU, unless the number of applications per EC is large, in which case having an additional EC running on the CPU would be beneficial. For large installations, for example, more EC processes are desirable (often distributed across multiple CPUs) so that large numbers of applications and channels can be handled concurrently.

During queue manager creation, a TS/MP server class is created for each EC specified on the **crtmqm** command. The specified EC server classes are distributed across all CPUs in the system, in a round-robin fashion. For example, specifying eight EC processes in a four-CPU system would result in two EC processes per CPU by default.

The `ExpectedNumECs` field of the `ECBoss` stanza in the `QMINI` file of the queue manager is set to the number of EC processes specified on creation. This value must be consistent with the TS/MP configuration at all times.

It is possible to change the number of EC processes in a queue manager after creation by adding or deleting TS/MP server classes, and modifying the ExpectedNumECs entry of the ECBoss stanza in the QMINI file. This can be done only while the queue manager is stopped.

System load balancing: The EC Boss is responsible for distributing the workload of a queue manager among the ECs. The processing load of a queue manager can be distributed among multiple CPUs in a balanced way, given an appropriate configuration of ECs.

When a new connection request arrives from a local application, or when a channel is to be started, the EC Boss allocates the request to the EC with the smallest number of active LQMA and MCAs.

Home terminal of the queue manager

All Tandem NSK processes, including the queue manager server processes, have a home terminal. The terminal *must* exist and be in the paused state. In general, the queue manager home terminal is not used for output. The home terminal can be any valid terminal device, including the Tandem Virtual Hometerm Service (VHS) product.

Tandem NonStop Kernel allows up to 255 primary openers of a physical terminal. Therefore, careful planning is required to ensure that this limit is not exceeded. You are recommended to use Tandem's VHS product if support for more than 255 openers is required.

You identify a queue manager's home terminal on the `-o HomeTerminalName` parameter of the **crtmqm** command. There is no default; this parameter is mandatory.

The HOMETERM and OUT attributes of all TS/MP server classes are set to the specified terminal device. These attributes may be altered at any time when server classes are in the stopped state, normally when the queue manager is stopped.

The HomeTerminalName entry in the Configuration stanza in the QMINI file must also be modified in order to change the home terminal of a queue manager.

The PATHMON process name for the queue manager

Each queue manager runs under its own TS/MP (Pathway) configuration. The controlling process for this is the PATHMON process. A unique name must be specified for each queue manager. Furthermore, the name must be unique within the system.

You specify the PATHMON process name on the `-n PATHMONProcessName` parameter of the **crtmqm** command. There is no default; this parameter is mandatory.

Specify a process name that is unique in the system, and is easy to associate with the queue manager it controls.

You can change the PATHMON process name for a queue manager, as follows:

1. Stop the queue manager.
2. Set your default volume and subvolume to the location of the queue manager data files (normally <QMgr name>D).

Creating queue managers

3. Modify the PathmonProcName entry in the queue manager's QMINI file to specify the new process name.

4. Run PATHMON up for the queue manager, using the *new* process name.

From TACL, execute the following command:

```
PATHMON /name $<newname>, nowait/
```

5. Run PATHCOM against the newly named PATHMON.

From TACL, execute the following command:

```
PATHCOM $<newname>
```

6. Load the queue manager Pathway configuration and confirm the change of name of the PATHMON process.

From PATHCOM, execute the following command:

```
START PATHWAY COOL
```

As the configuration is loading, you will be warned that the name of the new PATHMON process is different from the one stored in the configuration file.

After this, you will be asked to confirm whether you want to proceed. Type *y* at the prompt, and the configuration loading will complete.

7. Save the new pathway configuration information back to the database.

From PATHCOM, execute the following commands:

```
SHUTDOWN2  
EXIT
```

The PathmonProcName entry in the Configuration stanza of the QMINI file must also be changed.

The PATHMON process name change is now complete. The next **strmqm** will start the queue manager using the new PATHMON process name.

The CCSID of the Queue Manager

This is the Coded Character Set ID of the character set that is used by the queue manager to store information about messages.

You specify the CCSID on the *-l CCSID* parameter of the **crtmqm** command. The default is 819.

The CCSID of the queue manager can be changed at any time after queue manager creation using **runmqsc**, the MQM facility, or PCF commands.

The EMS Collector for the queue manager

The queue manager can be configured to use an alternative collector if required. EMS Events are sent to \$0 by default. The EMSCollectorName entry in the Configuration stanza in the QMINI file specifies the name of the EMS Collector for this queue manager.

The EMS collector can be changed at any time by modifying the value of this entry, though it does not take effect until the queue manager has been restarted.

The pool of agents kept ready by each EC in the queue manager

For each of the four basic types of agent, an EC can maintain a pool of idle agent processes, ready to be assigned to new work. The size of these pools can be configured in order to achieve an appropriate balance between response time to new work and resource utilization. The values of the following fields of the Configuration stanza in the QMINI file can be modified to specify a different number of processes to be kept idle:

MinIdleMCALU62Responders

Specifies the minimum number of SNA LU 6.2 responder MCAs to maintain in an idle state. The default value is 0.

MinIdleMCATCPResponders

Specifies the minimum number of TCP/IP responder MCAs to maintain in an idle state. The default value is 0.

MinIdleMCACallers

Specifies the minimum number of caller MCAs (not protocol specific) to maintain in an idle state. The default value is 0.

MinIdleLQMAgents

Specifies the minimum number of local queue manager agents (LQMAs) to maintain in an idle state. The default value is 1.

Note that the number of processes specified in these fields applies to each EC, not to each queue manager. Therefore, for a two-EC queue manager, there is a minimum of two idle LQMAs by default.

These values can be changed at any time, though the change does not take effect until the queue manager is restarted.

Maximum idle agents and process reuse

By default, a queue manager allows up to 10 agent processes of each type to be idle. This value is controlled by the `MaxIdleAgents` entry in the Configuration stanza of the QMINI file.

The `MaxIdleAgentReuse` entry determines the number of times an agent process can be reused before it is replaced by a new agent process. By default, `MaxIdleAgentReuse` is set to 10.

Process priority of queue manager processes

The priorities may need to be changed to balance resources between MQSeries and other applications. The process priorities of the TS/MP server classes may be changed by ALTERing the TS/MP objects when the queue manager is stopped.

The process priorities of the agent processes may be changed by editing the `MCAAgentPriority` and `LQMAgentPriority` fields of the EC stanza of the QMINI file.

Apart from the MQSS Servers, which have a default process priority of 176, the TS/MP configured processes are all given a default priority of 175. By default, both MCAs and LQMAs have a process priority of 165.

Ensure that the MQSS Servers have the highest priority, followed by the EC Boss and EC, which in turn must have a higher process priority than the MCAs and LQMAs.

Maximum number of channels for the queue manager

There is a limit to the number of channels that may be controlled at any one time for a queue manager. If the limit is too high, performance may be affected as this parameter dictates the size of the channel status table, on which numerous search operations are performed. If the limit is too low, you may not be able to control enough channels for your application. The `MaxChannels` field of the `Channels` stanza in the QMINI file defines the maximum number of channels that can be controlled simultaneously.

The default on creation is 10. There is no way to override the default on creation.

The `MaxChannels` entry in the `Channels` stanza of the QMINI file can be changed at any time, though the change does not take effect until the queue manager is restarted.

Maximum number of active channels for the queue manager

There is a limit to the number of concurrently active (running) channels in a queue manager. This may be used to control the peak demand on system resources by channels. The `MaxActiveChannels` entry in the `Channels` stanza in the QMINI file defines the maximum number of active channels for the Queue Manager.

The default on creation is 10. There is no way to override the default on creation.

The `MaxActiveChannels` entry in the QMINI file can be changed at any time, though the change does not take effect until the queue manager is restarted.

Channel initiator disconnect interval

This parameter determines the frequency of waited gets on the `SYSTEM.CHANNEL.INITQ` or whichever initiation queue the channel initiator is processing. Its default value is 10 seconds. Also, under normal channel operating conditions, it determines the interval between scans of the channel status table, which is used to establish when a channel needs to be started. Once it is decided that a channel requires action, a time is calculated for the next scan. If this parameter is made too high, although CPU usage will decrease, channel triggering will not occur as quickly, so a compromise must be reached.

Default TCP/IP port

The `TCPPort` entry in the `TCPCConfig` stanza in QMINI defines the default port number for outgoing channels. By default, port number 1414 is used. This default is overridden by port-number values specified in the `CONNNAME` field for a channel.

TCP/IP ports listened on by the queue manager

A queue manager with TCP/IP channels may be configured to listen for incoming connections on one or more TCP/IP ports. The `TCPCNumListenerPorts` and `TCPCListenerPort` entries in the `TCPCConfig` stanza in the QMINI file define how many ports to listen on, and the numbers of the ports assigned to this queue manager. For examples of the QMINI entries, see "TCP/IP communications example" on page 345.

There can be multiple queue managers on a single system. Each queue manager on a system must be assigned nonoverlapping sets of TCP/IP ports to listen on. The set of TCP/IP ports for each queue manager may be just one port, where the rate of incoming TCP/IP connect requests is low, or may be more than one port for

large configurations. The default TCP/IP port is 1414 and, by default, a queue manager is created to listen on only this port.

The list of listening ports may be changed by editing the `TCPConfig` stanza in the QMINI file and restarting the queue manager. In order to listen on more than one port, a queue manager must also be configured with additional TCP/IP listener server classes using TS/MP. This operation is performed manually using PATHCOM. Alternatively, a port number can be specified on the `runmqtsr` command (described in “runmqtsr (Run listener)” on page 260).

TCP/IP process used by the queue manager

The interface to the Tandem TCP/IP product is via a server process, known as the TCP/IP process. A queue manager can be configured to use a specific TCP/IP process, if the system default is not sufficient. The TS/MP server class configuration must be manually changed if the system default TCP/IP process, `$ZTC0`, is not sufficient or correct.

Server classes `MQS-TCPLIS00` and `MQS-EC00` must have the `DEFINE TCPIP^PROCESS^NAME` added to reference the required alternative TCP/IP process name. Note that, if you have multiple ECs, you must update all of them. Refer to the Tandem NSK TCP/IP product manuals for further information.

By default, the system default, `$ZTC0`, is used. There is no way to override this default on creation.

The change to the TS/MP server classes can be made only when the queue manager is stopped.

MQSeries for Tandem NSK can support only one TCP/IP process per queue manager. If multiple TCP/IP processes are required, multiple queue managers must be used. (Note, however, that multiple TCP/IP ports and listener processes are supported.)

Reconfiguring a queue manager for a nondefault TCP/IP process

Follow the general instructions in “Changing the parameters of Pathway server classes” on page 57. To specify a nondefault TCP/IP process, use the following Pathcom commands:

```
ALTER SERVER mqs-ec00, (define=TCPIP^PROCESS^NAME, FILE $ZZZZ)
ALTER SERVER mqs-tcp1is00, (define=TCPIP^PROCESS^NAME, FILE $ZZZZ)
```

where `$ZZZZ` is the name of your TCP/IP process.

Swap space allocation

MQSeries for Tandem NSK allocates swap space according to the `ExtPoolSize` values for the various executables in the QMINI configuration file. Therefore, if your queue manager is using the default QMINI file, which allocates 5 MB to each executable by default, and is running 10 outbound channels, 20 agents, and 10 TCP responders, you need at least 200 MB (40 * 5 MB) of swap space. To reduce this requirement, you can lower the values in the QMINI file. The `ExtPoolSize` values are the minimum additional memory allocated when the initial memory allocation is exhausted. The value does not have to be larger than the maximum message size for the queue manager.

Default MQSS Server name

A unique process name must be specified for the default MQSS Server process pair when you create a queue manager. You specify the default MQSS Server name on the mandatory `-s MQSServerName` parameter of the `crtmqm` command. There is no default value for this parameter. Specify a process name that is both unique in the system and easy to associate with the queue manager to which it belongs.

You can change the default MQSS Server process name for a queue manager as follows:

1. Stop the queue manager.
2. Set your default volume and subvolume to the location of the queue manager data files (usually `<Qmgrname>D`).
3. Edit the `DefaultMQSSName` entry in the Configuration stanza of the queue manager's QMINI file to record the new process name.
4. Run PATHMON for the queue manager by entering the following command from TACL:

```
PATHMON /name $<pmon>, nowait/
```

where `<pmon>` is the name of the PATHMON process for the queue manager.

5. Run PATHCOM against the PATHMON process by entering the following command from TACL:

```
PATHCOM $<pmon>
```

6. Load the queue manager Pathway configuration by entering the following command from PATHCOM:

```
START PATHWAY COOL
```

7. Modify the process name of the default MQSS Server by entering the following commands from PATHCOM:

```
ALTER SERVER MQS-STATUS00, DELETE PROCESS $<oldname>  
ALTER SERVER MQS-STATUS00, PROCESS $<newname>
```

8. Save the altered configuration back to disk by entering the following commands from PATHCOM:

```
SHUTDOWN2  
EXIT
```

The default MQSS Server process name change is now complete. The next `strmqm` command starts the queue manager using the new default MQSS Server process name.

Adding and removing nondefault MQSS Servers

To add additional MQSS Servers, use the existing default MQSS Server as a template. The queue manager does not need to be stopped to allow you to add a new MQSS Server.

The following procedure adds a new MQSS Server

Note: This procedure assumes that the queue manager is running. If this is not the case, you must start Pathmon and load the pathway configuration before starting this procedure. You must also omit step 5 on page 69, and save the Pathway configuration to disk at the end using a SHUTDOWN2 command.

1. Run PATHCOM against the PATHMON process by entering the following command from TACL:

```
PATHCOM $<pmon>
```

2. Create a working set of attributes based on the default MQSS Server class as a template by entering the following commands from PATHCOM:

```
RESET SERVER
SET LIKE MQS-STATUS00
```

3. Modify the working set for the new server by entering the following commands from PATHCOM:

```
SET SERVER CPUS(n:m)
RESET SERVER PROCESS $<default MQSS Server process name>
SET SERVER PROCESS $<new MQSS Server name>
```

4. Add the new server, giving it a new server class name by entering the following command from PATHCOM:

```
ADD SERVER MQS-STATUSxx
```

5. Start the new MQSS Server, so that it can be used, and exit from PATHCOM by entering the following commands:

```
START SERVER MQS-STATUSxx
EXIT
```

The recommended naming convention for additional server classes is MQS-STATUS01, MQS-STATUS02, and so on. However, there is no requirement to use this convention. Provided that the server class name begins with the character string MQS-STATUS, the server class will be started by **strmqm**.

Once an MQSS Server has been added and started (either explicitly using PATHCOM or implicitly by **strmqm**), objects can be reassigned to the new MQSS Server using **altmqfls**. For more information about reassigning objects, see “Reassignment of objects to MQSS Servers” on page 113.

Before removing an MQSS Server, check that all objects configured against this MQSS Server have been either deleted or reassigned to a different MQSS Server. You must **not** delete the default MQSS Server, otherwise the queue manager will become inoperable.

Volume structure

Files for MQSeries for Tandem NSK are distributed over several subvolumes. The volume in which these subvolumes reside is selected when you create the queue manager: it is either taken from the default volume value in MQSINI or specified on the `-p DefaultPrefix` parameter of the **crtmqm** command.

There are five subvolumes per queue manager. The contents of the subvolumes are determined by the final character of the subvolume name. For example, for a queue manager called QMGR resident on a volume \$DATA, the following subvolumes would be present:

\$DATA.QMGR	FFST™ subvolume
\$DATA.QMGRD	Queue manager data files subvolume
\$DATA.QMGRLE	Queue manager error logs subvolume
\$DATA.QMGRM	Message queue subvolume
\$DATA.QMGRS	Channel synchronization subvolume

If the queue manager name is more than seven characters, the subvolume names are transformed or shortened. The MQSINI file stanzas `QMVolume` and `QMSubVolume` for the queue manager are used to record the locations and names of these subvolumes.

Queue manager FFST subvolume

The FFST subvolume contains first failure support files. These files are all prefixed with the letters FD. They indicate serious problems with the MQSeries system, such as resource shortage, internal MQSeries errors, or problems with the NSK system.

Queue manager data files subvolume

CCHDEFS	Is the client connection channel definition file.
CCHDEFS0	Is the alternate key file for CCHDEFS.
CHDEFS	Is the channel definition file. This file contains configuration information for the channels that are defined for a queue manager.
CHDEFS0	Is the alternate key file for CHDEFS.
OAMDB	Is the authorization database.
OBJCAT	Is the object catalog. ABJCAT is the key file that stores indexes.
PATHCTL	Is the PATHWAY control file.
PRIDB	Is the MQSeries principal (user) database.
PRIDBA	Is the alternate key file for PRIDB.
QMINI	Is the queue manager initialization file.
RDFPURGE	Is the database used by the queue manager to record logically deleted files that will eventually be removed by cleanrdf .
RUNTIME	Is the file for ECBOSS and EC recovery coordination.
SHUTDOWN	Is the file that controls endmqm .
STATABLE	Is the channel status table file. This file holds dynamic information associated with channel status.

	STATABLO	Is the alternate key file for STATABLE.
	TRACEOPT	The TRACEOPT file contains the current trace settings for a queue manager in the form of an unformatted bit-map record. The control commands strmqtrc and endmqtrc modify the contents of the file, using the CONTROL 27 mechanism to notify all processes of the update.
	Txxxxxxx	Are the Touch files. If an object is altered, the Touch files change the object date stamp. Txxxxxxx is derived from the object name; otherwise it is a generated value.
	UMQSINI	Is a snapshot of the unstructured MQSINI file at queue manager startup.
	UQMINI	Is the unstructured version of QMINI.

Queue manager error log subvolume

The error log subvolume contains the error and trace logging files. The TR prefix identifies trace files. (You can change the prefix by editing the TracePrefix entry in the QMINI file.) Trace files contain diagnostic information, and are created only if tracing is switched on using either the MQM facility or the **strmqtrc** control command.

The error logs have names in the format MQERRLGn, where n is 1, 2, or 3. MQERRLG1 is always the current error log. Its contents are moved to MQERRLG2 when MQERRLG1 is full; MQERRLG2 is moved to MQERRLG3 when MQERRLG1 is next emptied. MQERRLG3 is overwritten if necessary. There are never more than three error logs, so they must be sized correctly to avoid loss of useful error information.

Queue manager message queue subvolume

The message subvolume contains queue files and queue overflow files. The file names are in the following format:

Qxxxxxxx	Is the queue file. Files starting with the letter A, B, or C and having the same xxxxxxx suffix as the queue file are alternate key files belonging to the queue file Qxxxxxxx:
Axxxxxxx	Is the alternate key file supporting retrieval by <i>Msgld</i> .
Bxxxxxxx	Is the alternate key file supporting retrieval by <i>Correlld</i> .
Cxxxxxxx	Is the alternate key file supporting retrieval by <i>Msgld</i> and <i>Correlld</i> .
	All three alternate key files are present for local queues whose retrieval criteria have not been modified using altmqfls . Any combination of these files (including absence of all three) is valid.
Oxxxxxxx	Is the overflow file.
xxxxxxx	May be the queue name if it is a unique, short name; otherwise it is a generated value. (See "Object name transformation" on page 72.)

Queue manager channel synchronization subvolume

The queue manager synchronization subvolume contains internal databases that record the status of units of work (or batches of messages) transmitted or received over the channels that are owned by the queue manager.

Once channels have been used on a queue manager, the subvolume contains the following files:

SYNCHIDX (file code 0)

The synchronization index file. Contains an entry for each synchronization file created by the queue manager.

Sxxxxxxx (file code 0)

Individual synchronization files. There is one file for each unique combination of local and remote channel that has been used in the queue manager. These files record the identities of the messages that have been transmitted or received within a batch of messages. The information is used in the resynchronization of channels following failure and the resolution of in-doubt channels.

Object name transformation

Object names are not necessarily valid file system names. Therefore, the object names might need to be transformed. The method used is different from that used for queue-manager names because, although there may be only a few queue-manager names per system, there can be a large number of other objects for each queue manager. Only process definitions and queues are represented in the file system; channels are not affected by these considerations. Queues have five files, starting with Q, A, B, C, and O respectively, as described in “Queue manager message queue subvolume” on page 71.

When a new name is generated by the transformation process there is no relationship with the original object name. You can use the **dspmqls** command to convert between real and transformed object names: **dspmqls** displays the names of the main files associated with an MQSeries object.

Working with queue managers

MQSeries provides control commands for creating, starting, ending, and deleting queue managers. You can also display a queue manager's attributes using the MQSC command DISPLAY QMGR and change them using ALTER QMGR. See “Displaying queue manager attributes” on page 101 and “Altering queue manager attributes” on page 102.

Ensure that the environment variable PMSEARCHLIST (described in “TACL environment variables” on page 26) specifies the location of your MQSeries executables before you attempt to use the control commands.

Creating a default queue manager

The following command creates a default queue manager called saturn.queue.mgr and specifies the names of both its default transmission queue and its dead-letter queue:

```
crtmqm -q -d MY.DEF.XMIT.Q -u SYSTEM.DEAD.LETTER.QUEUE -n $PMON -o $TRM0 -s $MQSS saturn.queue.mgr
```

where:

-q Indicates that this queue manager is the default queue manager.

-d MY.DEF.XMIT.Q
is the name of the default transmission queue.

-u SYSTEM.DEAD.LETTER.QUEUE
Is the name of the dead-letter queue.

-n \$PMON
Is the process name of PATHMON for the queue manager.

-o \$TRM01
Is the home terminal name (must be paused).

-s \$MQSS
Is the process name of the default MQSS Server.

saturn.queue.mgr
Is the name of this queue manager. For **crtmqm**, this name must be the last parameter in the command.

Creating MQSeries principals

The **crtmqm** command automatically creates a principal for the user that created the queue manager. This principal (also known as the default principal) is always called **mqm** for compatibility with other MQSeries implementations.

Once you have created a queue manager you may define principals for other users of MQSeries. This step may be performed at any time (whether or not the queue manager has been started). If no other users are required for the queue manager, this step can be omitted.

To create an MQSeries principal named **MQPRINCIPAL** corresponding to Tandem NSK user **MQM.MQUSER**, enter the command:

```
altnqusr -m saturn.queue.manager -p MQPRINCIPAL -u MQM.MQUSER
```

To display all the principals currently created, enter the command:

```
dspmqusr -m saturn.queue.manager
```

Remember that if you do not create a principal entry for a user, any attempt to access the queue manager by that user (whether the OAM is enabled or not) will result in an authorization error. This is a key difference between MQSeries for Tandem NSK V2.2 and V2.2.0.1.

Running cleanrdf for an RDF-enabled queue manager

If you are running a queue manager in the RDF environment and have enabled RDF-specific behavior using the **MQRDF PARAM**, you should run the **cleanrdf** utility periodically, as follows:

- After making any configuration changes (such as creating or deleting objects, or making changes to the **QMINI** file), **cleanrdf** should be run.

Working with queue managers

- If your application creates and deletes objects as part of its normal operation, especially if it uses dynamic queues, **cleanrdf** should be run during normal operation at a frequency depending upon the rate of object deletion. NetBatch or other scheduling software should be used.

Starting a queue manager

Although you have created a queue manager, it cannot process commands or MQI calls until it is started. Start the queue manager by entering this command:

```
strmqm saturn.queue.manager
```

The **strmqm** command does not return control until the queue manager has started and is ready to accept connect requests.

Creating the default and system objects

You must create a set of default and system objects for each queue manager you create by using the **runmqsc** command to specify both the name of the queue manager and the name of the command file containing the commands. (You can specify `amqscoma`, which is supplied with MQSeries for Tandem NSK and resides in the `ZMQSSMPL` subvolume.) The following command creates the default and system objects:

```
runmqsc -i $SYSTEM.ZMQSSMPL.AMQSCOMA -o defobj QMNAME
```

You can run this command immediately after the **strmqm** command has completed.

The file `defobj` is created, if it does not already exist. When the command has completed, `defobj` contains the output from the MQSC file. You should check that all the commands ran successfully before continuing.

For more information about running the MQSC facility (**runmqsc**), see “Running MQSCs from text files” on page 102.

Looking at object files

Each MQSeries queue, queue manager, or process object is represented by a file. Because the names of these objects are not necessarily valid file names, the queue manager converts the object name into a valid file name, where necessary. This process is described in “Object name transformation” on page 72.

Stopping a queue manager

To stop a queue manager, use the **endmqm** command. For example, to stop a queue manager called `saturn.queue.manager` use this command:

```
endmqm saturn.queue.manager
```


By default, this command performs a *controlled* or *quiesced* shutdown of the specified queue manager. This process might take a while to complete—a controlled shutdown waits until *all* connected applications have disconnected and until all running channels have stopped.

“Immediate and preemptive queue manager shutdowns” describes optional flags for the **endmqm** command that specify how the shutdown is to be carried out.

If you have problems

Problems in shutting down a queue manager are often caused by applications. For example, when applications:

- Do not check MQI return codes properly.
- Do not request a notification of a quiesce.

Immediate and preemptive queue manager shutdowns

If a shutdown of a queue manager is slow, or the queue manager does not stop, you can terminate the **endmqm** command using **BREAK** followed by **STOP**. You can then issue another **endmqm** command, but this time with a flag specifying either an immediate or a preemptive shutdown.

For an *immediate shutdown* any current MQI calls are allowed to complete, but any new calls fail. This type of shutdown does not wait for applications to disconnect from the queue manager. For an immediate shutdown, the command is:

```
endmqm -i saturn.queue.manager
```

If an immediate shutdown does not work, try a *preemptive* shutdown by specifying the **-p** flag. For example:

```
endmqm -p saturn.queue.manager
```

Attention: Do not use this method unless all other attempts to stop the queue manager using the **endmqm** command have failed. This method can have unpredictable consequences for connected applications.

If this method still does not work, see “Stopping a queue manager manually” on page 295 for an alternative.

For a detailed description of the **endmqm** command and its options, see “endmqm (End queue manager)” on page 253.

Restarting a queue manager

To restart a queue manager, use the command:

```
strmqm saturn.queue.manager
```

Making an existing queue manager the default

When you create a default queue manager, the name of the default queue manager is inserted in the *DefaultQueueManager* stanza in the MQSeries configuration file (MQSINI). The stanza and its contents are automatically created if they do not exist.

You might need to edit this stanza:

- To make an existing queue manager the default. To perform this task you have to change the queue manager name in this stanza to the name of the new default queue manager. You must perform this step manually using a text editor.
- If you do not have a default queue manager on the node, and you want to make an existing queue manager the default. To perform this task, you must create the *DefaultQueueManager* stanza—with the required name—yourself.
- If you accidentally make another queue manager the default and want to revert to the original default queue manager. To perform this task, edit the *DefaultQueueManager* stanza in the MQSeries configuration file, replacing the name of the unwanted default queue manager with that of the one you do want.

See Chapter 13, “Configuration files” on page 183 for information about configuration files.

When the stanza contains the required information, stop the queue manager and restart it.

Deleting a queue manager

To delete a queue manager, first stop it, then use the following command:

```
dltmqm saturn.queue.manager
```

Attention: Use caution if deleting a queue manager as you also delete all the resources associated with it, including all queues and their messages and all object definitions. Also, all files in the queue manager subvolumes might be purged (even if they were not created by MQSeries).

For a description of the **dltmqm** command and its options, see “dltmqm (Delete queue manager)” on page 238. You should ensure that only trusted administrators have the authority to use this command.

If the usual methods for deleting a queue manager do not work, see “Removing queue managers manually” on page 295 for an alternative.

Managing the command server for remote administration

Each queue manager has a command server associated with it. A command server processes any incoming commands from remote queue managers or PCF commands from applications. It presents the commands to the queue manager for processing and returns a completion code or operator message depending on the origin of the command. There are separate control commands for starting and stopping the command server.

Note: For remote administration, you must ensure that the target queue manager is running. Otherwise, the messages containing commands cannot leave the queue manager from which they are issued. Instead, these messages are queued in the local transmission queue that serves the remote queue manager.

Starting the command server

To start the default command server for queue manager `saturn.queue.manager`, use this command:

```
strmqcsv saturn.queue.manager
```

Displaying the status of the command server

For remote administration, you must ensure that the command server on the target queue manager is running. If it is not running, no remote commands can be processed. Any messages containing commands are queued in the target queue manager's command queue.

To display the status of the command server for a local queue manager, called here `saturn.queue.manager`, the command is:

```
dspmqcsv saturn.queue.manager
```

Stopping a command server

To end a command server, the command, using the previous example, is:

```
endmqcsv saturn.queue.manager
```

Using the Message Queue Management (MQM) facility

The Message Queue Management (MQM) facility of MQSeries for Tandem NSK V2.2.0.1 runs as a PATHWAY SCOBOL requester under the Terminal Control Process (TCP). It uses an MQM SERVERCLASS server, which invokes the C language API.

There is a separate instance of the MQM for each queue manager configured on a system, because each queue manager is controlled under its own PATHWAY configuration. Consequently, MQM is limited to the management of the queue manager to which it belongs.

Note: By default, a maximum of 10 users may use the MQM facility concurrently. To change this limit to 20, for example, enter:

```
alter tcp mqs-tcp-01, maxterms 20
```

from the PATHWAY of the queue manager. For more information, see Chapter 4, "Using administration command sets" on page 43.

To invoke MQM, enter `run mqmc` from the queue manager's PATHCOM prompt.

The MQM Main Menu is as follows:

```
IBM MQSeries for Tandem NonStop Kernel Version 2

      ** Main Menu **

Enter Choice:  _

1. Queue Manager
2. Queues
3. Channels

F1 - Enter                                     F16 - Return

83H8731,5697-A17 (C) Copyright IBM Corp. 1993, 1997 All Rights Reserved.
```

Figure 3. The MQSeries for Tandem NSK MQM Main Menu

You can select the following submenus from the MQM Main Menu:

1. Queue Manager
2. Queues
3. Channels

These submenus are described in the remainder of this chapter. You can return to the MQM Main Menu at any time by pressing Alt+F6. You can return to the previous screen by pressing the Return key (F16). When selected from the MQM Main Menu, F16 exits from the MQM facility.

Using the Queue Manager Menu

To select the Queue Manager option, type 1 in the Enter Choice field on the MQM Main Menu, then press the Enter key (F1). The Queue Manager Menu panel is displayed:

```

                                IBM MQSeries for Tandem NonStop Kernel Version 2

                                ** Queue Manager Menu **

Name                            : MT01
Description                      : _____

Command Level                    :      220  Trigger Interval   : 999999999
Coded Char Set                  :      819  Platform           : TANDEM_
Max Handles                     :      256  Max Uncommitted Msg:  10000
Max Message                     : 4194304  Max Priority        :      9
Dead Letter Queue Name          : SYSTEM.DEAD.LETTER.QUEUE_
Command Input Queue Name        : SYSTEM.ADMIN.COMMAND.QUEUE_
Default Xmit Queue Name         : _____

Authority Event Enabled Y/N?    : N                Inhibit Event Enabled Y/N?    : N
Local Event Enabled Y/N?       : N                Remote Event Enabled Y/N?     : N
Start/Stop Event Enabled Y/N?: N                Performance Event Enabled Y/N?: N

                                FORCE   Y/N?  _

F1 - Modify   F2 - Trace                                F16 - Return
  
```

Figure 4. The Queue Manager Menu panel

You can use the Queue Manager Menu panel to:

- Alter some attributes of the queue manager
- Control tracing of MQSeries objects

Altering Queue Manager attributes

Overtyping those values you want to alter on the Queue Manager Menu panel, and press the Modify key (F1). You are prevented from overtyping those values that cannot be modified.

Tracing MQSeries objects

Press the Trace key (F2) to display the QUEUE MANAGER TRACE MENU:

```
IBM MQSeries for Tandem NonStop Kernel Version 2

** QUEUE MANAGER TRACE MENU **

_ API      : MQI.
_ COMMS    : Communications networks processing flow.
_ CSFLOWS  : Common services processing flow.
_ LQMFLows : Local queue manager processing flow.
_ REMOTEFLows : Communications component processing flow.
_ ADMINFLow : Administrative processing flow.
_ OTHERFLows : Other components processing flow.
_ CSData   : Common services data buffers.
_ LQMData  : Local queue manager internal data buffers.
_ REMOTEData : Communications component internal data buffers.
_ ADMINData : Administrative internal data buffers.
_ OTHERData : Other components internal data buffers.
_ VERSIONData : Output version of MQSeries running.
_ COMMENTARY : Output program comments in the MQSeries components.

_ All      : Select all options.

F1-Start Trace  F2-Stop Trace                      F16-Return
```

Figure 5. The QUEUE MANAGER TRACE MENU

The following trace options are available:

- | | |
|--------------------|---|
| API | Output data for trace points associated with the MQI and major queue manager components. |
| COMMS | Output data for trace points associated with data flowing over communications networks. |
| CSFLOWS | Output data for trace points associated with processing flow in common services. |
| LQMFLows | Output data for trace points associated with processing flow in the local queue manager. |
| REMOTEFLows | Output data for trace points associated with processing flow in the communications component. |
| ADMINFLow | Output data for trace points associated with administrative internal data buffers. |
| OTHERFLows | Output data for trace points associated with other components' processing flow. |
| CSData | Output data for trace points associated with internal data buffers in common services. |
| LQMData | Output data for trace points associated with internal data buffers in the local queue manager. |
| REMOTEData | Output data for trace points associated with internal data buffers in the communications component. |

ADMINDATA	Output data for trace points associated with internal data buffers in the communications component.
OTHERDATA	Output data for trace points associated with other components' internal data buffers.
VERSIONDATA	Output data for trace points associated with the version of MQSeries that is running.
COMMENTARY	Output data for trace points associated with comments in the MQSeries components.
ALL	Trace points are enabled and a full trace is generated.

Type any character against the names of the components for which you want to start (or stop) tracing.

To start tracing of the selected components, press the Start Trace key (F1). To stop tracing of the selected components, press the Stop Trace key (F2).

Using the Queues menu

To select the Queues option, type 2 in the Enter Choice field on the MQM Main Menu, then press the Enter key (F1). The Search Criteria panel is displayed:

```
IBM MQSeries for Tandem NonStop Kernel Version 2
** Search Criteria **

Queue Name: _____
Enter a queue name or part of one:

Queue Type: _
choose one or leave blank:      1. Local
                                2. Model
                                3. Remote
                                4. Alias

F1 - Enter                      F16 - Return
```

Figure 6. The Search Criteria panel (queue)

In the Queue Name field of the Search Criteria panel, type a partial or complete queue name. You may also provide a Queue Type identifier if you wish to limit your search to queues of one type. Press the Enter key (F1). The Queue menu, which you use to display, modify, create, copy, delete, and monitor MQSeries queues, is displayed.

Message Queue Management (MQM)

```
IBM MQSeries for Tandem NonStop Kernel Version 2
** Queue Menu **

Queue Name                                     Type
- ANNE.ET01.RQSD.LOCAL                        QLOCAL
- ANNE.ET01.RQSD.REMOTE                       QREMOTE
- ANNE.ET01.RQSV.LOCAL                       QLOCAL
- ANNE.ET01.RQSV.REMOTE                      QREMOTE
- ANNE.ET01.SDRCL.LOCAL                      QLOCAL
- ANNE.ET01.SDRCL.REMOTE                     QREMOTE
- ANNE.M401.RQSD.LOCAL                       QLOCAL
- ANNE.M401.RQSD.REMOTE                      QREMOTE
- ANNE.M401.RQSV.LOCAL                       QLOCAL
- ANNE.M401.RQSV.REMOTE                      QREMOTE
- ANNE.M401.SDRCL.LOCAL                      QLOCAL
- ANNE.M401.SDRCL.REMOTE                     QREMOTE

F1 - Enter/Display/Modify      F2 - Create      F3 - Copy      F4 - Delete
F5 - Monitor                   PGDN            PGUP           F16 - Return
```

Figure 7. The Queue Menu

Note: You can create, modify, and delete queues only on the queue manager associated with the MQM requester that you are using.

Use the PGUP and PGDN keys to scroll the list of queues.

Creating a queue

From the Queue Menu, press the Create key (F2) to display the Create Queue panel:

```
IBM MQSeries for Tandem NonStop Kernel Version 2
** Create Queue **

Queue Type: _  1=Local, 2=Model, 3=Remote, 4=Alias,

Name: _____

Replace [Y/N]: _

F1 - Enter                                     F16 - Return
```

Figure 8. The Create Queue panel

To create a new queue definition:

1. Type 1 (for a local queue), 2 (for a model queue), 3 (for a remote queue), or 4 (for an alias queue) in the Queue Type field.
2. Type the queue manager name in the Name field.
3. If the queue is to replace an existing queue of the same name and type, type Y in the Replace field.
4. Press the Enter key (F1).

If you create a local queue, the Create Local Queue panel is displayed:

Message Queue Management (MQM)

```
IBM MQSeries for Tandem NonStop Kernel Version 2
** Create Local Queue **

Queue Name : TEST
Description: _____
Default Msg Priority : 0          Put Enabled [Y/N]      : _
Default Persistency : _          Get Enabled [Y/N]   : _

Retention Interval  :          0      Queue Definition Type : _____
Max Queue Depth    :          0      Priority/FIFO [P/F]   : _
Max Message Length :          0      Share [Y/N]          : _
Backout Threshold  :          0      Usage [N/X]          : _
Backout Requeue Name : _____
Init. Queue       : _____
Process Name      : _____
Trigger Type [N/E/F/D]: _          Trigger/NoTrigger [Y/N] : _
Trigger Depth     :          0      Trigger Priority      : 0
Trig. Data       : _____
Q Depth Max Event :          _      Q Serv. Int. Event[H/O/N]: _
Q Depth High Limit :          0      Q Depth High Event    : _
Q Depth Low Limit  :          0      Q Depth Low Event     : _
Q Service Interval :          0      Scope                  : _____
F1 - Enter                                             F16 - Return
```

Figure 9. The Create Local Queue panel

Complete the panel, and press the Enter key (F1).

If you create a remote queue, the Create Remote Queue panel is displayed:

```
IBM MQSeries for Tandem NonStop Kernel Version 2
** Create Remote Queue **

Queue Name : TEST_REMOTE
Description: _____
Default Msg Priority : 0          Put Enabled [Y/N]      : _
Default Persistency : _

Scope      : _____
Remote Queue Name : _____
Remote Queue Manager : _____
Transmit Queue Name : _____

F1 - Enter                                             F16 - Return
```

Figure 10. The Create Remote Queue panel

Complete the panel, and press the Enter key (F1).

Copying a queue

From the Queue Menu, press the Copy key (F3) to define a new queue by copying an existing definition. The Copy Queue panel is displayed:

```
IBM MQSeries for Tandem NonStop Kernel Version 2
** Copy Queue **

Name: ANNE.ET01.RQSD.LOCAL.2_____
Replace [Y/N]: _
Like Queue: ANNE.ET01.RQSD.LOCAL_____
Queue Type: QLOCAL__

F1 - Enter                                F16 - Return
```

Figure 11. The Copy Queue panel

Type the name of the new queue definition in the Name field; type Y in the Replace field if the new queue is to replace an existing queue of the same name and type; type the name of the definition you are copying in the Like Queue field; type the queue type in the Queue Type field. Press the Enter key (F1).

Modifying a queue

From the Queue Menu, press the Modify key (F1) to display the Display/Modify Local Queue panel:

```
IBM MQSeries for Tandem NonStop Kernel Version 2
** Display/Modify Local Queue **
Queue Name : ANNE.ET01.RQSD.LOCAL
Description: Local queue ET01 receiver
-----
Default Msg Priority : 0          Put Enabled [Y/N]      : Y
Default Persistency : N          Get Enabled [Y/N]     : Y

Retention Interval  : 999999999  Queue Definition Type : PREDEFINED
Max Queue Depth    :      5000    Priority/FIFO [P/F]   : P
Max Message Length :      1024    Share [Y/N]          : Y
Backout Threshold  :           0  Usage [N/X]           : N
Backout Requeue Name : _____
Init. Queue        : _____
Process Name       : _____
Trigger Type [N/E/F/D]: F        Trigger/NoTrigger [Y/N] : N
Trigger Depth      :           1  Trigger Priority        : 0
Trig. Data : _____
-----
Q Depth Max Event  : Y          Q Serv. Int. Event[H/O/N]: N
Q Depth High Limit :           80 Q Depth High Event      : N
Q Depth Low Limit  :           20 Q Depth Low Event       : N
Q Service Interval : 999999999  Scope                   : QMGR
F1 - Modify                          F16 - Return
```

Figure 12. The Display/Modify Local Queue panel

Overtyping those values you want to modify, and press the Modify key (F1). You are prevented from overtyping those values that cannot be modified.

Deleting a queue

On the Queue Menu, enter any character against the name of the queue that you want to delete. Press the Delete key (F4), then press F4 again to confirm deletion.

Monitoring a queue

Press the Monitor key (F5) from the Queue Menu to display the Monitor Local Queues panel:

```

          IBM MQSeries for Tandem NonStop Kernel Version 2
          ** Monitor Local Queues **
Queue      OPEN INPUT  OPEN OUTPUT  DEPTH
-----
ANNE_M401_RQSD_LOCAL
ANNE_M401_RQSV_LOCAL
ANNE_M401_SDRG_LOCAL
ANNE_MA02_RQSD_LOCAL
ANNE_MA02_RQSV_LOCAL
ANNE_MA02_SDRG_LOCAL
ANNE_MD01_RQSD_LOCAL
ANNE_MD01_RQSV_LOCAL
ANNE_MD01_SDRG_LOCAL
ANNE_MD01_SVRC_LOCAL
ANNE_ME02_RQSD_LOCAL
ANNE_ME02_RQSV_LOCAL
ANNE_ME02_SDRG_LOCAL      10
ANNE_ME02_SVRC_LOCAL

F12 - Refresh          PGDN          PGUP          F16 - Return
    
```

Figure 13. The Monitor Local Queues panel

In this example, the queues are open neither for input nor for output. One queue, ANNE_ME02_SDRG_LOCAL, contains 10 messages.

The MQMQMREFRESHINT pathway parameter for MQS-MQMSVR00 determines the frequency with which monitor screens for channels and queues are refreshed. The default frequency is every 30 seconds. To change the frequency to every 10 seconds, for example, enter:

```
alter server mqs-mqmsvr00, param mqmqmrefreshint 10
```

from the pathway for your queue manager.

Using the Channels menu

To select the Channels option, type 3 in the Enter Choice field on the MQM Main Menu, then press the Enter key (F1). The channel Search Criteria panel is displayed:

Message Queue Management (MQM)

```
IBM MQSeries for Tandem NonStop Kernel Version 2
** Search Criteria **

Channel Name: _____
Enter a channel name or part of one:

Channel Type: _____
choose one or Leave blank:      1. Sender
                                   2. Server
                                   3. Receiver
                                   4. Requester
                                   5. SvrConn

F1 - Enter                                F16 - Return
```

Figure 14. The Search Criteria panel (channel)

In the Channel Name field, type a partial or complete channel name. In the Channel Type field, you may enter a number between 1 and 5 to identify the type of channel you are interested in. Press the Enter key (F1) to display the Channel Menu:

```
IBM MQSeries for Tandem NonStop Kernel Version 2
** Channel Menu **

Channel Name      TYPE      STATUS
- MA02.MT01.SDRC.0001    RECEIVER
- MA02_MT01_RQSD_0001    REQUESTER
- MA02_MT01_RQSV_0001    REQUESTER
- MA02_MT01_SDRC_0001    RECEIVER
- MD01_MT01_RQSD_0001    REQUESTER
- MD01_MT01_RQSV_0001    REQUESTER
- MD01_MT01_SDRC_0001    RECEIVER
- MD01_MT01_SVRC_0001    RECEIVER
- ME02_MT01_RQSD_0001    REQUESTER
- ME02_MT01_RQSV_0001    REQUESTER
- ME02_MT01_SDRC_0001    RECEIVER
- ME02_MT01_SVRC_0001    RECEIVER

F1 - Enter/Display/Modify    F2 - Create    F3 - Copy    F4 - Delete
F5 - Monitor                F6 - Resolve   F7 - Reset MSN F8 - Start/Stop F10 - Status
F12 - Refresh                PGDN           PGUP         F16 - Return
```

Figure 15. The Channel Menu

The Channel Menu displays a list of channels that match your search criteria. From the Channel Menu you can:

- Display and modify channel status.
- Create a new channel definition.
- Copy a channel definition.
- Delete a channel definition.
- Monitor channel status.
- Resolve a channel.
- Reset a message sequence number (MSN).
- Start or stop a channel.

Modifying a channel

On the Channel Menu, type any character against the channel you want to modify, and press the Enter/Display/Modify key (F1). The appropriate panel is displayed. For example, if you select a sender channel, the Display/Modify Sender Channel panel is displayed:

```

          IBM MQSeries for Tandem NonStop Kernel Version 2
                ** Display/Modify Sender Channel **
Channel Name   : MT01.M401.SDRC.0001_
Description    : Sender to M401_____
Xmit Queue Name : M401.TQ.SDRC.0001_____
Data Conversion Y/N: N
User Id       : _____ Password           : _____
MCA Name      : _____ MCA UserID        : _____
Batch Size    : _____ 50 Max Message Size : _____ 4194304
MSN Wrap Count : _____ 9999999 Disconnect Interval: _____ 60
Short Retry Count : _____ 10 Short Timer    : _____ 60
Long Retry Count : _____ 9999999 Long Timer     : _____ 1200

Transport Protocol : 1 (1=Lu6.2/ 2=TCP/IP) TCP/IP Port Number : _____
TCP/IP Address     : _____
Lu62 AutoStart Y/N : N APC/ICE Process      : $BP01_
Local LU Name      : IYAHT080 Remote LU Name  : IYAFT110
Local TP Name      : INTCRS6A_____ Mode Name   : LU62PS__
Remote TP Name     : _____

F1 - Modify      PGDN - Exits                               F16 - Return
    
```

Figure 16. The Display/Modify Sender Channel panel (1)

Press the PGDN key to display the second panel of information:

Message Queue Management (MQM)

```
IBM MQSeries for Tandem NonStop Kernel Version 2
** Display/Modify Sender Channel **

Message Data: _____
Message Exit: _____

Receive Data: _____
Receive Exit: _____

Scrty Data: _____
Scrty Exit: _____

Send Data: _____
Send Exit: _____

PGUP - Return
```

Figure 17. The Display/Modify Sender Channel panel (2)

Overtyping those values you want to modify, and press the Modify key (F1). You are prevented from overtyping those values that cannot be modified.

Creating a channel definition

From the Channel Menu, press the Create key (F2) to display the Create Channel panel:

```
IBM MQSeries for Tandem NonStop Kernel Version 2
** Create Channel **

Channel Type: 1 1=Sender, 2 = Server, 3=Receiver,
               4=Requester, 5 = Server Connection

Name: _____

Replace [Y/N]: _

F1 - Enter                                F16 - Return
```

Figure 18. The Create Channel panel

To create a new channel definition:

1. Type 1 (for a sender channel), 2 (for a server channel), 3 (for a receiver channel), 4 (for a requester channel), or 5 (for a server connection) in the Channel Type field.
2. Type the name of the channel definition in the Name field.
3. Press the Enter key (F1).
4. Type Y in the Replace field if the definition is to replace an existing definition of the same name and type.

If you enter a 1 in the Channel Type field, the Create Sender Channel panel is displayed:

```

          IBM MQSeries for Tandem NonStop Kernel Version 2
                ** Create Sender Channel **
Channel Name   : TANDEM_TO_SOLARIS__
Description    : _____
Xmit Queue Name : _____
Data Conversion Y/N:
User Id       : _____ Password       : _____
MCA Name      : _____ MCA UserID     : _____
Batch Size    : _____ 50 Max Message Size : _____ 4194304
MSN Wrap Count : _____ 999999999 Disconnect Interval: _____ 6000
Short Retry Count : _____ 999999 Short Timer      : _____ 60
Long Retry Count  : _____ 999999 Long Timer       : _____ 1200

Transport Protocol : _ (1=Lu6.2/ 2=TCP/IP) TCP/IP Port Number : _____
TCP/IP Address     : _____
Lu62 AutoStart Y/N : _____ APC/ICE Process : _____
Local LU Name      : _____ Remote LU Name  : _____
Local TP Name      : _____ Mode Name      : _____
Remote TP Name     : _____

F1 - Enter      PGDN - Exits                      F16 - Return
    
```

Figure 19. The Create Sender Channel panel

If you enter a 3 in the Channel Type field, the Create Receiver Channel panel is displayed:

Message Queue Management (MQM)

```
IBM MQSeries for Tandem NonStop Kernel Version 2
** Create Receiver Channel **

Channel Name      : SOLARIS_TO_TANDEM__
Description      : _____
                  _____

Put Authority D/C : _                MSN Wrap Count   : 999999999
User Id          : _____        MCA UserID      : _____
Batch Size       : _____        Max Message Size : 4194304
Retry Count      : _____        Retry Interval   : 1000

Transport Protocol : _ (1=Lu6.2/ 2=TCP/IP)
Lu62 AutoStart Y/N : _                APC/ICE Process  : _____
Local LU Name     : _____
Local TP Name     : _____

F1 - Enter      PGDN - Exits                      F16 - Return
```

Figure 20. The Create Receiver Channel panel

If you enter a 5 in the Channel Type field, the Create Server Connection Channel panel is displayed:

```
IBM MQSeries for Tandem NonStop Kernel Version 2
** Create Server Connection Channel **

Channel Name      : WINDOWS_CLIENT_____
Description      : _____
                  _____

MCA UserID       : _____
Max Message Size : 4194304

Transport Protocol : _ (1=Lu6.2/ 2=TCP/IP)
Lu62 AutoStart Y/N : _                APC/ICE Process  : _____
Local LU Name     : _____
Local TP Name     : _____

F1 - Enter      PGDN - Exits                      F16 - Return
```

Figure 21. The Create Server Connection Channel panel

To create a new channel definition, complete the requested panel and press the Enter key (F1).

Monitoring a channel

Press the Monitor key (F5) from the Channel Menu panel to display the Monitor Channels panel:

```

      IBM MQSeries for Tandem NonStop Kernel Version 2
            ** Monitor Channels **
Channel Name      Status      Curr MSN   Last MSN   MCA Status  Stop
-----
MT01.MH01.SDRC.0002  BINDING
MT01.VM03.SDRC.0002  RUNNING      6266      6266      RUNNING     NO

F12 - Refresh                PGDN                PGUP                F16 - Return

Refreshing.....
    
```

Figure 22. The Monitor Channels panel

The MQMQMREFRESHINT pathway parameter for MQS-MQMSVR00 determines the frequency with which monitor screens for channels and queues are refreshed. The default frequency is every 30 seconds. To change the frequency to every 10 seconds, for example, enter:

```
alter server mqs-mqmsvr00, param mqmqmrefreshint 10
```

from the pathway for your queue manager.

Deleting a channel

On the Channel Menu, select a channel to delete by typing any character against the channel name. Press the Delete key (F4) to delete the channel, then press F4 again to confirm the deletion request.

Displaying channel status

Press the Status key (F10) from the Channel Menu panel to display the Channel Status panel:

Message Queue Management (MQM)

```
IBM MQSeries for Tandem NonStop Kernel Version 2
Channel Status

Channel Name : MT01.VM03.SDRC.0002_
Xmit Queue Name: VM03NCM.TQ.SDRC.0001
Connection Name: $BP01.IYAHT080.IYCNVM03

Channel Status : RUNNING_ In Doubt : NO_
Start Date : 1997-09-09 Start Time : 15.07.14
Last Msg Date : 1997-09-08 Last Msg Time : 16.34.04

MCA Job Name : 000069AA
Current LUW ID : 03544240E28B0277
Last LUW ID : 03544240E28B0277 Current Messages : 0
MCA Status : RUNNING_ Current Seq Num : 6266
Stop Requested : NO_ Last Seq Num : 6266
Number of Batches : 6 Number of Messages : 6
Number of Buffers Sent: 14 Number of Buffers Recvd: 7
Number of Bytes Sent : 3204 Number of Bytes Recvd : 196
Num of Long Retry Left: 9999999 Num of Short Retry Left: 10

F12 - Refresh F16 - Return
```

Figure 23. The Channel Status panel

Starting and stopping a channel

Press the Start/Stop key (F8) from the Channel Menu to display the Start/Stop Channel panel:

```
IBM MQSeries for Tandem NonStop Kernel Version 2
Start/Stop Channel

Name: MT01_MA02_SDRC_0001_

Status:

Action: _ choose one of the following:
1. Start Channel
2. Stop Immediate
3. Stop Quiesce

F1 - Enter F16 - Return
```

Figure 24. The Start/Stop Channel panel

Type the name of the channel in the Name field, and type a number between 1 and 3 in the Action field. Press the Enter key (F1).

Resetting a Message Sequence Number (MSN)

From the Channel Menu, press the Reset MSN key (F7) to display the Reset Channel panel:

```
IBM MQSeries for Tandem NonStop Kernel Version 2
Reset Channel

Name: MT01_M401_RQSD_0001_____
Sequence Number:      1

F1 - Enter                                     F16 - Return
```

Figure 25. The Reset Channel panel

The MSN ensures nonduplication of messages, and ensures that messages are stored in the same order as they are transmitted. This screen lets you reset the sequence number of a channel if necessary.

Resolving a channel

From the Channel Menu, press the Resolve key (F6) to display the Resolve Channel panel.

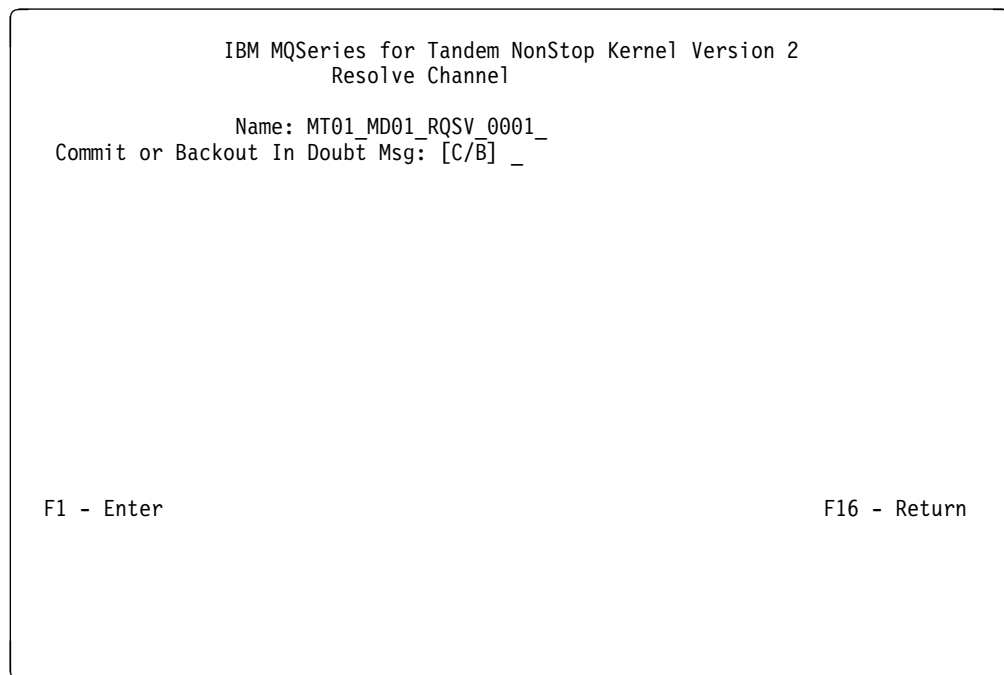


Figure 26. The Resolve Channel panel

You can:

- Backout the in-doubt message batch (B)
- Commit the in-doubt message batch (C)

Copying a channel

On the Channel Menu, press the Copy key (F3). The Copy Channel panel is displayed:

```
IBM MQSeries for Tandem NonStop Kernel Version 2
** Copy Channel **

Name: _____
Replace [Y/N]: _
Like Name: MT01_M401_RQSV_0001_
Channel Type: SERVER

F1 - Enter                                     F16 - Return
```

Figure 27. The Copy Channel panel

Type the name of the new channel in the Name field; type the name of the channel definition you are copying in the Like Name field; type the channel type in the Channel Type field. Press the Enter key (F1) to copy the channel definition.

Message Queue Management (MQM)

Chapter 6. Administering local MQSeries objects

This chapter explains how to administer local MQSeries objects to support application programs that use the Message Queuing Interface (MQI). The MQI lets application programs access message queuing services.

Local administration is when you create, display, change, copy, and delete MQSeries objects.

This chapter contains these sections:

- “Supporting application programs that use the MQI”
- “Issuing MQSC commands for administration”
- “Running MQSCs from text files” on page 102
- “Troubleshooting MQSC” on page 106
- “Working with local queues” on page 108
- “Working with alias queues” on page 117
- “Working with model queues” on page 119
- “Managing objects for triggering” on page 120

Supporting application programs that use the MQI

MQI application programs need specific objects before they can run successfully. An MQI application can remove messages from a queue, process them, and send the results to another queue on the same queue manager.

Whereas applications can put (using MQPUT) messages on local or remote queues, they can only get (using MQGET) messages directly from local queues.

Before this application can be run, these conditions must be satisfied:

- The queue manager must exist and be running.
- The first application queue, from which the messages are to be removed, must be defined.
- The second queue, on which the application puts the messages, must also be defined (unless it is a dynamic queue).
- The application must be able to connect to the queue manager. To perform this task, it must be linked to the product code. See Appendix G, “Building and running applications” on page 315 for more information.
- The applications that put the messages on the first queue must also connect to a queue manager. If they are remote, they must also be set up with transmission queues and channels.

Issuing MQSC commands for administration

MQSeries commands (MQSC) let you manipulate MQSeries objects. You can issue commands using the **runmqsc** command at the command prompt.

See Appendix E, “MQSC supported by MQSeries for Tandem NSK” on page 297 for more information about using MQSC in the MQSeries for Tandem NSK environment.

Before you start

Before you begin, you must create and then start the queue manager, which runs the MQSC commands. See “Creating a default queue manager” on page 72 for more information.

MQSeries object names

When you are issuing MQSC commands, you must specify the local names of queue objects. For example: `ORANGE.LOCAL.QUEUE`, where `LOCAL.QUEUE` signifies that this queue is a local queue. This naming convention is not required for the names of all local queues.

In this section, the name `saturn.queue.manager` is used as a queue manager name.

Note that queue manager names are case sensitive.

Object names on MQSC commands

MQSC commands and their attributes can be in uppercase or lowercase letters. Object names in MQSC commands are folded (that is, `QUEUE` and `queue` are not differentiated), unless the names are put in single quotation marks. If quotation marks are not used, uppercase letters are used for the object name. See the *MQSeries Command Reference* book for more information.

However, some arguments of the `runmqsc` command, which invokes the MQSC facility, are case sensitive; see “Using control commands” on page 43.

Entering MQSC interactively

To enter commands interactively, open a TACL session and enter:

```
runmqsc
```

In this example, a queue manager name has not been specified, therefore the MQSCs are processed by the default queue manager. You can enter any MQSC command. For example:

```
MQSC>DEFINE QLOCAL (ORANGE.LOCAL.QUEUE)
```

Getting feedback from MQSCs

When you issue MQSCs, the queue manager provides confirmation or error messages. For example:

```
AMQ8006: MQSeries queue created
.
.
.
AMQ8405: Syntax error detected at or near end of command segment below:-
```

The first message confirms that a queue has been successfully created. The second message indicates that you have made a syntax error. If you have not entered the command correctly, refer to the *MQSeries Command Reference* manual for the correct syntax.

Ending interactive input to MQSC

If you are using MQSC interactively, you can exit by entering the EOF character CTRL+Y, or by typing `exit` or `quit` and pressing Enter.

If you are redirecting input from other sources, such as a text file, MQSC terminates when the end of file is reached.

Displaying queue manager attributes

To display the attributes of the queue manager specified on the `runmqsc` command, use the following MQSeries command:

```
MQSC>DISPLAY QMGR ALL
```

An example output is as follows:

```
1 : display qmgr all
AMQ8408: Display Queue Manager details.
DESCR( )
DEADQ(SYSTEM.DEAD.LETTER.QUEUE)
DEFXMITQ(MY.DEFAULT.XMIT.QUEUE)
COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE)
QMNAME(saturn.queue.manager)
TRIGINT(999999999)
MAXHANDS(256)
MAXUMSGS(10000)
AUTHOREV(DISABLED)
INHIBTEV(ENABLED)
LOCALEV(DISABLED)
REMOTEEV(DISABLED)
PERFMEEV(DISABLED)
STRSTPEV(ENABLED)
MAXPRTY(9)
CCSID(819)
MAXMSGL(4194304)
CMDLEVEL(220)
PLATFORM(NSK)
SYNCPT
```

Figure 28. Example output for QMGR ALL

The ALL parameter on the DISPLAY QMGR command causes all the queue manager attributes to be displayed. The output tells us the queue manager name (saturn.queue.manager), and the names of the dead-letter queue (SYSTEM.DEAD.LETTER.QUEUE) and the command queue (SYSTEM.ADMIN.COMMAND.QUEUE). Both these queues are created when you run the sample MQSC input file, `amqscoma`; see “Creating the default and system

Running MQSC commands

objects” on page 74. Note that, if you do not specify the name of a dead-letter queue on the **crtmqm** command, you must alter the queue manager to associate a dead-letter queue with the queue manager.

You should confirm that these queues are created by entering the command:

```
DISPLAY QUEUE (*)
```

Using a nondefault queue manager

You can specify a queue manager name when executing the **runmqsc** command to run MQSCs on a local queue manager (other than the default). For example, to run MQSCs on queue manager named `jupiter.queue.manager`, use this command:

```
runmqsc jupiter.queue.manager
```

All the MQSCs you enter are processed by this queue manager providing the queue manager is on the same node and is already running.

You can also run MQSC commands on a remote queue manager; see “Issuing MQSC commands remotely” on page 130.

Altering queue manager attributes

To alter the attributes of the queue manager specified with the **runmqsc** command, use the MQSC ALTER QMGR, specifying the attributes and values that you want to change. For example, use the following commands to alter the attributes of `jupiter.queue.manager`:

```
runmqsc jupiter.queue.manager  
  
ALTER QMGR DEADQ (ANOTHERDLQ) INHIBTEV (ENABLED)
```

The ALTER QMGR command changes the dead-letter queue used, and enables inhibit events.

Running MQSCs from text files

Running MQSCs interactively is appropriate for quick tests; however, if you have long commands, or commands that you want to repeat, you should take input from a text file.

To perform this task, create a text file containing the MQSCs using your text editor. When you use the **runmqsc** command, use the TA CL IN and OUT redirection operators, or the flags **-i** and **-o** on **runmqsc**. For example, the following command runs a sequence of commands contained in the text file `mymqscin`:

```
runmqsc /IN mymqscin/
```

or

```
runmqsc -i mymqscin
```

Similarly, you can redirect the output to a file. A file containing the MQSCs for input is called an *MQSC file*. The output file containing replies from the queue manager is called the *report file*.

To redirect both input and output on the **runmqsc** command, use this command:

```
runmqsc /IN mymqscin, OUT mymqscou/
```

or

```
runmqsc -i mymqscin -o mymqscou
```

This command invokes the MQSC commands contained in the file `mymqscin`. Because a queue manager name is not specified, the MQSC commands are run against the default queue manager. The output is sent to the report file `mymqscou`. Figure 29 on page 104 shows an extract from the MQSC command file `mymqscin`, and Figure 30 on page 105 shows the corresponding extract of the output in `mymqscou`.

To redirect input and output on the **runmqsc** command for a queue manager (`saturn.queue.manager`) that is not the default, use the command:

```
runmqsc /IN mymqscin, OUT mymqscou/ saturn.queue.manager
```

or

```
runmqsc -i mymqscin -o mymqscou saturn.queue.manager
```

Using MQSC files

MQSC command files are written as EDIT files (Tandem file type code 101). Figure 29 on page 104 is an extract from an MQSC file showing an MQSeries command (DEFINE QLOCAL) with its attributes. The *MQSeries Command Reference* manual contains a description of each MQSC command and its syntax.

Running MQSC commands

```
.  
.   
.   
DEFINE QLOCAL(ORANGE.LOCAL.QUEUE) REPLACE +  
DESCR(' ') +  
PUT(ENABLED) +  
DEFPRTY(0) +  
DEFPSIST(NO) +  
GET(ENABLED) +  
MAXDEPTH(5000) +  
MAXMSGL(1024) +  
DEFSOPT(SHARED) +  
NOHARDENBO +  
USAGE(NORMAL) +  
NOTRIGGER  
.   
.   
. 
```

Figure 29. Extract from the MQSC command file, *mymqscin*

You must limit lines to a maximum of 72 characters. The plus sign (+) indicates that the command is continued on the next line. Note that the plus sign must be preceded by a space.

Using MQSC reports

The *runmqsc* command returns a *report*, which is sent to the current OUT stream. The report contains:

- A header identifying MQSC as the source of the report:
Starting MQSeries Commands.
- An optional numbered listing of the MQSC commands issued. By default, the text of the input is echoed to the output. Within this output, each command is prefixed by a sequence number, as shown in Figure 30 on page 105. However, you can use the *-e* flag on the **runmqsc** command to suppress the output.
- A syntax error message for any commands found to be in error.
- An *operator message* indicating the outcome of running each command. For example, the operator message for the successful completion of a DEFINE QLOCAL command is:
AMQ8006: MQSeries queue created.
- Other messages resulting from general errors when running the script file.
- A brief statistical summary of the report indicating the number of commands read, the number of commands with syntax errors, and the number of commands that could not be processed.

Note: The queue manager attempts to process only those commands that have no syntax errors.

```

Starting MQSeries Commands.
.
.
12:    DEFINE QLOCAL('RED.LOCAL.QUEUE') REPLACE +
:      DESCR(' ') +
:      PUT(ENABLED) +
:      DEFPRTY(0) +
:      DEFPSIST(NO) +
:      GET(ENABLED) +
:      MAXDEPTH(5000) +
:      MAXMSGL(1024) +
:      DEFSOPT(SHARED) +
:      USAGE(NORMAL) +
:      NOTRIGGER
AMQ8006: MQSeries queue created.
:
.
.
15 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.

```

Figure 30. Extract from the MQSC report file, mymqscou

Running the supplied MQSC command files

When you install MQSeries for Tandem NSK, these MQSC files are supplied:

amqscoma	Default and system objects.
amqscos0	Definitions of objects used by sample programs.

The files are located in the samples subvolume, by default \$SYSTEM.ZMQSSMPL.

You should already have run **runmqsc** against the command file amqscoma. If you have not performed this step, or if you have deleted any of the objects created from it, run it again by entering:

```
runmqsc -i $SYSTEM.ZMQSSMPL.AMQSCOMA
```

The DEFINE commands in amqscoma specify the REPLACE option, which overwrites the existing definitions, if possible. See the *MQSeries Command Reference* manual for more information about REPLACE.

Using runmqsc to verify commands

You can use the **runmqsc** command to verify MQSC commands on a local default queue manager without actually running them. To perform this step, set the -v flag on the **runmqsc** command. For example:

```
runmqsc -i mymqscin -o mymqscou -v
```

Troubleshooting

When you invoke **runmqsc** against an MQSC command file, the queue manager verifies each command and returns a report without actually running the MQSC commands. This action lets you check the syntax of all the commands in your command file. This step is important if you are:

- Running a large number of commands from a command file
- Using an MQSC command file many times over

This report is similar to that shown in Figure 30 on page 105.

You cannot use this method to verify MQSC commands remotely. For example, if you attempt this command:

```
runmqsc -i mymqscin -o mymqscou -w 30 -v jupiter.queue.manager
```

the **-w** flag is ignored, and the command is run locally.

Troubleshooting MQSC

If MQSCs do not run properly, use the following checklist to see if any of these common problems apply to you.

When you use the **runmqsc** command:

- Check that `$SYSTEM.ZMQSEXE` is in `PMSEARCH` in `TACLCSTM`.
- Remember to specify a fully qualified file name for `amqscoma` on input to **runmqsc** if you are not in the subvolume `ZMQSSMPL`.
- Use the `IN` operator or the `-i` flag when redirecting input from a file. Otherwise, the queue manager interprets the file name as a queue manager name. For example:

```
runmqsc amqscoma

5697-A17 (C) Copyright IBM Corp. 1997. ALL RIGHTS RESERVED.
Starting MQSeries Commands.

AMQ8118: MQSeries queue manager does not exist.
0 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

- If you redirect output to a file, use the `OUT` operator or the `-o` flag. By default, the output file is created using the `TACL` defaults in effect at the time the command was issued. Specify a fully qualified file name to send your output to a specific file. For example:

```
runmqsc -i $SYSTEM.ZMQSSMPL.AMQSCOMA -o $DATA0.OUTPUT.MYFILE
```


- Check that you successfully created the queue manager that is going to run the commands. To do this, look in the configuration file MQSINI, which by default is located in the installation subvolume, \$SYSTEM.ZMQSSYS. This file contains the names of the queue managers and the name of the default queue manager, if you have one.
- The queue manager should already be started; if it is not, start it, as described in “Starting a queue manager” on page 74. You get an error message if the queue manager is already started.
- Specify a queue manager name on the **runmqsc** command if you have not defined a default queue manager, otherwise you get this error:

```
runmqsc -i $SYSTEM.ZMQSSMPL.AMQSCOMA

5697-A17 (C) Copyright IBM Corp. 1997. ALL RIGHTS RESERVED.
Starting MQSeries Commands.

AMQ8146: MQSeries queue manager not available.
0 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

For information about correcting this type of problem, see “Making an existing queue manager the default” on page 76.

- You cannot specify an MQSC command as a **runmqsc** parameter:

```
runmqsc DEFINE QLOCAL(FRED)
```

- You cannot enter MQSC commands from TACL before you issue the **runmqsc** command. For example:

```
DEFINE QLOCAL(Queue1)

* Error Name of Variable, built-in, or file needed.
```

- You cannot run control commands from **runmqsc**. For example, you cannot start a queue manager once you are running MQSC interactively:

```
runmqsc
5697-A17 (C) Copyright IBM Corp. 1997. ALL RIGHTS RESERVED.
Starting MQSeries Commands.

strmqm saturn.queue.manager
  1 : strmqm saturn.queue.manager
AMQ8405: Syntax error detected at or near end of command segment below:
```

See also “If you have problems using MQSC remotely” on page 131.

Working with local queues

This section contains examples of MQSCs that you can use. Refer to the *MQSeries Command Reference* for a complete description of these commands.

Defining a local queue

For an application, the local queue manager is the queue manager to which the application is connected. Queues that are managed by the local queue manager are local to that queue manager.

Use the MQSC DEFINE QLOCAL to create a definition of a local queue and also to create the data structure that is called a queue. You can also modify the queue characteristics from those of the default local queue.

In this example, ORANGE.LOCAL.QUEUE is specified to have these characteristics:

- It is enabled for gets, disabled for puts, and operates on a first-in-first-out (FIFO) basis.
- It is an 'ordinary' queue. That is, it is not an initiation queue or a transmission queue, and it does not generate trigger messages.
- The maximum queue depth is 1000 messages; the maximum message length is 2000 bytes.

The following MQSC command performs this action:

```
DEFINE QLOCAL (ORANGE.LOCAL.QUEUE) +
  DESCR('Queue for messages from other systems') +
  PUT (DISABLED) +
  GET (ENABLED) +
  NOTRIGGER +
  MSGDLVSQ (FIFO) +
  MAXDEPTH (1000) +
  MAXMSGL (2000) +
  USAGE (NORMAL)
```

Notes:

1. Most of these attributes are the defaults as supplied with the product. However, they are shown here for purposes of illustration. You can omit them if you are sure that the defaults are what you want or have not been changed.
2. USAGE (NORMAL) indicates that this queue is not a transmission queue.
3. If you already have a local queue on the same queue manager with the name ORANGE.LOCAL.QUEUE, this command fails. Use the REPLACE attribute, if you want to overwrite the existing definition of a queue, but see also "Changing local queue attributes" on page 111.

Default physical file size for queues

By default, the queue manager creates queue data files that support up to 100 MB of data. If this limit is reached, applications receive the return code MQRC_Q_SPACE_NOT_AVAILABLE. To change the maximum storage allocated to a queue, identify the physical files that hold the data for the queue using **dspmqls**, then use the FUP utility to change the MAXEXTENTS attribute on the queue file (prefix Q) or the overflow file (prefix O). The maximum value for MAXEXTENTS on any file is 978, which supports approximately 195 MB of data. If more storage is required, you can modify the queue file to support larger extent sizes and, if necessary, you can partition the file across multiple volumes.

Note: A queue that has had all message-retrieval options disabled (as described in “Specifying retrieval options for queues” on page 40) cannot be partitioned across multiple volumes.

Defining a dead-letter queue

Each queue manager should have a local queue to be used as a dead-letter queue so that messages that cannot be delivered to their correct destination can be stored for later retrieval.

You must tell the queue manager about the dead-letter queue. You can do this by specifying a dead-letter queue on the **crtmqm** command or you can use the ALTER QMGR command to specify one later. You must also define the dead-letter queue before it can be used.

A sample dead-letter queue called SYSTEM.DEAD.LETTER.QUEUE is supplied with the product in the file amqscoma. This queue is automatically created when you run the sample. You can modify this definition, if required. There is no need to rename it.

A dead-letter queue has no special requirements except that it must be a local queue and its MAXMSGL (maximum message length) attribute must enable the queue to accommodate the largest messages that the queue manager has to handle.

MQSeries provides a dead-letter queue handler that lets you specify how messages found on a dead-letter queue are to be processed or removed. For further information, see Chapter 9, “MQSeries dead-letter queue handler” on page 157.

Displaying default object attributes

When you define an MQSeries object, it takes any attributes that you do not specify from the default object. For example, when you define a local queue, the queue inherits any attributes that you omit in the definition from the default local queue, which is called SYSTEM.DEFAULT.LOCAL.QUEUE. To see exactly what these attributes are, use the following command:

```
DISPLAY QUEUE (SYSTEM.DEFAULT.LOCAL.QUEUE) ALL
```

Note: The syntax of this command is different from that of the corresponding DEFINE command.

Copying a local queue definition

You can selectively display attributes by specifying them individually. For example:

```
DISPLAY QUEUE (ORANGE.LOCAL.QUEUE) +  
    MAXDEPTH +  
    MAXMSGL +  
    CURDEPTH
```

This command displays the three specified attributes as follows:

```
AMQ8409: Display Queue details.  
    QUEUE(ORANGE.LOCAL.QUEUE)  
    MAXDEPTH(1000)  
    MAXMSGL(2000)  
    CURDEPTH(0)
```

CURDEPTH is the current queue depth, that is, the number of messages on the queue. This is a useful attribute to display, because by monitoring the queue depth you can ensure that the queue does not become full.

Copying a local queue definition

You can copy a queue definition using the LIKE attribute on the DEFINE command. For example:

```
DEFINE QLOCAL (MAGENTA.QUEUE) +  
    LIKE (ORANGE.LOCAL.QUEUE)
```

This command creates a queue with the same attributes as our original queue ORANGE.LOCAL.QUEUE, rather than those of the system default local queue.

You can also use this form of the DEFINE command to copy a queue definition, but substituting one or more changes to the attributes of the original. For example:

```
DEFINE QLOCAL (THIRD.QUEUE) +  
    LIKE (ORANGE.LOCAL.QUEUE) +  
    MAXMSGL(1024)
```

This command copies the attributes of the queue ORANGE.LOCAL.QUEUE to the queue THIRD.QUEUE, but specifies that the maximum message length on the new queue is to be 1024 bytes, rather than 2000.

Notes:

1. When you use the LIKE attribute on a DEFINE command, you are copying the queue attributes only. You are not copying the messages on the queue.
2. If you define a local queue, without specifying LIKE, it is the same as DEFINE LIKE(SYSTEM.DEFAULT.LOCAL.QUEUE).

Changing local queue attributes

You can change queue attributes in two ways, using either the ALTER QLOCAL command or the DEFINE QLOCAL command with the REPLACE attribute. In “Defining a local queue” on page 108, we defined the queue ORANGE.LOCAL.QUEUE. Suppose, for example, you wanted to increase the maximum message length on this queue to 10 000 bytes.

- Using the ALTER command:

```
ALTER QLOCAL (ORANGE.LOCAL.QUEUE) MAXMSGL(10000)
```

This command changes a single attribute, that of the maximum message length; all the other attributes remain the same.

- Using the DEFINE command with the REPLACE option, for example:

```
DEFINE QLOCAL (ORANGE.LOCAL.QUEUE) MAXMSGL(10000) REPLACE
```

This command changes not only the maximum message length, but all the other attributes, which are given their default values. The queue is now put enabled, whereas previously it was put inhibited. Put enabled is the default, as specified by the queue SYSTEM.DEFAULT.LOCAL.QUEUE, unless you have changed it.

If you *decrease* the maximum message length on an existing queue, existing messages are not affected. Any new messages, however, must meet the new criteria.

Changing the volume of a local queue

Use the **altmqfls** command to change the volume on which a local, predefined queue is stored. This might be necessary to spread disk I/O across volumes to balance the system for optimum performance. The queue manager must have been started before this command is issued, and the queue itself must not be open. Only one queue may be named on any **altmqfls** command. See “altmqfls (Alter queue file attributes)” on page 220 for the syntax of the **altmqfls** command.

Changing the recalculation, update, and retrieval options for a local queue

Use the **altmqfls** command to change:

- The frequency with which queue depth is recalculated, and the events that cause the queue depth to be recalculated, by the MQSS Server for a queue. CURDEPTH for a queue can become inaccurate as a result of TM/MP aborting transactions containing MQSeries messages.

Use this to select the appropriate trade-off between the accuracy of the CURDEPTH attribute for a queue and the cost of recalculation (where MQSeries must read all messages on the queue in order to count them).

The options are:

OPEN

Recalculate depth on every MQOPEN.

CLOSE

Recalculate depth on every MQCLOSE.

BOTH

Recalculate depth on every MQOPEN and MQCLOSE.

FIRSTOPEN

Recalculate depth every time the first opener opens the queue.

LASTCLOSE

Recalculate depth every time the last opener closes the queue.

FIRSTLAST

Recalculate depth every time the first opener opens the queue and the last opener closes the queue.

NEVER

Never recalculate the depth of the queue.

n

Recalculate the depth of the queue every *n* hours.

The greatest accuracy in queue depth (but the highest cost in terms of processing) is achieved using the BOTH option. The lowest cost in terms of processing (but the lowest accuracy in queue depth) is achieved using the NEVER option. The default is FIRSTOPEN.

- The frequency with which latest status of an object is saved to disk, and the events that cause latest status to be saved, by the MQSS Server.

Use this to keep the object catalog up to date with the latest status data.

The options are:

LASTCLOSE

Update the object catalog every time the last opener closes the object.

SHUTDOWN

Update the object catalog on queue manager shutdown.

NEVER

Never update the object catalog.

n

Update the object catalog every *n* hours.

The default is LASTCLOSE.

- The retrieval options that are supported for a queue.

Use this to reduce the I/O activity on queue files when applications do not need to use MQGET retrieval by *Msgld*, *Correld*, or both.

The options are:

M C B Any combination of M, C and B that indicates the combination of retrieval criteria to support. Select M for *Msgld*, C for *Correld*, and B for the combination of *Msgld* and *Correld*.

NONE Support default retrieval criteria (FIFO and FIFO within priority) only.

The default is MCB, indicating that all retrieval criteria can be used.

You can use the **dspmqfls** command to display the current settings for these attributes.

Note that these attributes can be set for local and model queues. Dynamic queues inherit these attributes from the model queue that is used to create them.

You must not remove all message-retrieval options for a partitioned local queue. The **altmqfls** command does not prevent you from doing this; however, if there are no alternate key files for a queue that is partitioned, the waited MQGET function does not operate correctly.

Reassignment of objects to MQSS Servers

The **altmqfls** command is used to reconfigure the MQSS Server for an object. Initially, all objects are created to use the default MQSS Server. Using **altmqfls** after an object has been created, you can configure the object to use an MQSS Server other than the default. You must have configured the new Pathway server class for the MQSS Server and have started it before you can use the object.

You may specify either a process name or the word DEFAULT on the command line for **altmqfls**. No checking is performed that the new MQSS Server is active or configured at the time the object is reconfigured.

You can use the **dspmqfls** command to display the current MQSS Server for an object.

Note that the MQSS Server can be set for local and model queues. Dynamic queues inherit the MQSS Server from the model queue that is used to create them.

Clearing a local queue

To delete all the messages from a local queue called MAGENTA.QUEUE, use the following command:

```
CLEAR QLOCAL (MAGENTA.QUEUE)
```

You cannot clear a queue if:

- There are uncommitted messages that have been put on the queue under syncpoint.
- An application currently has the queue open.

Deleting a local queue

Use the MQSC command DELETE QLOCAL to delete a local queue. A queue cannot be deleted if it has uncommitted messages on it. However, if the queue has one or more committed messages, and no uncommitted messages, it can only be deleted if you specify the PURGE option. For example:

```
DELETE QLOCAL (PINK.QUEUE) PURGE
```

Specifying NOPURGE instead of PURGE ensures that the queue is not deleted if it contains any committed messages.

Browsing queues

MQSeries for Tandem NSK provides a sample queue browser to enable you to look at the contents of the messages on a queue. The browser is supplied both as source and as a module that can be run. By default, the file names and paths are:

```
Source      $SYSTEM.ZMQSSMPL.AMQSBCG0
Executable  $SYSTEM.ZMQSSMPL.AMQSBCG
```

The sample takes two parameters:

```
Queue name      For example, SYSTEM.ADMIN.RESPQ.tpp01.
Queue manager name For example, snooker.
```

For example:

```
AMQSBCG SYSTEM.ADMIN.RESPQ.tpp01 snooker
```

There are no defaults; both parameters are required. Typical results from this command are:

```
AMQSBCG - starts here
*****
```

```
MQCONN to snooker
MQOPEN - 'SYSTEM.ADMIN.RESPQ.tpp01'
```

```
MQGET of message number 1
****Message descriptor****
```

```
StrucId  : 'MD '  Version : 1
Report   : 0  MsgType : 8
Expiry   : -1  Feedback : 0
Encoding : 273  CodedCharSetId : 850
Format   : 'AMQMRESP'
Priority  : 5  Persistence : 1
MsgId    : X'414D5120736E6F66B6572202020202ED47690071A6D00'
CorrelId : X'000000000000000000000000000000000000000000000000'
BackoutCount : 0
ReplyToQ   : '
ReplyToQMgr : 'snooker
** Identity Context
UserIdentifier : 'tpp01'
```


Browsing queues

length - 36 bytes

```
00000000: 0000 0002 0000 0024 0000 0001 0000 0015 '.....$......'  
00000010: 0000 0001 0000 0001 0000 0000 0000 0000 '.....'  
00000020: 0000 0000                                     '.....'
```

MQGET of message number 3

****Message descriptor****

```
StrucId : 'MD ' Version : 1  
Report  : 0 MsgType : 8  
Expiry  : -1 Feedback : 0  
Encoding : 273 CodedCharSetId : 850  
Format  : 'AMQMRESP'  
Priority : 5 Persistence : 1  
MsgId   : X'414D5120736E6F6F6B6572202020202020ED477D62A9EA100'  
CorrelId : X'000000000000000000000000000000000000000000000000000000000000000000'  
BackoutCount : 0  
ReplyToQ      : '  
ReplyToQMgr   : 'snooker'  
** Identity Context  
UserIdentifier : 'trevor'  
AccountingToken :  
X'04373037300000000000000000000000000000000000000000000000000000000000'  
ApplIdentityData : '  
** Origin Context  
PutApplType  : '6'  
PutApplName  : '  
PutDate     : '19941124' PutTime : '11240678'  
ApplOriginData : ' ' '
```

**** Message ****

length - 188 bytes

```
00000000: 736E 6F6F 6B65 7220 2020 2020 2020 2020 'snooker'  
00000010: 2020 2020 2020 2020 2020 2020 2020 2020 '  
00000020: 2020 2020 2020 2020 2020 2020 2020 2020 '  
00000030: 534E 4F4F 4B45 522E 5749 4748 542E 5443 'SNOOKER.WIGHT.TC'  
00000040: 5020 2020 2020 2020 2020 2020 2020 2020 'P'  
00000050: 2020 2020 2020 2020 2020 2020 2020 2020 '  
00000060: 0000 0001 0000 0024 0000 0001 0000 0015 '.....$......'  
00000070: 0000 0001 0000 0001 0000 0000 0000 0000 '.....'  
00000080: 0000 0002 0000 0004 0000 0028 0000 0DAD '.....(.....'  
00000090: 0000 0000 0000 0014 534E 4F4F 4B45 522E '.....SNOOKER.'  
000000A0: 5749 4748 542E 5443 5020 2020 0000 0003 'WIGHT.TCP .....'  
000000B0: 0000 0010 0000 05E7 0000 0001 '.....'
```

MQGET of message number 4

****Message descriptor****

```
StrucId : 'MD ' Version : 1  
Report  : 0 MsgType : 2  
Expiry  : -1 Feedback : 0  
Encoding : 273 CodedCharSetId : 850  
Format  : 'MQADMIN ' '
```

Defining an alias queue

The following command creates an alias queue:

```
DEFINE QALIAS (MY.ALIAS.QUEUE) TARGQ (YELLOW.QUEUE)
```

This command redirects MQI calls that specify MY.ALIAS.QUEUE, to the queue YELLOW.QUEUE. The command does not create the target queue; the MQI calls fail if the queue YELLOW.QUEUE does not exist at run time.

If you change the alias definition, you can redirect the MQI calls to another queue. For example:

```
DEFINE QALIAS (MY.ALIAS.QUEUE) TARGQ (MAGENTA.QUEUE) REPLACE
```

This command redirects MQI calls to another queue, MAGENTA.QUEUE.

You can also use alias queues to make a single queue (the target queue) appear to have different attributes for different applications. You do this by defining two aliases, one for each application. Suppose there are two applications:

- Application ALPHA can put messages on YELLOW.QUEUE, but is not allowed to get messages from it.
- Application BETA can get messages from YELLOW.QUEUE, but is not allowed to put messages on it.

You can perform this action using the following commands:

```
* This alias is put enabled and get disabled for application ALPHA  
  
DEFINE QALIAS (ALPHAS.ALIAS.QUEUE) +  
  TARGQ (YELLOW.QUEUE) +  
  PUT (ENABLED) +  
  GET (DISABLED)  
  
* This alias is put disabled and get enabled for application BETA  
  
DEFINE QALIAS (BETAS.ALIAS.QUEUE) +  
  TARGQ (YELLOW.QUEUE) +  
  PUT (DISABLED) +  
  GET (ENABLED)
```

ALPHA uses the queue name ALPHAS.ALIAS.QUEUE in its MQI calls; BETA uses the queue name BETAS.ALIAS.QUEUE. They both access the same queue, but in different ways.

You can use the LIKE and REPLACE attributes when you define queue aliases, in the same way that you use them with local queues.

Using other commands with queue aliases

You can use the appropriate MQSC commands to display or alter queue alias attributes, or delete the queue alias object. For example:

```
* Display the queue alias' attributes
* ALL = Display all attributes

DISPLAY QUEUE (ALPHAS.ALIAS.QUEUE) ALL

* ALTER the base queue name, to which the alias resolves.
* FORCE = Force the change even if the queue is open.

ALTER QALIAS (ALPHAS.ALIAS.QUEUE) TARGQ(ORANGE.LOCAL.QUEUE) FORCE

* Delete this queue alias, if you can.

DELETE QALIAS (ALPHAS.ALIAS.QUEUE)
```

You cannot delete a queue alias if, for example, an application currently has the queue open or has a queue open that resolves to this queue. See the *MQSeries Command Reference* manual for more information about this and other queue alias commands.

Working with model queues

A queue manager creates a *dynamic queue* if it receives an MQI call from an application specifying a queue name that has been defined as a model queue. The name of the new dynamic queue is generated by the queue manager when the queue is created. A *model queue* is a template that specifies the attributes of any dynamic queues created from it.

Model queues provide a convenient method for applications to create queues as they are required.

Defining a model queue

You define a model queue with a set of attributes in the same way that you define a local queue. Model queues and local queues have the same set of attributes except that on model queues you can specify whether the dynamic queues created are temporary or permanent. (Permanent queues are maintained across queue manager restarts, temporary ones are not). For example:

Managing objects for triggering

```
DEFINE QMODEL (GREEN.MODEL.QUEUE) +
  DESCR('Queue for messages from application X') +
  PUT (DISABLED) +
  GET (ENABLED) +
  NOTRIGGER +
  MSGDLVSQ (FIFO) +
  MAXDEPTH (1000) +
  MAXMSGL (2000) +
  USAGE (NORMAL) +
  DEFTYPE (PERMDYN)
```

This command creates a model queue definition. From the DEFTYPE attribute, the actual queues created from this template are permanent dynamic queues.

Note: The attributes not specified are automatically copied from the SYSYSTEM.DEFAULT.MODEL.QUEUE default queue.

You can use the LIKE and REPLACE attributes when you define model queues, in the same way that you use them with local queues.

Using other commands with model queues

You can use the appropriate MQSC commands to display or alter a model queue's attributes, or delete the model queue object. For example:

```
* Display the model queue's attributes
* ALL = Display all attributes

DISPLAY QUEUE (GREEN.MODEL.QUEUE) ALL

* ALTER the model to enable puts on any
* dynamic queue created from this model.

ALTER QMODEL (BLUE.MODEL.QUEUE) PUT(ENABLED)

* Delete this model queue:

DELETE QMODEL (RED.MODEL.QUEUE)
```

Managing objects for triggering

MQSeries provides a facility for starting an application automatically when certain conditions on a queue are met. One example of the conditions is when the number of messages on a queue reaches a specified number. This facility is called *triggering* and is described in detail in the *MQSeries Application Programming Guide*. This section describes how to set up the required objects to support triggering on MQSeries for Tandem NSK.

Defining an application queue for triggering

An application queue is a local queue that is used by applications for messaging, through the MQI. Triggering requires a number of queue attributes to be defined on the application queue. Triggering itself is enabled by the *Trigger* attribute (TRIGGER in MQSC).

In this example, a trigger event is to be generated when there are 100 messages of priority five or greater on the local queue MOTOR.INSURANCE.QUEUE, as follows:

```
DEFINE QLOCAL (MOTOR.INSURANCE.QUEUE) +
  PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) +
  MAXMSGL (2000) +
  DEFPSIST (YES) +
  INITQ (MOTOR.INS.INIT.QUEUE) +
  TRIGGER +
  TRIGTYPE (DEPTH) +
  TRIGDPTH (100)+
  TRIGMPRI (5)
```

Where:

QLOCAL (MOTOR.INSURANCE.QUEUE)

Specifies the name of the application queue being defined.

PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)

Specifies the name of the application to be started by a trigger monitor program.

MAXMSGL (2000)

Specifies the maximum length of messages on the queue.

DEFPSIST (YES)

Specifies that messages are persistent on this queue.

INITQ (MOTOR.INS.INIT.QUEUE)

Is the name of the initiation queue on which the queue manager is to put the trigger message.

TRIGGER

Is the trigger attribute value.

TRIGTYPE (DEPTH)

Specifies that a trigger event is generated when the number of messages of the required priority (TRIMPRI) reaches the number specified in TRIGDPTH.

TRIGDPTH (100)

Specifies the number of messages required to generate a trigger event.

TRIGMPRI (5)

Is the priority of messages that are to be counted by the queue manager in deciding whether to generate a trigger event. Only messages with priority 5 or higher are counted.

Defining an initiation queue

When a trigger event occurs, the queue manager puts a trigger message on the initiation queue specified in the application queue definition. Initiation queues have no special settings, but you can use the following definition of the local queue MOTOR.INS.INIT.QUEUE for guidance:

```
DEFINE QLOCAL(MOTOR.INS.INIT.QUEUE) +
  GET (ENABLED) +
  NOSHARE +
  NOTRIGGER +
  MAXMSGL (2000) +
  MAXDEPTH (10)
```

Creating a process definition

Use the DEFINE PROCESS command to create a process definition. A process definition associates an application queue with the application that is to process messages from the queue. This is done through the PROCESS attribute on the application queue MOTOR.INSURANCE.QUEUE. The following MQSC command defines the required process, MOTOR.INSURANCE.QUOTE.PROCESS, identified in this example:

```
DEFINE PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) +
  DESCR ('Insurance request message processing') +
  APPLTYPE (NSK) +
  APPLICID ('$DATA1.TEST.IRMP01') +
  USERDATA ('open, close, 235')
```

Where:

MOTOR.INSURANCE.QUOTE.PROCESS
Is the name of the process definition.

DESCR ('Insurance request message processing')
Is the descriptive text of the application program to which the definition relates, following the keyword. This text is displayed when you use the DISPLAY PROCESS command. This can help you to identify what the process does. If you use spaces in the string, you must enclose the string in single quotes.

APPLTYPE(NSK)
Is the type of the application that runs on Tandem NSK

APPLICID ('\$DATA1.TEST.IRMP01')
Is the name of the application executable program on the local system.

USERDATA ('open, close, 235')
Is user-defined data, which can be used by the application.

Displaying your process definition

Use the DISPLAY PROCESS command, with the ALL keyword, to examine the results of your definition. For example:

```
DISPLAY PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) ALL

      24 : DISPLAY PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) ALL
AMQ8407: Display Process details.
      DESCR (Insurance request message processing)
      APPLICID ($DATA1.TEST.IRMP01)
      ENVRDATA ( )
      USERDATA (open, close, 235)
      PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)
      APPLTYPE (NSK)
```

USERDATA is a string representing the arguments passed to the triggered application. See the sample programs AMQSTRG0 and AMQINQA (in ZMQSSMPL subvolume) for examples of how to write trigger monitors and triggered applications.

You can also use the MQSC ALTER PROCESS to alter an existing process definition and DELETE PROCESS to delete a process definition.

Displaying your process definition

Chapter 7. Administering remote MQSeries objects

This chapter explains how to administer MQSeries objects on another queue manager. It also explains how you can use remote queue objects to control the destination of messages and reply messages.

It contains these sections:

- “Understanding channels and remote queuing”
- “Administering a remote queue manager” on page 126
- “Creating a local definition of a remote queue” on page 132
- “Using remote queue definitions as aliases” on page 135

For more information about channels, their attributes, and how to set them up, refer to the *MQSeries Intercommunication* book.

Understanding channels and remote queuing

Queue managers communicate with each other using channels. For example, if an application is to put a message on a queue managed by a remote queue manager, a channel must be set up between the two queue managers. The channel is defined to the queue managers at each end of the connection. Each channel is named and has a number of attributes that define, for example, the type of channel and the protocol to be used for communication.

Channels are used for sending messages between queue managers. These messages may originate from:

- User-written application programs that transfer data from one node to another.
- User-written administration applications that use PCFs.
- Queue managers sending:
 - Instrumentation event messages to another queue manager.
 - MQSC commands issued from a **runmqsc** command in indirect mode – where the commands are run on another queue manager.

Channels are unidirectional; that is, messages can be sent in one direction only. Channel definitions are made in complementary pairs, one at each end of the connection. For example, if one end is a sender, the other must be a receiver or a requester.

Channels are ‘linked’ to queue managers (and therefore the applications they serve) by transmission queues and remote queue definitions. A transmission queue is used to forward messages (through a channel) to another queue manager. A remote queue definition identifies a queue on another queue manager. To give you an idea of how these things can fit together:

- A remote queue definition specifies a transmission queue.
- A channel serves a transmission queue, which is specified when the channel is defined.

“Preparing channels and transmission queues for remote administration” on page 127 shows how to use these definitions to set up remote administration.

Administering a remote queue manager

You define a channel using the DEFINE CHANNEL MQSC command. Channels, their attributes, and how you use them in distributed queuing, are discussed at length in the *MQSeries Intercommunication* book. In this section, the examples concerned with channels use the default channel attributes unless otherwise specified.

Administering a remote queue manager

This section explains how to administer a remote queue manager from a local queue manager. You can implement remote administration from a local node using:

- MQSC commands
- PCF commands

Preparing the queues and channels is essentially the same for both methods. In this book, the examples show MQSC commands, because they are easier to understand. However, you can convert the examples to PCFs if you wish. For more information about writing administration programs using PCFs, see the *MQSeries Programmable System Management*.

In remote administration you send MQSC commands to a remote queue manager—either interactively or from a text file containing the commands. The remote queue manager may be on the same machine or, more typically, on a different machine. You can remotely administer queue managers in different MQSeries environments, including UNIX®, Tandem NSK, AS/400, MVS/ESA™, OS/2, and OS/390.

To implement remote administration, you must create certain objects. Unless you have specialized requirements, you should find that the default values (for example, for message length) are sufficient.

Preparing queue managers for remote administration

Figure 31 on page 127 shows the configuration of queue managers and channels that are required for remote administration. `source.queue.manager` is the *source* queue manager from which you can issue MQSC commands and to which the results of these commands (operator messages) are returned, if possible. `target.queue.manager` is the destination queue manager, which processes the commands and generates any operator messages.

Note: `source.queue.manager` must be the default queue manager on the machine you are using. For further information on creating a queue manager, see “`crtmqm` (Create queue manager)” on page 234.

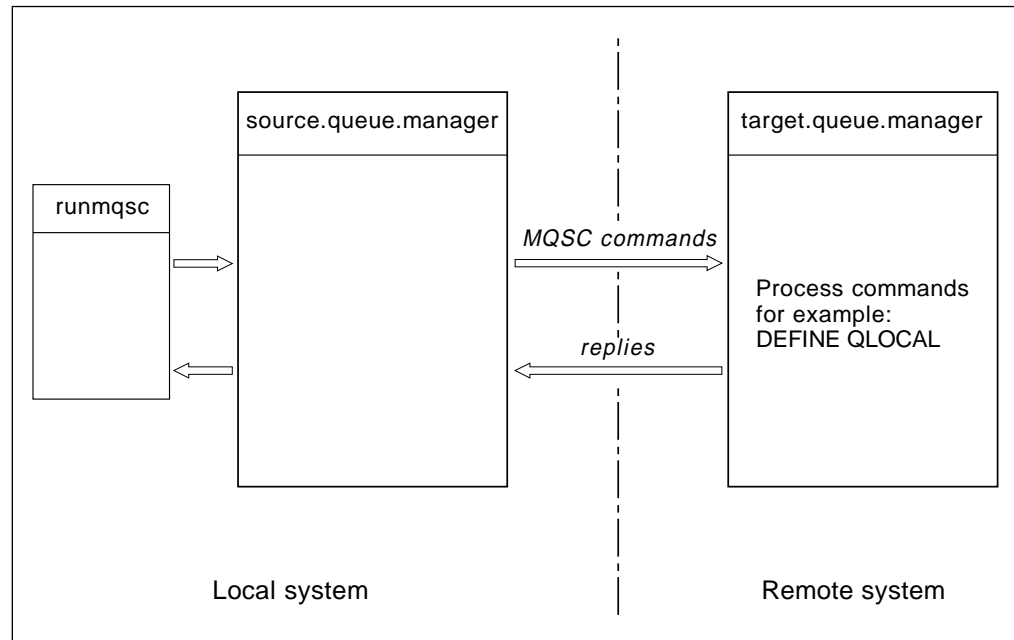


Figure 31. Remote administration

On both systems, if you have not already done so, you must:

- Create the queue manager, using the **crtmqm** command.
- Start the queue manager, using the **strmqm** command.
- Run the sample amqscoma, using the **runmqsc** command.

See “Creating the default and system objects” on page 74 for more information about these steps. You have to run these commands locally or over a network facility, for example Telnet.

On the destination queue manager:

- The command queue, SYSTEM.ADMIN.COMMAND.QUEUE, must be present. This queue is created by the sample MQSC command amqscoma.
- The command server must be started, using the **strmqcsv** command.

Preparing channels and transmission queues for remote administration

To run MQSC commands remotely, you must set up two channels, one for each direction, and their associated transmission queues. This example assumes that TCP/IP is being used as the transport type and that you know the TCP/IP address involved.

The channel `source.to.target` is for sending MQSC commands from the source queue manager to the destination. Its sender is at `source.queue.manager` and its receiver is at queue manager `target.queue.manager`. The channel `target.to.source` is for returning the output from commands and any operator messages that are generated to the source queue manager. You must also define a transmission queue for each sender. This queue is a local queue that is given the name of the receiving queue manager. Figure 32 on page 128 summarizes this configuration. However, you should be aware that the `SYSTEM.MQSC.REPLY.QUEUE` is the name of the model queue in `amqscoma`

Administering a remote queue manager

that is used by MQSC to develop its own dynamic reply queue. This queue name varies and is internal to MQSC.

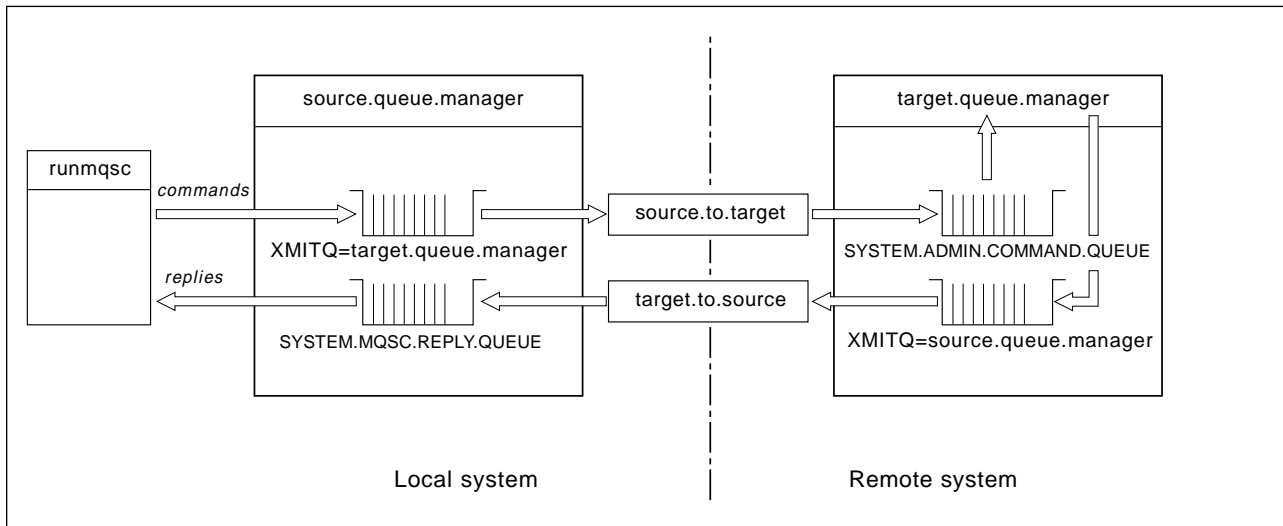


Figure 32. Setting up channels and queues for remote administration

See the *MQSeries Intercommunication* book for more information about setting up remote channels.

Defining channels and transmission queues

On the source queue manager, issue these MQSC commands to define the channels and the transmission queue:

```
* Define the sender channel at the source queue manager
DEFINE CHANNEL ('source.to.target') +
  CHLTYPE(SDR) +
  CONNAME ('198.210.60.37(1414)') +
  XMITQ ('target.queue.manager') +
  TRPTYPE(TCP)
  CONVERT(YES)

* Define the receiver channel at the source queue manager

DEFINE CHANNEL ('target.to.source') +
  CHLTYPE(RCVR) +
  TRPTYPE(TCP)

* Define the transmission queue on the source

DEFINE QLOCAL ('target.queue.manager') +
  USAGE (XMITQ)
```

Issue these commands on the destination queue manager (`target.queue.manager`), to create the channels and the transmission queue there:

```
* Define the sender channel on the destination queue manager
```

```
DEFINE CHANNEL ('target.to.source') +
  CHLTYPE(SDR) +
  CONNAME ('198.210.60.37(1414)') +
  XMITQ ('source.queue.manager') +
  TRPTYPE(TCP)
```

```
* Define the receiver channel on the destination queue manager
```

```
DEFINE CHANNEL ('source.to.target') +
  CHLTYPE(RCVR) +
  TRPTYPE(TCP)
```

```
* Define the transmission queue on the destination queue manager
```

```
DEFINE QLOCAL ('source.queue.manager') +
  USAGE (XMITQ)
```

Notes:

1. The TCP/IP connection names specified for the CONNAME attribute in the sender channel definitions are for illustration only. This is the IP address or network name of the machine at the other end of the connection. Use the values appropriate for your network.
2. On the sender channel, set the CONVERT attribute to YES if MQSeries for Tandem NSK is to exchange data with systems using different code pages. CONVERT(YES) specifies that the required data conversion between the systems is performed at the Tandem NSK end.

Starting the channels

The following description assumes that both ends of the channel are running on MQSeries for Tandem NSK. If this is not the case, refer to the relevant documentation for the non-Tandem NSK end of the channel.

To start the two channels, first ensure that the Tandem NSK TCP listener process has been configured for MQSeries on both nodes and are running at both ends of the connections. Then start the channels in **runmqsc**.

- On the source queue manager, enter:

```
start channel ('source.to.target')
```

- On the destination queue manager, enter:

```
start channel ('target.to.source')
```

Issuing MQSC commands remotely

The command server *must* be running on the destination queue manager, if it is going to process MQSC commands remotely. (This is not necessary on the source queue manager.)

- On the destination queue manager, type:

```
strmqcsv target.queue.manager
```

- On the source queue manager, you can then run MQSC interactively in queued mode by entering:

```
runmqsc -w 30 target.queue.manager
```

This form of the **runmqsc** command— with the **-w** flag—runs the MQSC commands in queued mode, where commands are put (in a modified form) on the command-server input queue and executed in order.

When you type in an MQSC command, it is redirected to the remote queue manager, in this case, `target.queue.manager`. The timeout is set to 30 seconds; if a reply is not received within 30 seconds, the following message is generated on the local (source) queue manager:

```
AMQ8416: MQSC timed out waiting for a response from the command server.
```

At the end of the MQSC session, the local queue manager displays any timed-out responses that have arrived. When the MQSC session is finished, any further responses are discarded.

In queued mode, you can also run an MQSC command file on a remote queue manager. For example:

```
runmqsc /IN mycmds, OUT report/ -w 60 target.queue.manager
```

where `mycmds` is a file containing MQSC commands and `report` is the report file.

Working with queue managers on MVS/ESA

You can issue MQSC commands to an MVS/ESA queue manager from an MQSeries for Tandem NSK queue manager. However, to do this, you must modify the **runmqsc** command and the channel definitions at the sender.

In particular, you add the **-x** flag to the **runmqsc** command on a Tandem NSK node:

```
runmqsc -w 30 -x QMRI
```


The channel definition is as follows:

```
* Define the sender channel at the source
queue manager on Tandem NSK

DEFINE CHANNEL ('source.to.target') +
  CHLTYPE(SDR) +
  CONNAME ('198.210.60.37(1414)') +
  XMITQ (QMRI) +
  TRPTYPE(TCP) +
  CONVERT (YES)
```

You must also define the receiver channel and the transmission queue at the source queue manager as before. Again, this example assumes that TCP/IP is the transmission protocol being used.

Recommendations for remote queuing

When you are implementing remote queuing:

1. Put the MQSC commands to be run on the remote system in a command file.
2. Verify your MQSC commands locally, by specifying the `-v` flag on the **runmqsc** command.

You cannot use **runmqsc** to verify MQSC commands on another queue manager.

3. Check, as far as possible, that the command file runs locally without error.
4. Finally, run the command file against the remote system.

If you have problems using MQSC remotely

If you have difficulty in running MQSC commands remotely, use the following checklist to see if you have:

- Started the command server on the destination queue manager.
- Defined a valid transmission queue.
- Defined the two ends of the message channels for both:
 - The channel along which the commands are being sent.
 - The channel along which the replies are to be returned.
- Specified the correct connection name (CONNAME) in the channel definition.
- Started the listeners before you started the message channels.
- Checked that the disconnect interval has not expired, for example, if a channel started but then shut down after some time. This is especially important if you start the channels manually.

See also “Troubleshooting MQSC” on page 106.

Creating a local definition of a remote queue

You can use a remote queue definition as a local definition of a remote queue. You create a remote queue object on your local queue manager to identify a local queue on another queue manager.

Understanding how local definitions of remote queues work

An application connects to a local queue manager and then issues an MQOPEN call. In the open call, the queue name specified is that of a remote queue definition on the local queue manager. The remote queue definition supplies the names of the destination queue, the destination queue manager, and optionally, a transmission queue. To put a message on the remote queue, the application issues an MQPUT call, specifying the handle returned from the MQOPEN call. The queue manager appends the remote queue name and the remote queue manager name to a transmission header in the message. This information is used to route the message to its correct destination in the network.

As administrator, you can control the destination of the message by altering the remote queue definition.

Example

An application is required to put a message on a queue owned by a remote queue manager.

How it works

The application connects to a queue manager, for example, saturn.queue.manager. The destination queue is owned by another queue manager.

On the MQOPEN call, the application specifies these fields in the MQOD:

Field value	Description
<i>ObjectName</i> CYAN.REMOTE.QUEUE	Specifies the local name of the remote queue object. This defines the destination queue and the destination queue manager.
<i>ObjectType</i> (Queue)	Identifies this object as a queue.
<i>ObjectQmgrName</i> Blank or saturn.queue.manager	This field is optional. If blank, the name of the local queue manager is assumed. (This is the queue manager on which the remote queue definition was made and to which the application is connected). If not blank, the name of the local queue manager must be specified.

After this, the application issues an MQPUT call to put a message on to this queue.

On the local queue manager, you can create a local definition of a remote queue using the following MQSC commands:

```
DEFINE QREMOTE ('CYAN.REMOTE.QUEUE') +
  DESCR ('Queue for auto insurance requests from the branches') +
  RNAME ('AUTOMOBILE.INSURANCE.QUOTE.QUEUE') +
  RQMNAME ('jupiter.queue.manager') +
  XMITQ ('INQUOTE.XMIT.QUEUE')
```

Where:

QREMOTE ('CYAN.REMOTE.QUEUE')

Is the local name of the remote queue object. This is the name that applications connected to this queue manager must specify in the MQOPEN call to open the queue AUTOMOBILE.INSURANCE.QUOTE.QUEUE on the remote queue manager jupiter.queue.manager.

DESCR ('Queue for auto insurance requests from the branches')

Is additional text that describes the use of the queue.

RNAME ('AUTOMOBILE.INSURANCE.QUOTE.QUEUE')

Is the name of the destination queue on the remote queue manager. This is the real destination queue for messages that are sent by applications that specify the queue name 'CYAN.REMOTE.QUEUE'. The queue 'AUTOMOBILE.INSURANCE.QUOTE.QUEUE' must be defined as a local queue on the remote queue manager.

RQMNAME ('jupiter.queue.manager')

Is the name of the remote queue manager that owns the destination queue 'AUTOMOBILE.INSURANCE.QUOTE.QUEUE'.

XMITQ ('INQUOTE.XMIT.QUEUE')

Is the name of the transmission queue. This is optional; if the name is not specified, a queue with the same name as the remote queue manager is used.

In either case, the appropriate transmission queue must be defined as a local queue with a *Usage* attribute specifying that it is a transmission queue (USAGE(XMITQ) in MQSC).

An alternative way of putting messages on a remote queue

Using a local definition of a remote queue is not the only way of putting messages on a remote queue. Applications can specify the full queue name, which includes the remote queue manager name, as part of the MQOPEN call. In this case, a local definition of a remote queue is not required. However, this alternative means that applications must either know or have access to the name of the remote queue manager at run time.

Using other commands with remote queues

You can use the appropriate MQSC commands to display or alter the attributes of a remote queue object, or you can delete the remote queue object. For example:

```
* Display the remote queue's attributes.  
* ALL = Display all attributes  
  
DISPLAY QUEUE (CYAN.REMOTE.QUEUE) ALL  
  
* ALTER the remote queue to enable puts.  
* This does not affect the destination queue,  
* only applications that specify this remote queue.  
  
ALTER QREMOTE (CYAN.REMOTE.QUEUE) PUT(ENABLED)  
  
* Delete this remote queue  
* This does not affect the destination queue  
* only its local definition  
  
DELETE QREMOTE (CYAN.REMOTE.QUEUE)
```

Note: If you delete a remote queue, you only delete the local representation of the remote queue. You do not delete the remote queue itself or any messages on it.

Creating a transmission queue

A transmission queue is a local queue that is used when a queue manager forwards messages to a remote queue manager through a message channel. The channel provides a one-way link to the remote queue manager. Messages are queued at the transmission queue until the channel can accept them. When you define a channel, you must specify a transmission queue name at the sending end of the message channel.

The *Usage* attribute (USAGE in MQSC) defines whether a queue is a transmission queue or a normal queue.

Default transmission queues

Optionally, you can specify a transmission queue in a remote queue object, using the *XmitQName* attribute (XMITQ in MQSC). If no transmission queue is defined, a default is used. When applications put messages on a remote queue, if a transmission queue with the same name as the destination queue manager exists, that queue is used. If this queue does not exist, the queue specified by the *DefaultXmitQ* attribute (DEFXMITQ in MQSC) on the local queue manager is used.

For example, the following MQSC command creates a default transmission queue on source.queue.manager for messages going to target.queue.manager:

```
DEFINE QLOCAL ('target.queue.manager') +
  DESCR ('Default transmission queue for target qm') +
  USAGE (XMITQ)
```

Applications can put messages directly on a transmission queue, or they can be put there indirectly, for example, through a remote queue definition. See also “Creating a local definition of a remote queue” on page 132.

Using remote queue definitions as aliases

In addition to locating a queue on another queue manager, you can also use a local definition of a remote queue for both:

- Queue manager aliases
- Reply-to queue aliases

Both types of alias are resolved through the local definition of a remote queue.

As usual in remote queuing, the appropriate channels must be set up if the message is to arrive at its destination.

Queue manager aliases

An alias is the process by which the name of the destination queue manager—as specified in a message—is modified by a queue manager on the message route. Queue manager aliases are important because you can use them to control the destination of messages within a network of queue managers.

You do this by altering the remote queue definition on the queue manager at the point of control. The sending application is not aware that the queue manager name specified is an alias.

For more information about queue manager aliases, see the *MQSeries Intercommunication* book.

Reply-to queue aliases

Optionally, an application can specify the name of a reply-to queue when it puts a *request message* on a queue. If the application that processes the message extracts the name of the reply-to queue, it knows where to send the *reply message*, if required.

A reply-to queue alias is the process by which a reply-to queue – as specified in a request message – is altered by a queue manager on the message route. The sending application is not aware that the reply-to queue name specified is an alias.

A reply-to queue alias lets you alter the name of the reply-to queue and optionally its queue manager. This in turn lets you control which route is used for reply messages.

For more information about request messages, reply messages, and reply-to queues, see the *MQSeries Application Programming Reference*. For more information about reply-to queue aliases, see the *MQSeries Intercommunication* book.

Aliases

Chapter 8. Implementing security control

This chapter explains the features of security control in MQSeries for Tandem NSK, and describes how you can implement security control.

This chapter contains these sections:

- “Understanding user IDs in the MQM user group”
- “Protecting MQSeries resources” on page 138
- “Understanding the Object Authority Manager (OAM)” on page 138
- “Using the Object Authority Manager (OAM) commands” on page 141
- “Access authorizations” on page 144
- “Display authority command” on page 144
- “Object Authority Manager (OAM) guidelines” on page 144
- “Understanding the authorization specification tables” on page 148
- “Understanding authorization files” on page 153

Understanding user IDs in the MQM user group

All queue manager resources run with the group ID MQM.

To be able to access MQSeries for Tandem NSK, your user ID must correspond to an MQSeries principal. Initially, only the user ID that created the queue manager has a corresponding MQSeries principal. You must create a principal for each user that will access MQSeries.

If your user ID belongs to the MQSeries for Tandem NSK group MQM, and an MQSeries principal has been created for your user ID, you have all authorities to all MQSeries resources. Your user ID *must* belong to the MQM group to be able to use all the MQSeries for Tandem NSK control commands (except **crtmqcvx**). In particular, you need this authority to:

- Use the **runmqsc** command to run MQSCs.
- Administer authorities on MQSeries for Tandem NSK using the **setmqaut** command.

If you are sending channel commands to queue managers on a remote Tandem NSK system, you must ensure that your user ID is a member of Tandem NSK group MQM on the target system. For a list of PCF and MQSC channel commands, see “Channel command security” on page 147

It is not essential for your user ID to belong to group MQM for issuing:

- PCF commands— including Escape PCFs—from an administration program
- MQI calls from an application program

Getting additional information

For more information about:

- MQSeries for Tandem NSK command sets, see Chapter 4, “Using administration command sets” on page 43
- MQSeries for Tandem NSK control commands, see Chapter 15, “The MQSeries control commands” on page 217

- PCF commands and Escape PCFs, see the *MQSeries Programmable System Management* manual
- MQI calls, see the *MQSeries Application Programming Guide* and *MQSeries Application Programming Reference* manuals

Protecting MQSeries resources

Because MQSeries queue managers handle the transfer of information that is potentially valuable, you need the safeguard of an authority system. This step ensures that the resources that a queue manager owns and manages are protected from unauthorized access, which could lead to the loss or disclosure of the information. In a secure system, it is essential that none of the following are accessed or changed by any unauthorized user or application:

- Connections to a queue manager.
- Access to MQSeries objects such as queues, channels, and processes.
- Commands for queue manager administration, including MQSCs and PCF commands.
- Access to MQSeries messages.
- Context information associated with messages.

You should develop your own policy with respect to which users have access to which resources.

Understanding the Object Authority Manager (OAM)

By default, access to queue manager resources is controlled through an authorization service installable component. The authorization service component supplied with MQSeries for Tandem NSK is called the OAM and is automatically installed and enabled for each queue manager you create, unless you specify otherwise. In this chapter, the term OAM is used to denote the Object Authority Manager supplied with this product.

The OAM is an *installable component* of the authorization service. Providing the OAM as an installable component gives you the flexibility to:

- Replace the supplied OAM with your own authorization service component using the interface provided.
- Augment the facilities supplied by the OAM with those of your own authorization service component, again using the interface provided.
- Remove or disable the OAM and run with no authorization service at all.

For more information on installable services, see the *MQSeries Programmable System Management* manual.

The OAM manages users' authorizations to manipulate MQSeries objects, including queues, process definitions, and channels. It also provides a command interface through which you can grant or revoke access authority to an object for a specific group of users. The decision to allow access to a resource is made by the OAM, and the queue manager follows that decision. If the OAM cannot make a decision, the queue manager prevents access to that resource.

How the OAM works

The OAM uses the user and group IDs and security features of the Tandem NSK operating system. Users can access queue manager objects only if they have the required authority.

Managing access through user groups

Managing access permissions to MQSeries resources is based on Tandem NSK *groups*. The OAM maintains authorizations at the group level.

In the command interfaces, MQSeries principals are used rather than user IDs. The reason for this is that authorities granted to a user ID can also be granted to other entities. For example, authorities can be granted to an application program that issues MQI calls, or to an administration program that issues PCF commands. In these cases the principal associated with the program is not necessarily the user ID that was used when the program was started.

Tandem NSK user IDs may have the form <group>.<name> where both group and name may be up to 8 characters each, whereas MQSeries principal names can be up to 12 characters. In addition, the period character (.) is illegal in user IDs on some other platforms. In MQSeries for Tandem NSK, the principal database contains mappings of Tandem NSK user IDs to MQSeries principal names of 12 characters or fewer.

When a user belongs to more than one user group

The authorization that a user has is the union of the authorizations of all the groups to which the user belongs and the default authorization for all users. You can use the control command **setmqaut** to set the authorizations for a specific group.

Note: Any changes made using the **setmqaut** command take immediate effect, unless the object is in use. In this case, the change occurs when the object is next opened.

Group sets and the primary group

Management of access permissions to MQSeries resources is based on Tandem NSK user groups. When SAFEGUARD is running, a Tandem NSK user ID can be associated with more than one group, and therefore the corresponding MQSeries principal is also associated with these groups. The primary group is always the Tandem Administrative Group. Secondary groups are configured by creating SAFEGUARD File Sharing Groups, and associating a Tandem NSK user ID with that File Sharing Group.

The OAM maintains authorizations at the level of groups rather than individual principals. The mapping of principals to group names is carried out within in the OAM using the principal database and the Tandem NSK and SAFEGUARD facilities; OAM operations are carried out at the group level. You can, however, display the authorizations of an individual principal.

Protecting resources with the OAM

Through OAM you can control:

- Access to MQSeries objects through the MQI. When an application program attempts to access an object, the OAM checks if the user ID making the request has the authorization (through its user group) for the operation requested.

In particular, this means that queues, and the messages on queues, can be protected from unauthorized access.
- Permission to use MQSC commands; only members of user group mqm, or those authorized via **setmqaut**, can execute queue manager administration commands, for example, to create a queue.
- Permission to use control commands; only members of user group mqm can execute control commands, for example, creating a queue manager or starting a command server.
- Permission to use PCF commands.

Different groups of users can be granted different kinds of access authority to the same object. For example, for a specific queue, one group might be allowed to perform both put and get operations; another group can only be allowed to browse the queue (MQGET with browse option). Similarly, some groups might have get and put authority to a queue, but are not allowed to alter or delete the queue.

Using groups for authorizations

Using groups rather than individual principals for authorization reduces the amount of administration required. Typically, a particular kind of access is required by more than one principal. For example, you might define a group consisting of end users who want to run a particular application. New users can be given access by adding the appropriate group to their Tandem NSK user ID. Unless MQSeries is installed on a system using SAFEGUARD to create data sharing groups, each user ID can be associated with a single, primary group only.

You should keep the number of groups as small as possible. For example, you can divide users into one group for application users and one for administrators.

Disabling the Object Authority Manager (OAM)

By default, the OAM is enabled. You can disable the OAM by setting the Tandem NSK environment variable MQSNOAUT before the queue manager is created, as follows:

```
PARAM MQSNOAUT 1
```

However, if you disable the OAM for a queue manager, you cannot restart the OAM later. You might want to have the OAM enabled and ensure that all users and applications have access through an appropriate user ID. You can also disable the OAM for testing purposes only either by removing the authorization service stanza in the queue manager configuration file (QMINI) or by setting MQAUTH off in the Authority stanza of QMINI, as described in “Queue manager configuration file (QMINI)” on page 185.

Note: Specifying PARAM MQSN0AUT 0 does not enable the OAM. The environment variable must not exist in the environment if the OAM is to be reenabled.

Using the Object Authority Manager (OAM) commands

The OAM provides a command interface for granting and revoking authority. Before you can use these commands, you must be authorized— your user ID must belong to the Tandem NSK MQM group. This group should have been set up before you installed the product. (See “Preparing for installation” on page 20 for more information.)

If your user ID is a member of group MQM, you have a ‘super user’ authority to the queue manager. You are now authorized to issue any MQI request or control command from your user ID.

The OAM provides four commands that you can invoke from TACL to manage the authorizations of users. These are:

- **altmqusr** (Create, remove, or alter an MQSeries principal)
- **dspmqusr** (Display principal)
- **setmqaut** (Set or reset authority)
- **dspmqaut** (Display authority)

Authority checking occurs in the following calls: MQCONN, MQOPEN, MQPUT1, and MQCLOSE. Authority checking is only performed at the first instance of any of these calls, and authority is not amended until you reset (that is, close and reopen) the object. Therefore, any changes made to the authority of an object using **setmqaut** do not take effect until you reset the object.

What to specify when using the OAM commands

The OAM commands apply to the specified queue manager; if you do not specify a queue manager, the default queue manager is used. On these commands, you must specify the object uniquely, that is, you must specify the object name and its type. You also have to specify the user or group name to which the authority applies.

Authorization lists

You specify a list of authorizations with **setmqaut** command. This is a quick way of specifying whether authorization is to be granted or revoked, and which resources in which the authorization applies. Each authorization in the list is specified as a lowercase keyword, prefixed with a plus sign (+) or a minus sign (-). You can use a plus sign to add the specified authorization or a minus sign to remove the authorization. You can specify any number of authorizations in a single command. For example:

```
+browse -get +put
```

Using the **altmqsr** command

Provided you have the required authorization, you can use the **altmqsr** command to create an MQSeries principal and associate it with a Tandem NSK user ID (or SAFEGUARD alias). The following example shows how the **altmqsr** command is used:

```
altmqsr -m saturn.queue.manager -p MQPRINCIPAL -u MQM.MQUSER
```

In this example:

This term...	Specifies the...
saturn.queue.manager	Queue manager name
MQPRINCIPAL	Principal name to be created
MQM.MQUSER	Tandem NSK user ID

See “altmqsr (Alter MQSeries user information)” on page 223 for a description of this command.

The **altmqsr** command can also be used to remove a principal (and thereby revoke all access rights to MQSeries). For example:

```
altmqsr -m saturn.queue.manager -p MQPRINCIPAL -remove
```

In this example:

This term...	Specifies the...
saturn.queue.manager	Queue manager name
MQPRINCIPAL	Principal name to be removed
-remove	Instruction to delete the principal

Using the **dspmqsqr** command

You can display the contents of the principal database, in addition to the Tandem NSK administrative and file-sharing groups that the user ID corresponding to each MQSeries principal belongs to, using the **dspmqsqr** command. The **-p** parameter restricts the information displayed to the specified principal. For example:

```
dspmqsqr -m saturn.queue.manager -p MQPRINCIPAL
```

In this example:

This term...	Specifies the...
saturn.queue.manager	Queue manager name
MQPRINCIPAL	Principal name to be displayed

See “dspmqsqr (Display MQSeries user information)” on page 249 for a description of this command.

Using the setmqaut command

Provided you have the required authorization, you can use the **setmqaut** command to grant or revoke authorization of a principal or user group to access a particular object. The following example shows how the **setmqaut** command is used:

```
setmqaut -m saturn.queue.manager -t queue -n RED.LOCAL.QUEUE -g GroupA +browse -get +put
```

In this example:

This term....	Specifies the....
saturn.queue.manager	Queue manager name
queue	Object type
RED.LOCAL.QUEUE	Object name
GroupA	ID of the group to be given the authorizations
+browse -get +put	Authorization list for the specified queue. There must be no spaces between the "+" or "-" signs and the keyword.

The authorization list specifies the authorizations to be given, where:

This term...	Specifies...
+browse	Add authorization to browse (MQGET with browse option)
-get	Remove authorization to get (MQGET) messages from the queue.
+put	Add authorization to put (MQPUT) messages on the queue.

Applications started with user IDs that belong to Tandem NSK user group GroupA have these authorizations.

The following command revokes put authority on the queue MyQueue to groups GroupA and GroupB.

```
setmqaut -m saturn.queue.manager -t queue -n MyQueue -g GroupA -g GroupB -put
```

For a formal definition of the command and its syntax, see “setmqaut (Set/reset authority)” on page 265

Authority commands and installable services

The **setmqaut** command takes an additional parameter that specifies the name of the authorization service component to which the update applies. You must specify this parameter if you have multiple authorization components running at the same time. By default, this is not the case. If the parameter is omitted, the update is made to the first authorization component it finds, if one exists. By default, this is the supplied OAM.

Access authorizations

Authorizations defined by the authorization list associated with the **setmqaut** command can be categorized as follows:

- Authorizations related to MQI calls
- Authorization related administration commands
- Context authorizations
- General authorizations, that is, for MQI calls, for commands, or both

Each authorization is specified by a keyword used with the **setmqaut** and **dspmqaut** commands. These are described in “**setmqaut (Set/reset authority)**” on page 265

Display authority command

You can use the command **dspmqaut** to view the authorizations that a specific principal or group has for a particular object. The flags have the same meaning as those in the **setmqaut** command. Authorization can be displayed for only one group or principal at a time. See “**dspmqaut (Display authority)**” on page 240 for a formal specification of this command.

For example, the following command displays the authorizations that the group **GpAdmin** has to a process definition named **Annuities** on queue manager **QueueMan1**.

```
dspmqaut -m QueueMan1 -t process -n Annuities -g GpAdmin
```

The keywords displayed as a result of this command identify the authorizations that are active.

Object Authority Manager (OAM) guidelines

Some operations are particularly sensitive and should be limited to privileged users. For example:

- Creating, deleting, starting, and stopping queue managers
- Accessing certain special queues, such as transmission queues or the command queue **SYSTEM.ADMIN.COMMAND.QUEUE**
- Programs that use full MQI context options
- Creating and copying application queues

User IDs

The special group called **MQM** that you create is intended for use by product administrators only. It should never be available to nonprivileged users.

Queue manager volumes

The volume containing queues and other queue manager data is private to the product. Objects in this directory have Tandem NSK user authorizations that relate to their OAM authorizations. Standard Tandem NSK commands cannot be used to grant or revoke authorizations to MQI resources because:

- MQSeries objects are not necessarily the same as the corresponding system object name. See “Volume structure” on page 70 for more information about this.
- MQSeries objects do not necessarily map to the object’s NSK security settings.

Queues

The authority to access a dynamic queue is based on—but not necessarily the same as—that of the model queue from which it is derived.

For alias queues and remote queues, the authorization is that of the object itself, not the queue to which the alias or remote queue resolves. It is, therefore, possible to authorize a principal to access an alias queue that resolves to a local queue to which the principal has no access permissions.

You should limit the authority to create queues to privileged users. If you do not limit this authority, users can bypass the normal access control by creating an alias.

Alternate user authority

Alternate user authority controls whether one user ID can use the authority of another user ID when accessing an MQSeries object. This method is essential when a server receives requests from a program and the server needs to ensure that the program has the required authority for the request. The server can have the required authority, but it needs to know whether the program has the authority for the actions it has requested.

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, you can use alternate user authority to control whether PAYSERV is allowed to specify USER1 as an alternate user ID when it opens the reply-to queue.

The alternate user ID is specified on the *AlternateUserId* field of the object descriptor.

Both the user ID and the alternate user IDs must be specified as principals corresponding to entries in the principal database associated with a Tandem NSK user ID for authorization to be granted.

Note: You can use alternate user IDs on any MQSeries object. Use of an alternate user ID does not affect the user ID used by any other resource managers.

Context authority

Context is information that applies to a particular message and is contained in the message descriptor, MQMD, which is part of the message. The context information comes in two sections:

Identity section This part specifies who the message came from. It consists of the following fields:

- *UserIdentifier*
- *AccountingToken*
- *ApplIdentityData*

Origin section This section specifies where the message came from, and when it was put onto the queue. It consists of the following fields:

- *PutApplType*
- *PutApplName*
- *PutDate*
- *PutTime*
- *ApplOriginData*

Applications can specify the context data when either an MQOPEN or an MQPUT call is made. This data can be generated by the application, it can be passed on from another message, or it can 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, testing whether the message came from an application, running under an authorized user ID.

A server program can use the *UserIdentifier* to determine the user ID of an alternate user. The *UserIdentifier* must be specified as a principal corresponding to an entry in the principal database.

You use context authorization 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.

Remote security considerations

For remote security, you should consider:

Put authority For security across queue managers you can specify the put authority that is used when a channel receives a message sent from another queue manager.

Specify the channel attribute PUTAUT as follows:

DEF Default user ID. The user ID that the message channel agent is running under.

CTX The user ID in the message context.

In both cases, the user ID must be specified as a principal corresponding to an entry in the principal database.

Transmission queues

Queue managers automatically put remote messages on a transmission queue; no special authority is required. However, putting a message directly on a transmission queue requires special authorization; see Table 7 on page 149

Channel exits Channel exits can be used for added security.

For more information, see the *MQSeries Intercommunication* book.

Channel command security

Channel commands can be issued as PCF commands, MQSC commands, and control commands.

PCF commands

You can issue PCF channel commands by sending a PCF message to the SYSTEM.ADMIN.COMMAND.QUEUE on a remote Tandem NSK system. The user ID as specified in the message descriptor of the PCF message must be specified as a principal corresponding to an entry in the principal database associated with a Tandem NSK user ID belonging to the mqm group on the target system. These commands are:

- *ChangeChannel*
- *CopyChannel*
- *CreateChannel*
- *DeleteChannel*
- *PingChannel*
- *ResetChannel*
- *StartChannel*
- *StopChannel*
- *ResolveChannel*

See the *MQSeries Programmable System Management* manual for the PCF security requirements.

MQSC channel commands

You can issue MQSC channel commands to a remote Tandem NSK system either by sending the command directly in a PCF escape message or by issuing the command using **runmqsc** in indirect mode. The user ID as specified in the message descriptor of the PCF message must be specified as a principal corresponding to an entry in the principal database associated with a Tandem NSK user ID belonging to the mqm group on the target system. (PCF commands are implicit in MQSC commands issued from **runmqsc** in indirect mode.) These commands are:

- ALTER CHANNEL
- DEFINE CHANNEL
- DELETE CHANNEL
- PING CHANNEL
- RESET CHANNEL
- START CHANNEL
- START CHINIT
- STOP CHANNEL
- RESOLVE CHANNEL

Authorization specification tables

For MQSC commands issued from the **runmqsc** command, the user ID in the PCF message is normally that of the current user.

Understanding the authorization specification tables

The authorization specification tables starting on page 149 define precisely how the authorizations work and the restrictions that apply. The tables apply to these situations:

- Applications that issue MQI calls.
- Administration programs that issue MQSC commands as escape PCFs.
- Administration programs that issue PCF commands.

In this section, the information is presented as a set of tables that specify the following:

Action to be performed	MQI option, MQSC command, or PCF command.
Access control object	Queue, process, or queue manager.
Authorization required	Expressed as an 'MQZAO_' constant.

In the tables, the constants prefixed by MQZAO_ correspond to the keywords in the authorization list for the **setmqaut** command for the particular entity. For example, MQZAO_BROWSE corresponds to the keyword +browse; similarly, the keyword MQZAO_SET_ALL_CONTEXT corresponds to the keyword +setall and so on. These constants are defined in the header file CMQZCH in subvolume ZMQSLIB, which is supplied with the product.

MQI authorizations

An application is allowed to issue certain MQI calls and options only if the user identifier under which it is running (or whose authorizations it is able to assume) has been granted the relevant authorization.

Four MQI calls may require authorization checks: MQCONN, MQOPEN, MQPUT1, and MQCLOSE.

For MQOPEN and MQPUT1, the authority check is made on the name of the object being opened, and not on the name, or names, resulting after a name has been resolved. For example, an application may be granted authority to open an alias queue without having authority to open the base queue to which the alias resolves. The rule is that the check is carried out on the first definition encountered during the process of name resolution that is not a queue-manager alias, unless the queue-manager alias definition is opened directly; that is, its name appears in the *ObjectName* field of the object descriptor. Authority is always needed for the particular object being opened; in some cases additional queue-independent authority—which is obtained through an authorization for the queue-manager object—is required.

Table 7 on page 149 summarizes the authorizations needed for each call.

<i>Table 7. Security authorization needed for MQI calls</i>			
Authorization required for :	Queue object (1)	Process object	Queue manager object
MQCONN option	Not applicable	Not applicable	MQZAO_CONNECT
MQOPEN Option			
MQOO_INQUIRE	MQZAO_INQUIRE (2)	MQZAO_INQUIRE (2)	MQZAO_INQUIRE (2)
MQOO_BROWSE	MQZAO_BROWSE	Not applicable	No check
MQOO_INPUT_*	MQZAO_INPUT	Not applicable	No check
MQOO_SAVE_ ALL_CONTEXT (3)	MQZAO_INPUT	Not applicable	No check
MQOO_OUTPUT (Normal queue) (4)	MQZAO_OUTPUT	Not applicable	No check
MQOO_PASS_ IDENTITY_CONTEXT (5)	MQZAO_PASS_ IDENTITY_ CONTEXT	Not applicable	No check
MQOO_PASS_ ALL_CONTEXT (5, 6)	MQZAO_PASS_ ALL_CONTEXT	Not applicable	No check
MQOO_SET_ IDENTITY_CONTEXT (5, 6)	MQZAO_SET_ IDENTITY_ CONTEXT	Not applicable	MQZAO_SET_ IDENTITY_ CONTEXT (7)
MQOO_SET_ ALL_CONTEXT (5, 8)	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT (7)
MQOO_OUTPUT (Transmission queue) (9)	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT (7)
MQOO_SET	MQZAO_SET	Not applicable	No check
MQOO_ALTERNATE_ USER_AUTHORITY	(10)	(10)	MQZAO_ALTERNATE_ USER_ AUTHORITY (10, 11)
MQPUT1 Option			
MQPMO_PASS_ IDENTITY_CONTEXT	MQZAO_PASS_ IDENTITY_ CONTEXT (12)	Not applicable	No check
MQPMO_PASS_ ALL_CONTEXT	MQZAO_PASS_ ALL_CONTEXT (12)	Not applicable	No check
MQPMO_SET_ IDENTITY_CONTEXT	MQZAO_SET_ IDENTITY_ CONTEXT (12)	Not applicable	MQZAO_SET_ IDENTITY_ CONTEXT (7)
MQPMO_SET_ ALL_CONTEXT	MQZAO_SET_ ALL_CONTEXT (12)	Not applicable	MQZAO_SET_ ALL_CONTEXT (7)
(Transmission queue) (9)	MQZAO_SET_ ALL_CONTEXT	Not applicable	MQZAO_SET_ ALL_CONTEXT (7)
MQPMO_ALTERNATE_ USER_AUTHORITY	(13)	Not applicable	MQZAO_ALTERNATE_ USER_ AUTHORITY (11)
MQCLOSE Option			
MQCO_DELETE	MQZAO_DELETE (14)	Not applicable	Not applicable
MQCO_DELETE_PURGE	MQZAO_DELETE (14)	Not applicable	Not applicable

Specific notes:

1. If a model queue is being opened:
 - MQZAO_DISPLAY authority is needed for the model queue, in addition to whatever other authorities (also for the model queue) are required for the open options specified.
 - MQZAO_CREATE authority is not needed to create the dynamic queue.

Authorization specification tables

- The user identifier used to open the model queue is automatically granted all of the queue-specific authorities (equivalent to MQZAO_ALL) for the dynamic queue created.
2. Either the queue, process, or queue manager object is checked, depending on the type of object being opened.
 3. MQOO_INPUT_* must also be specified. This is valid for a local, model, or alias queue.
 4. This check is performed for all output cases, except the case specified in note 9.
 5. MQOO_OUTPUT must also be specified.
 6. MQOO_PASS_IDENTITY_CONTEXT is also implied by this option.
 7. This authority is required for both the queue manager object and the particular queue.
 8. MQOO_PASS_IDENTITY_CONTEXT, MQOO_PASS_ALL_CONTEXT, and MQOO_SET_IDENTITY_CONTEXT are also implied by this option.
 9. This check is performed 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 (either by specifying the names of the remote queue manager and remote queue, or by specifying the name of a local definition of the remote queue).
 10. At least one of MQOO_INQUIRE (for any object type), or (for queues) MQOO_BROWSE, MQOO_INPUT_*, MQOO_OUTPUT, or MQOO_SET must also be specified. The check carried out is as for the other options specified, using the supplied alternate user identifier for the specific-named object authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
 11. This authorization allows any *AlternateUserId* to be specified.
 12. An MQZAO_OUTPUT check is also carried out, if the queue does not have a *Usage* queue attribute of MQUS_TRANSMISSION.
 13. The check carried out is as for the other options specified, using the supplied alternate user identifier for the specific-named queue authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
 14. The check is carried out only if both of the following are true:
 - A permanent dynamic queue is being closed and deleted.
 - The queue was not created by the MQOPEN which returned the object handle being used.Otherwise, there is no check.

General notes:

1. The special authorization MQZAO_ALL_MQI includes all of the following that are relevant to the object type:
 - MQZAO_CONNECT
 - MQZAO_INQUIRE
 - MQZAO_SET
 - MQZAO_BROWSE
 - MQZAO_INPUT

- MQZAO_OUTPUT
 - MQZAO_PASS_IDENTITY_CONTEXT
 - MQZAO_PASS_ALL_CONTEXT
 - MQZAO_SET_IDENTITY_CONTEXT
 - MQZAO_SET_ALL_CONTEXT
 - MQZAO_ALTERNATE_USER_AUTHORITY
2. MQZAO_DELETE (see note 14 on page 150) and MQZAO_DISPLAY are classed as administration authorizations. They are not therefore included in MQZAO_ALL_MQI.
 3. 'No check' means that no authorization checking is carried out.
 4. 'Not applicable' means that authorization checking is not relevant to this operation. For example, you cannot issue an MQPUT call to a process object.

Administration authorizations

These authorizations allow a user to issue administration commands. This can be an MQSC command as an escape PCF message or as a PCF command itself. These methods allow a program to send an administration command as a message to a queue manager, for execution on behalf of that user.

Authorizations for MQSC commands in escape PCFs

Table 8 summarizes the authorizations needed for each MQSC command that is contained in Escape PCF.

(2) Authorization required for:	Queue object	Process object	Queue manager object
MQSC command			
ALTER object	MQZAO_CHANGE	MQZAO_CHANGE	MQZAO_CHANGE
CLEAR QLOCAL	MQZAO_CLEAR	Not applicable	Not applicable
DEFINE object NOREPLACE (3)	MQZAO_CREATE (4)	MQZAO_CREATE (4)	Not applicable
DEFINE object REPLACE (3, 5)	MQZAO_CHANGE	MQZAO_CHANGE	Not applicable
DELETE object	MQZAO_DELETE	MQZAO_DELETE	Not applicable
DISPLAY object	MQZAO_DISPLAY	MQZAO_DISPLAY	MQZAO_DISPLAY

Specific notes:

1. The user identifier, under which the program (for example, **runmqsc**) which submits the command is running, must also have MQZAO_CONNECT authority to the queue manager.
2. Either the queue, process, or queue manager object is checked, depending on the type of object.
3. For DEFINE commands, MQZAO_DISPLAY authority is also needed for the LIKE object if one is specified, or on the appropriate SYSTEM.DEFAULT.xxx object if LIKE is omitted.
4. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects, for a specified queue manager, by specifying an object type of QMGR on the SETMQAUT command.
5. This applies if the object to be replaced does in fact already exist. If it does not, the check is as for DEFINE object NOREPLACE.

Authorization specification tables

General notes:

1. To perform any PCF command, you must have DISPLAY authority on the queue manager.
2. The authority to execute an escape PCF depends on the MQSC command within the text of the escape PCF message.
3. 'Not applicable' means that authorization checking is not relevant to this operation. For example, you cannot issue a CLEAR QLOCAL on a queue manager object.

Authorizations for PCF commands

Table 9 summarizes the authorizations needed for each PCF command.

(2) Authorization required for:	Queue object	Process object	Queue manager object
PCF command			
Change object	MQZAO_CHANGE	MQZAO_CHANGE	MQZAO_CHANGE
Clear Queue	MQZAO_CLEAR	Not applicable	Not applicable
Copy object (without replace) (3)	MQZAO_CREATE (4)	MQZAO_CREATE (4)	Not applicable
Copy object (with replace) (3, 6)	MQZAO_CHANGE	MQZAO_CHANGE	Not applicable
Create object (without replace) (5)	MQZAO_CREATE (4)	MQZAO_CREATE (4)	Not applicable
Create object (with replace) (5, 6)	MQZAO_CHANGE	MQZAO_CHANGE	Not applicable
Delete object	MQZAO_DELETE	MQZAO_DELETE	Not applicable
Inquire object	MQZAO_DISPLAY	MQZAO_DISPLAY	MQZAO_DISPLAY
Inquire object names	No check	No check	No check
Reset queue statistics	MQZAO_DISPLAY and MQZAO_CHANGE	Not applicable	Not applicable

Specific notes:

1. The user identifier under which the program submitting the command is running must also have authority to connect to its local queue manager, and to open the administration command queue for output.
2. Either the queue, process, or queue-manager object is checked, depending on the type of object.
3. For Copy commands, MQZAO_DISPLAY authority is also needed for the From object.
4. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects, for a specified queue manager, by specifying an object type of QMGR on the SETMQAUT command.
5. For Create commands, MQZAO_DISPLAY authority is also needed for the appropriate SYSTEM.DEFAULT.* object.
6. This applies if the object to be replaced already exists. If it does not, the check is as for Copy or Create without replace.

General notes:

1. To perform any PCF command, you must have DISPLAY authority on the queue manager.
2. The special authorization MQZAO_ALL_ADMIN includes all of the following that are relevant to the object type:
 - MQZAO_CHANGE
 - MQZAO_CLEAR
 - MQZAO_DELETE
 - MQZAO_DISPLAY

MQZAO_CREATE is not included, because it is not specific to a particular object or object type.
3. 'No check' means that no authorization checking is carried out.
4. 'Not applicable' means that authorization checking is not relevant to this operation. For example, you cannot use a Clear Queue command on a process object.

Understanding authorization files

For MQSeries for Tandem NSK V2.2.0.1, all authorization information is stored in the following TM/MP audited files in location \$VOL.<QmgrSubVol>D:

OAMDB The OAM Database
PRIDB The principal database
PRIDBA The principal database alternate key file

The principal database

Each record in the principal database maps a Tandem NSK user ID to a principal name. The principal database is an ENSCRIBE key-sequenced file that provides a mapping between the OAM principals and Tandem NSK logon IDs.

OAM Principal MQPRINCIPAL
 Tandem logon ID 0x2CFF

The primary key is the OAM principal (12 characters). The alternate key is the Tandem logon ID (a 2-byte short integer). The OAM principal is always case sensitive. The bytes of the logon ID field are <group>.<user>; the example above is for Tandem NSK ID (44,255).

The OAM Database

Each record in the new OAM authorizations database refers to a specific queue manager object, or class of object. The primary key is the object name plus object type. The records are of variable length, and the record layout is as follows:

Object Name	Type	#Auth Entries	Auth Entries
QUEUE.AUTH	1	2	PAYROLL 0x00000004, ADMIN 0xFFFFFFFF

Authorization files

The Object Name field is the full 48-character, blank-filled object name. The type field (4 bytes) differentiates between the types of MQSeries object, and the classes of object required by the OAM.

The Type field may take the following values:

- 1 Queue name
- 2 Process name
- 4 Queue manager name
- 128 Class

The #Auth Entries field (4 bytes) specifies the number of individual authorizations in the Auth Entries field in this record. Each of the Auth Entries specifies a group name and the authorization for that group for this object:

```
Group Name   PAYROLL
Auth         0x00000004
```

The Group Name field is 12 bytes in length and contains a blank-filled Tandem NSK Administrative or SAFEGUARD File-sharing Group Name (first 12 characters only). The Auth field is a 4-byte (ULONG) bit mask with the authority for the group. There may be up to 250 individual Group and Auth pairs in each record.

Multiple records for the same object are used to hold authorization information for more than 250 groups if necessary.

The authority specification is the union of the individual bit patterns based on the following assignments:

Authorization keyword	Formal name	Hexadecimal Value
connect	MQZAO_CONNECT	0x00000001
browse	MQZAO_BROWSE	0x00000002
get	MQZAO_INPUT	0x00000004
put	MQZAO_OUTPUT	0x00000008
inq	MQZAO_INQUIRE	0x00000010
set	MQZAO_SET	0x00000020
passid	MQZAO_PASS_IDENTITY_CONTEXT	0x00000040
passall	MQZAO_PASS_ALL_CONTEXT	0x00000080
setid	MQZAO_SET_IDENTITY_CONTEXT	0x00000100
setall	MQZAO_SET_ALL_CONTEXT	0x00000200
altusr	MQZAO_ALTERNATE_USER_AUTHORITY	0x00000400
allmqi	MQZAO_ALL_MQI	0x000007FF
crt	MQZAO_CREATE	0x00010000
dlt	MQZAO_DELETE	0x00020000
dsp	MQZAO_DISPLAY	0x00040000
chg	MQZAO_CHANGE	0x00080000
clr	MQZAO_CLEAR	0x00100000
chgaut	MQZAO_AUTHORIZE	0x00800000
alladm	MQZAO_ALL_ADMIN	0x009E0000
none	MQZAO_NONE	0x00000000
all	MQZAO_ALL	0x009E07FF

These definitions are made in the header file cmqzc.h. In the following example, groupB has been granted authorizations based on the hexadecimal number 0x40007. This corresponds to:

MQZAO_CONNECT	0x00000001
MQZAO_BROWSE	0x00000002
MQZAO_INPUT	0x00000004
MQZAO_DISPLAY	0x00040000

Authority is:	0x00040007

These access rights mean that anyone in groupB can issue the MQI calls:

- MQCONN
- MQGET (with browse)
- MQPUT

They also have DISPLAY authority for the object associated with this authorization file.

Class authorization records

The class authorization records hold authorizations that relate to the entire class. The object name and type fields correspond as follows:

Object Name	Type
@QMGRCLASS	0x80
@PROCESSCLASS	0x80
@QUEUECLASS	0x80

The entry MQZAO_CRT in the authorization field gives authorization to create an object in the class. This is the only class authority.

All-class authorization record

The all-class authorization record holds authorizations that apply to an entire queue manager. The object name and type fields correspond as follows:

Object Name	Type
@ALLCLASSES	0x80

The following authorizations apply to the entire queue manager and are held in the all class authorization file.

Entry...	Gives authorization to...
MQZAO_ALTUSER	Assume the identity of another user when interacting with MQSeries objects.
MQZAO_SET_ALL_CONTEXT	Set the context of a message when issuing MQPUT.
MQZAO_SET_IDENTITY_CONTEXT	Set the identity context of a message when issuing MQPUT.

Authorization files

Chapter 9. MQSeries dead-letter queue handler

MQSeries for Tandem NonStop Kernel, Version 2 Release 2.0.1 provides a dead-letter queue (DLQ), also known as an undelivered-message queue, which is a holding queue for messages that cannot be delivered to their destination queues. Every queue manager in a network should have a DLQ.

Messages are put on the DLQ by queue managers, message channel agents (MCAs), and applications. All messages on the DLQ should be prefixed with the *dead-letter header* structure MQDLH. Messages put on the DLQ by a queue manager or by a message channel agent always have this header structure. Applications putting messages on the DLQ should also supply an MQDLH. The *Reason* field of the MQDLH structure contains a reason code that identifies why the message is on the DLQ.

You should have a routine that runs regularly to process messages on the DLQ. MQSeries supplies a default routine called the *dead-letter queue handler* (the DLQ handler), which you invoke using the **runmqdlq** command.

Instructions for processing messages on the DLQ are supplied to the DLQ handler by means of a user-written *rules table*. That is, the DLQ handler matches messages on the DLQ against entries in the rules table. When a DLQ message matches an entry in the rules table, the DLQ handler performs the action associated with that entry.

This chapter contains the following sections:

- “Invoking the DLQ handler”
- “DLQ handler rules table” on page 158
- “How the rules table is processed” on page 165
- “Example DLQ handler rules table” on page 166

Invoking the DLQ handler

You invoke the DLQ handler using the **runmqdlq** command. You can name the DLQ that you want to process and the queue manager that you want to use as follows:

- From the command prompt using parameters. For example:

```
runmqdlq /IN qrule/ ABC1.DEAD.LETTER.QUEUE ABC1.QUEUE.MANAGER
```

- In the rules table. For example:

```
INPUTQ(ABC1.DEAD.LETTER.QUEUE) INPUTQM(ABC1.QUEUE.MANAGER)
```

The above examples apply to the DLQ called ABC1.DEAD.LETTER.QUEUE, owned by the queue manager ABC1.QUEUE.MANAGER.

Rules table

If you do not specify the DLQ or the queue manager as shown above, the default queue manager for the installation is used along with the DLQ belonging to that queue manager.

The **runmqdlq** command reads input from the rules table, supplied to the standard IN file. You associate the rules table with **runmqdlq** by redirecting IN to the rules file.

To run the DLQ handler, you must be authorized to access both the DLQ itself and any message queues to which messages on the DLQ are forwarded. Furthermore, if the DLQ handler is to be able to put messages on queues with the authority of the user ID in the message context, you must be authorized to assume the identity of other users.

For more information about the **runmqdlq** command, see “runmqdlq (Run dead-letter queue handler)” on page 259

DLQ handler rules table

The DLQ handler rules table defines how the DLQ handler is to process messages that arrive on the DLQ. There are two types of entry in a rules table:

- The first entry in the table, which is optional, contains *control data*.
- All other entries in the table are *rules* for the DLQ handler to follow. Each rule consists of a *pattern* (a set of message characteristics) that a message is matched against, and an *action* to be taken when a message on the DLQ matches the specified pattern. There must be at least one rule in a rules table.

Each entry in the rules table comprises one or more keywords.

For a description of the syntax rules applicable to the rules tables, see “Rules table conventions” on page 163.

Control data

This section explains the keywords that you can include in a control-data entry in a DLQ handler rules table. Please note the following:

- The default value for a keyword, if any, is underlined.
- The vertical line (|) separates alternatives, only one of which can be specified.
- All keywords are optional.

INPUTQ (*QueueName*|' _')

This keyword is the name of the DLQ to which the rules table applies. It lets you name the DLQ you want to process:

1. If you specify a *QName* parameter on the **runmqdlq** command, it overrides any INPUTQ value in the rules table.
2. If you do not specify a *QName* parameter on the **runmqdlq** command, but you specify a value in the rules table, the INPUTQ value in the rules table is used.
3. If you do not specify a DLQ or you specify INPUTQ(' ') in the rules table, the DLQ belonging to the queue manager whose name is supplied on the *QMgrName* parameter on the **runmqdlq** command or on the INPUTQM keyword in the rules table is processed.

INPUTQM (*QueueManagerName*'_'')

This keyword is the name of the queue manager that owns the DLQ. It lets you name the queue manager that owns the DLQ named on the INPUTQ keyword:

1. If you specify a *QMgrName* parameter on the **runmqdlq** command, it overrides any INPUTQM value in the rules table.
2. If you do not specify a *QMgrName* parameter on the **runmqdlq** command, the INPUTQM value in the rules table is used.
3. If no queue manager is specified or you specify INPUTQM(' ') in the rules table, the default queue manager for the installation is used.

RETRYINT (*Interval***[60]**)

This keyword is the interval (in seconds) at which the DLQ handler should attempt to reprocess messages on the DLQ that could not be processed at the first attempt, and for which repeated attempts are requested. By default, the retry interval is 60 seconds.

WAIT (**YES|NO**|*nnn*)

This keyword indicates whether the DLQ handler should wait for further messages to arrive on the DLQ when it detects that there are no further messages that it can process.

YES This keyword causes the DLQ handler to wait indefinitely.

NO This keyword causes the DLQ handler to terminate when it detects that the DLQ is either empty or contains no messages that it can process.

nnn This keyword causes the DLQ handler to wait for *nnn* seconds for new work to arrive before terminating, after it detects that the queue is either empty or contains no messages that it can process.

You should specify WAIT (YES) for busy DLQs, and WAIT (NO) or WAIT (*nnn*) for DLQs that have a low level of activity. If the DLQ handler is allowed to terminate, you should reinvoke it by using triggering.

The control data shown in Figure 33 shows that the rules table applies to the DLQ belonging to queue manager QM1. The plus sign (+) at the end of line 1 indicates that the control data continues from the first nonblank character on line 2.

```
INPUTQ' ' +
INPUTQM'QM1'
```

Figure 33. Example control data

As an alternative to including control data in the rules table, you can supply the names of the DLQ and its queue manager as input parameters of the **runmqdlq** command. If any value is specified both in the rules table and on input to the **runmqdlq** command, the value specified on the **runmqdlq** command takes precedence.

Note: If a control-data entry is included in the rules table, it must be the first entry in the table.

Rules (patterns and actions)

Figure 34 shows an example rule from a DLQ handler rules table. This rule instructs the DLQ handler to make three attempts to deliver to its destination queue any persistent message that was put on the DLQ because MQPUT and MQPUT1 were inhibited.

```
PERSIST(MQPER_PERSISTENT) REASON (MQRC_PUT_INHIBITED) +
ACTION (RETRY) RETRY (3)
```

Figure 34. Example rule. The plus sign (+) at the end of line 1 indicates that the rule continues from the first nonblank character on line 2.

All keywords that you can use on a rule are explained in the remainder of this section. Please note the following:

- The default value for a keyword, if any, is underlined. For most keywords, the default value is * (asterisk), which matches any value.
- The vertical line (|) separates alternatives, only one of which can be specified.
- All keywords except ACTION are optional.

This section begins with a description of the pattern-matching keywords (those against which messages on the DLQ are matched), and then explains the action keywords (those that determine how the DLQ handler is to process a matching message).

Pattern-matching keywords

The pattern-matching keywords that you use to specify values against matched messages on the DLQ are explained below. All pattern-matching keywords are optional.

APPLIDAT (*ApplIdentityData*|*)

This keyword is the *ApplIdentityData* value specified in the message descriptor (MQMD) of the message on the DLQ.

APPLNAME (*PutAppName*|*)

This keyword is the name of the application that issued the MQPUT or MQPUT1 call, as specified in the *PutAppName* field of the message descriptor (MQMD) of the message on the DLQ.

APPLTYPE (*PutAppType*|*)

This keyword is the *PutAppType* value specified in the message descriptor (MQMD) of the message on the DLQ.

DESTQ (*QueueName*|*)

This keyword is the name of the message queue for which the message is destined.

DESTQM (*QueueManagerName*|*)

This keyword is the name of the queue manager of the message queue for which the message is destined.

FEEDBACK (*Feedback*|*)

If the *MsgType* value is MQFB_REPORT, the keyword *Feedback* describes the nature of the report.

Symbolic names can be used. For example, you can use the symbolic name MQFB_COA to identify those messages on the DLQ that require confirmation of their arrival on their destination queues.

FORMAT (*Format**)

This keyword is the name that the sender of the message uses to describe the format of the message data.

MSGTYPE (*MsgType**)

This keyword is the message type of the message on the DLQ.

Symbolic names can be used. For example, you can use the symbolic name MQMT_REQUEST to identify those messages on the DLQ that require replies.

PERSIST (*Persistence**)

This keyword is the persistence value of the message. (The persistence of a message determines whether it survives restarts of the queue manager.)

Symbolic names can be used. For example, you can use the symbolic name MQPER_PERSISTENT to identify those messages on the DLQ that are persistent.

REASON (*ReasonCode**)

This keyword is the reason code that describes why the message was put to the DLQ.

Symbolic names can be used. For example, you can use the symbolic name MQRC_Q_FULL to identify those messages placed on the DLQ because their destination queues were full.

REPLYQ (*QueueName**)

This keyword is the name of the reply-to queue specified in the message descriptor (MQMD) of the message on the DLQ.

REPLYQM (*QueueManagerName**)

This keyword is the name of the queue manager of the reply-to queue, as specified in the message descriptor (MQMD) of the message on the DLQ.

USERID (*UserIdentifier**)

This keyword is the user ID of the user who originated the message on the DLQ, as specified in the message descriptor (MQMD).

Action keywords

The action keywords that you use to describe how a matching message is to be processed are detailed as follows:

ACTION (**DISCARD**|**IGNORE**|**RETRY**|**FWD**)

This keyword is the action to be taken for any message on the DLQ that matches the pattern defined in this rule.

DISCARD This keyword causes the message to be deleted from the DLQ.

IGNORE This keyword causes the message to be left on the DLQ.

RETRY This keyword causes the DLQ handler to try again to put the message on its destination queue.

FWD This keyword causes the DLQ handler to forward the message to the queue named on the FWDQ keyword.

You must specify the ACTION keyword. The number of attempts made to implement an action is governed by the RETRY keyword. The interval between attempts is controlled by the RETRYINT keyword of the control data.

FWDQ (QueueName|&DESTQ|&REPLYQ)

This keyword is the name of the message queue to which the message should be forwarded when ACTION (FWD) is requested.

QueueName This keyword is the name of a message queue. FWDQ(' ') is not valid.

&DESTQ This keyword causes the queue name to be taken from the *DestQName* field in the MQDLH structure.

&REPLYQ This keyword causes the name to be taken from the *ReplyToQ* field in the message descriptor, MQMD.

To avoid error messages when a rule specifying FWDQ (&REPLYQ) matches a message with a blank *ReplyToQ* field, you can specify REPLYQ (?*) in the message pattern.

FWDQM (QueueManagerName|&DESTQM|&REPLYQM|' _')

This keyword identifies the queue manager of the queue to which a message is to be forwarded.

QueueManagerName

This keyword is the name of the queue manager of the queue to which a message is to be forwarded when ACTION (FWD) is requested.

&DESTQM

This keyword causes the queue manager name to be taken from the *DestQMGrName* field in the MQDLH structure.

&REPLYQM

This keyword causes the name to be taken from the *ReplyToQMGr* field in the message descriptor (MQMD).

' '

FWDQM(' ') is the default value and identifies the local queue manager.

HEADER (YES|NO)

This keyword specifies whether the MQDLH should remain on a message for which ACTION (FWD) is requested. By default, the MQDLH remains on the message. The HEADER keyword is not valid for actions other than FWD.

PUTAUT (DEF|CTX)

This keyword defines the authority with which messages should be put by the DLQ handler:

DEF This keyword causes messages to be put with the authority of the DLQ handler itself.

CTX This keyword causes the messages to be put with the authority of the user ID in the message context. If you specify PUTAUT (CTX), you must be authorized to assume the identity of other users.

RETRY (*RetryCount*1)

RETRY is the number of times, in the range 1–999, that an action should be attempted (at the interval specified on the RETRYINT keyword of the control data).

Note: The count of attempts made by the DLQ handler to implement any particular rule is specific to the current instance of the DLQ handler; the count does not persist across restarts. If the DLQ handler is restarted, the count of attempts made to apply a rule is reset to zero.

Rules table conventions

The rules table must adhere to the following conventions regarding its syntax, structure, and contents:

- A rules table must contain at least one rule.
- Keywords can occur in any order.
- A keyword can be included once only in any rule.
- Keywords are not case-sensitive.
- A keyword and its parameter value must be separated from other keywords by at least one blank or comma.
- Any number of blanks can occur at the beginning or end of a rule, and between keywords, punctuation, and values.
- Each rule must begin on a new line.
- For reasons of portability, the significant length of a line should not be greater than 72 characters.
- Use the plus sign (+) as the last nonblank character on a line to indicate that the rule continues from the first nonblank character in the next line. Use the minus sign (–) as the last nonblank character on a line to indicate that the rule continues from the start of the next line. Continuation characters can occur within keywords and parameters.
- Comment lines, which begin with an asterisk (*), can occur anywhere in the rules table.
- Blank lines are ignored.
- Each entry in the DLQ handler rules table comprises one or more keywords and their associated parameters. The parameters must follow these syntax rules:

- Each parameter value must include at least one significant character. The delimiting quotation marks in quoted values are not considered significant. For example, these parameters are valid:

```

FORMAT('ABC') 3 significant characters
FORMAT(ABC)   3 significant characters
FORMAT('A')   1 significant character
FORMAT(A)     1 significant character

```

Rules table conventions

FORMAT(' ') 1 significant character

These parameters are invalid because they contain no significant characters:

FORMAT('')

FORMAT()

FORMAT()

FORMAT

- Wildcard characters are supported: you can use the question mark (?) in place of any single character, except a trailing blank; you can use the asterisk (*) in place of zero or more adjacent characters. The asterisk (*) and the question mark (?) are *always* interpreted as wildcard characters in parameter values.
- Wildcard characters cannot be included in the parameters of these keywords: ACTION, HEADER, RETRY, FWDQ, FWDQM, and PUTAUT.
- Trailing blanks in parameter values, and in the corresponding fields in the message on the DLQ, are not significant when performing wildcard matches. However, leading and embedded blanks within strings in quotation marks are significant to wildcard matches.
- Numeric parameters cannot include the question mark (?) wildcard character. The asterisk (*) can be used in place of an entire numeric parameter, but cannot be included as part of a numeric parameter. For example, these are valid numeric parameters:

MSGTYPE(2) Only reply messages are eligible

MSGTYPE(*) Any message type is eligible

MSGTYPE('*') Any message type is eligible

However, MSGTYPE('2*') is not valid, because it includes an asterisk (*) as part of a numeric parameter.

- Numeric parameters must be in the range 0–999. If the parameter value is in this range, it is accepted, even if it is not currently valid in the field to which the keyword relates. Symbolic names can be used for numeric parameters.
- If a string value is shorter than the field in the MQDLH or MQMD to which the keyword relates, the value is padded with blanks to the length of the field. If the value, excluding asterisks, is longer than the field, an error is diagnosed. For example, these are all valid string values for an 8-character field:

'ABCDEFGH' 8 characters

'A*C*E*G*I' 5 characters excluding asterisks

'*A*C*E*G*I*K*M*O*' 8 characters excluding asterisks

- Strings that contain blanks, lowercase characters, or special characters other than period (.), forward slash (/), underscore (_), and percent sign (%) must be enclosed in single quotation marks. Lowercase characters not enclosed in quotation marks are folded to uppercase. If the string includes a quotation, two single quotation marks must be used to denote both the beginning and the end of the quotation. When the length of the string is calculated, each occurrence of double quotation marks is counted as a single character.

How the rules table is processed

The DLQ handler searches the rules table for a rule whose pattern matches a message on the DLQ. The search begins with the first rule in the table and continues sequentially through the table. When a rule with a matching pattern is found, the action from that rule is attempted. The DLQ handler increments the retry count for a rule by one whenever it attempts to apply that rule. If the first attempt fails, the attempt is repeated until the count of attempts made matches the number specified on the RETRY keyword. If all attempts fail, the DLQ handler searches for the next matching rule in the table.

This process is repeated for subsequent matching rules until an action is successful. When each matching rule has been attempted the number of times specified on its RETRY keyword, and all attempts have failed, ACTION (IGNORE) is assumed. ACTION (IGNORE) is also assumed if no matching rule is found.

Notes:

1. Matching rule patterns are sought only for messages on the DLQ that begin with an MQDLH. Messages that do not begin with an MQDLH are reported periodically as being in error, and remain on the DLQ indefinitely.
2. All pattern keywords can be allowed to default, such that a rule can consist of an action only. However, action-only rules are applied to all messages on the queue that have MQDLHs and that have not already been processed in accordance with other rules in the table.
3. The rules table is validated when the DLQ handler is started, and errors are flagged at that time. (Error messages issued by the DLQ handler are described in Appendix N, “Messages” on page 363.) You can make changes to the rules table at any time, but those changes do not take effect until the DLQ handler is restarted.
4. The DLQ handler does not alter the content of messages, of the MQDLH, or of the message descriptor. The DLQ handler always puts messages to other queues with the message option MQPMO_PASS_ALL_CONTEXT.
5. The DLQ handler opens the DLQ with the MQOO_INPUT_AS_Q_DEF option.
6. Multiple instances of the DLQ handler could run concurrently against the same queue, using the same rules table. However, it is more usual for there to be a one-to-one relationship between a DLQ and a DLQ handler.

Ensuring that all DLQ messages are processed

The DLQ handler keeps a record of all messages on the DLQ that have been viewed but not removed. If you use the DLQ handler as a filter to extract a small subset of the messages from the DLQ, the DLQ handler still has to keep a record of those messages on the DLQ that it did not process. Also, the DLQ handler cannot guarantee that new messages arriving on the DLQ are viewed, even if the DLQ is defined as first-in-first-out (FIFO). Therefore, if the queue is not empty, a periodic rescan of the DLQ is performed to check all messages. For these reasons, you should ensure that the DLQ contains as few messages as possible. If messages that cannot be discarded or forwarded to other queues (for whatever reason) are allowed to accumulate on the queue, the workload of the DLQ handler increases and the DLQ itself is in danger of filling up.

Example rules table

You can take specific measures to enable the DLQ handler to empty the DLQ. For example, do not use ACTION (IGNORE), which leaves messages on the DLQ. ACTION (IGNORE) is assumed for messages that are not explicitly addressed by other rules in the table. Instead, for those messages that you would otherwise ignore, use an action that moves the messages to another queue. For example:

```
ACTION (FWD) FWDQ (IGNORED.DEAD.QUEUE) HEADER (YES)
```

Similarly, the final rule in the table should process messages that have not been addressed by earlier rules in the table. For example, the final rule in the table could be:

```
ACTION (FWD) FWDQ (REALLY.DEAD.QUEUE) HEADER (YES)
```

This action causes messages that fall through to the final rule in the table to be forwarded to the queue REALLY.DEAD.QUEUE, where they can be processed manually. If you do not have such a rule, messages are likely to remain on the DLQ indefinitely.

Example DLQ handler rules table

The following is an example rules table that contains a single control-data entry and several rules:

```
*****
*           An example rules table for the runmqdlq command           *
*****
* Control data entry
* -----
* If no queue manager name is supplied as an explicit parameter to
* runmqdlq, use the default queue manager for the machine.
* If no queue name is supplied as an explicit parameter to runmqdlq,
* use the DLQ defined for the local queue manager.
*
inputqm(' ') inputq(' ')

* Rules
* ----
* We include rules with ACTION (RETRY) first to try to
* deliver the message to the intended destination.

* If a message is placed on the DLQ because its destination
* queue is full, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_Q_FULL) ACTION(RETRY) RETRY(5)

* If a message is placed on the DLQ because of a put inhibited
* condition, attempt to forward the message to its
```

* destination queue. Make 5 attempts at approximately
 * 60-second intervals (the default value for RETRYINT).

REASON(MQRC_PUT_INHIBITED) ACTION(RETRY) RETRY(5)

* The AAAA corporation are always sending messages with incorrect
 * addresses. When we find a request from the AAAA corporation,
 * we return it to the DLQ (DEADQ) of the reply-to queue manager
 * (&REPLYQM).
 * The AAAA DLQ handler attempts to redirect the message.

MSGTYPE(MQMT_REQUEST) REPLYQM(AAAA.*) +
 ACTION(FWD) FWDQ(DEADQ) FWDQM(&REPLYQM)

* The BBBB corporation never do things by half measures. If
 * the queue manager BBBB.1 is unavailable, try to
 * send the message to BBBB.2

DESTQM(bbbb.1) +
 action(fwd) fwdq(&DESTQ) fwdqm(bbbb.2) header(no)

* The CCCC corporation considers itself very security
 * conscious, and believes that none of its messages
 * will ever end up on one of our DLQs.
 * Whenever we see a message from a CCCC queue manager on our
 * DLQ, we send it to a special destination in the CCCC organization
 * where the problem is investigated.

REPLYQM(CCCC.*) +
 ACTION(FWD) FWDQ(ALARM) FWDQM(CCCC.SYSTEM)

* Messages that are not persistent run the risk of being
 * lost when a queue manager terminates. If an application
 * is sending nonpersistent messages, it should be able
 * to cope with the message being lost, so we can afford to
 * discard the message.

PERSIST(MQPER_NOT_PERSISTENT) ACTION(DISCARD)

* For performance and efficiency reasons, we like to keep
 * the number of messages on the DLQ small.
 * If we receive a message that has not been processed by
 * an earlier rule in the table, we assume that it
 * requires manual intervention to resolve the problem.
 * Some problems are best solved at the node where the
 * problem was detected, and others are best solved where
 * the message originated. We don't have the message origin,
 * but we can use the REPLYQM to identify a node that has
 * some interest in this message.
 * Attempt to put the message onto a manual intervention
 * queue at the appropriate node. If this fails,
 * put the message on the manual intervention queue at
 * this node.

REPLYQM('?*') +
 ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION) FWDQM(&REPLYQM)

ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION)

Example rules table

Chapter 10. Instrumentation and EMS events

This chapter:

- Provides a brief introduction to MQSeries instrumentation events, which you can use to monitor the operation of queue managers. For detailed information about instrumentation events, see the *MQSeries Programmable System Management* manual.
- Describes the use of Event Management Service (EMS) events by MQSeries for Tandem NSK.

MQSeries instrumentation events

Instrumentation events cause *event messages* to be generated when a queue manager detects a predefined set of conditions. For example, a *Queue Full* event results from the following conditions:

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

Other conditions that can cause instrumentation events include:

- A limit on the number of messages on a queue being reached
- A queue not being serviced within a specified time
- A channel instance being started or stopped
- An application attempting to open a queue specifying a user ID that is not authorized

With the exception of channel events, all instrumentation events must be enabled before they can be generated.

The event message contains information about the conditions resulting in the event. It is put onto an *event queue*. An application can retrieve the event message from this queue for analysis.

If you define event queues as remote queues, you can put all the event queues on a single queue manager (for those nodes that support instrumentation events). You can then use the events generated to monitor a network of queue managers from a single node.

Types of event

There are four types of instrumentation event:

Queue manager events

Queue manager events are related to the definitions of resources within queue managers. For example, a queue manager event could be generated when an application attempts to put a message to a queue that does not exist.

Instrumentation events

Performance events

Performance events are notifications that a threshold has been reached by a resource. For example, a performance event could be generated when a queue-depth limit has been reached or, following an MQGET call, if a queue has not been serviced within a predefined time.

Channel events

Channel events are reported by channels as a result of conditions detected during their operation. For example, a channel event could be generated when a channel instance is stopped.

Trigger events

A trigger event can occur when a queue manager detects that the conditions for the trigger event have been met. For example, a queue can be configured to generate a trigger event each time a message arrives. (The conditions for trigger events and instrumentation events are quite different.)

A trigger event causes a trigger message to be put on an initiation queue and, optionally, an application program is started.

Event notification through event queues

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

- Gets the message from the queue.
- Processes the message to extract the event data. For a description of event message formats, see the *MQSeries Programmable System Management* manual.

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

This event queue...	Contains messages from...
SYSTEM.ADMIN.QMGR.EVENT	Queue manager events
SYSTEM.ADMIN.PERFM.EVENT	Performance events
SYSTEM.ADMIN.CHANNEL.EVENT	Channel events

Using triggered event queues

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

Enabling instrumentation events

How you enable an instrumentation event depends on the event type:

- Queue manager events are enabled by setting attributes on the queue manager.
- Performance events as a whole must be enabled on the queue manager. You must also enable specific performance events by setting the appropriate queue

attribute, and identify the conditions, such as a queue-depth-high limit, that will result in the event.

- Channel events occur automatically; they do not need to be enabled. If you do not want to monitor channel events, you can put-inhibit the channel event queue.

You enable and disable the generation of instrumentation events using either of the following:

- MQSC commands. For more information, see the *MQSeries Command Reference* manual.
- PCF commands for queue managers. For more information, see the *MQSeries Programmable System Management* manual.

Event messages

Event messages contain information relating to the origin of an event, including the type of event, the name of the application that caused the event, and, for performance events, a short statistics summary for the queue.

The format of event messages is similar to that of PCF response messages. The message data can be retrieved from event messages by user-written administration programs using the data structures described in the *MQSeries Programmable System Management* manual.

Event Management Service (EMS) events

MQSeries for Tandem NSK generates Event Management Service (EMS) event messages that correspond to the MQSeries queue-manager events, channel events, and performance events. EMS messages may also be generated that correspond to the message entries in the MQSeries logs and to FFSTs. These event messages can alert system operators and administrators to software conditions that could have an adverse effect on the MQSeries operating environment.

EMS template files supplied with MQSeries for Tandem NSK

The following files are supplied in subvolume ZMQSSYS:

ZMQSTMPL (file code 839)

An EMS template object file containing the formatting templates for the EMS events generated by MQSeries.

ZMQSDDL (file code 101)

The Data Definition Language schema for the EMS events generated by MQSeries.

ZMQSC (file code 101)

Compiled output (C) from the DDL compiler of definitions of the EMS events generated by the product.

ZMQSCOB (file code 101)

Compiled output (COBOL) from the DDL compiler of definitions of the EMS events generated by the product.

ZMQSPAS (file code 101)

Compiled output (PASCAL) from the DDL compiler of definitions of the EMS events generated by the product.

ZMQSTACL (file code 101)

Compiled output (TACL) from the DDL compiler of definitions of the EMS events generated by the product.

ZMQSTAL (file code 101)

Compiled output (TAL) from the DDL compiler of definitions of the EMS events generated by the product.

The subvolume ZMQSSYS contains the EMS template file SMQSTMPL, from which the template file ZMQSTMPL is generated. The file ZMQSTMPL is ready for integration with your system's event templates using COUP and SYSGEN. The source of the event templates is supplied, so that you can modify the formatting of the events when they are used in your environment.

For example, you might not be interested in displaying all of the information that is contained in an event, or you might want to add or change text that is displayed along with the information in the event. See the Tandem documentation for a description of the EMS event template source language, and for the procedures used to compile the definitions to produce an alternative ZMQSTMPL file.

Integrating the MQSeries EMS event templates

The template object file must be integrated into your system's resident and nonresident EMS template files, so that programs such as VIEWPOINT and EMSDIST can format and display MQSeries EMS events.

A procedure for integrating the MQSeries EMS templates into the system templates is described in the remainder of this section. Note that different procedures might be preferred in your installation.

1. Determine the names of the current system templates using the COUP command INFO ALLPROCESSORS: note the values displayed for the EMS^TEMPLATES parameter. For example:

```
$DEV2 ZMQSSYS 425> coup
CONFIGURATION UTILITY PROGRAM - T9023D30 - (26MAY95)      SYSTEM
\RAPTOR
COPYRIGHT TANDEM COMPUTERS INCORPORATED 1987-1994
CONFIG $SYSTEM.SYS06.OSCONFIG
1) info allprocessors
    EMS^TEMPLATES ( RESIDENT $SYSTEM.EMS.NEWRES,
                   NONRESIDENT $SYSTEM.EMS.NEWNRES )
    SYSTEM^ID     ( NAME \RAPTOR, NUMBER 001 )
    SYSTEM^TIME   ( GMT^OFFSET -05:00, DST USA66 )
    DP2_UPSOPTION ( OFF )
2) exit
```

2. Run the TEMPLI compiler to create new system template files combining the current system templates with the new MQSeries templates. This is a two-step process:

- a. Create a text file containing the following commands:

```
FILE <current NONRESIDENT system template file>
FILE <MQSeries install volume>.ZMQSSYS.ZMQSTMPL
EXIT
```

For example:

```
FILE $SYSTEM.SYS06.TEMPLATE
FILE $DEV2.ZMQSSYS.ZMQSTMPL
EXIT
```

- b. Run the TEMPLI compiler, specifying the new text file as input:

```
TEMPLI /IN <command file>/<new resident template file>, <new
nonresident template file>
```

For example, if the command file you created is called TEMGUIDE and you are creating new template files in \$SYSTEM.EMS:

```
TEMPLI /IN TEMGUIDE/$SYSTEM.EMS.NEWRES, $SYSTEM.EMS.NEWRES
```

The compilation of the new template files can take several minutes, as all the EMS event templates required on your system are processed.

3. Using the COUP command, configure your system to use the new EMS event templates in place of the current templates:

```
ASSUME ALLPROCESSORS
ALTER EMS^TEMPLATES(RESIDENT <new resident template file>,
NONRESIDENT <new nonresident template file>)
EXIT
```

Note: In order to make this change permanent, you must update the system using SYSGEN.

For further information about EMS templates, see the Tandem *DSM Template Services Manual*. This manual also describes how to use SYSGEN to perform this task.

Defining the PARAM MQEMSEVENTS

To complete the enablement of MQSeries EMS events, you must ensure that the PARAM MQEMSEVENTS is correctly defined. The value is a four-character string interpreted as a bit map, as follows:

EMS message	Bit-map entry	MQEMSEVENT value
FFST	0x00000001	1
START / STOP	0x00000002	2
PERFORMANCE	0x00000004	4
CHANNEL	0x00000008	8
QUEUE MANAGER	0x00000010	16
MESSAGE	0x00000020	32
ERROR	0x00000040	64
ALL	0x0000007F	127

Thus, to switch on all EMS events for MQSeries, you must define the following PARAM in the TACL environment from which any administration commands are issued:

```
PARAM MQEMSEVENTS 127
```

This definition is also required in server class definitions of all server classes for MQSeries. Each server class may be configured with different options. See “Changing the parameters of Pathway server classes” on page 57 for more information.

By default, no EMS events are generated (that is, the PARAMs are not defined).

Using an alternative collector

On a Tandem system, the default EMS event collector is called \$0, and is always present. All EMS events generated by an MQSeries queue manager are sent to the default collector. If you want a different collector to collect EMS events for a queue manager, modify the EMSCollector entry in the Configuration stanza in the QMINI file, and restart the queue manager. You may specify a different EMS event collector for each queue manager.

Writing programs to process MQSeries EMS events

You can write an application to monitor an MQSeries queue manager by processing EMS event messages. Such an application could also affect the operation of the queue manager by issuing PCF commands in response to the EMS event messages generated.

The files ZMQSC, ZMQSTAL, ZMQSCOB, ZMQSPAS, and ZMQSTACL supplied with MQSeries for Tandem NSK in the ZMQSSYS subvolume define the tokens contained in the MQSeries EMS event messages in C, TAL, COBOL, PASCAL, and TACL. These definitions could be used by an administration program to understand the format of the messages.

For further information about the EMS events generated by MQSeries, see Appendix M, “EMS event template used by MQSeries for Tandem NSK” on page 359.

Chapter 11. Understanding transactional support and messaging

Applications that use the Message Queue Interface (MQI) let you execute put and get operations under syncpoint control. In MQSeries for Tandem NSK, there are two transactional operations as follows:

- **Commit** — the act of completing a transaction so that changes to the database are recorded and stable. Protected resources are released after the transaction is committed.
- **Back out**— 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.

Commit and back out are provided as part of the TM/MP (or TMF) Transaction environment on Tandem NSK. On MQSeries for Tandem NSK, MQPUT, MQGET, and MQPUT1 are syncpointed operations by default. That is, unless no syncpointing is requested explicitly by an application, a TMF transaction *must* be in progress or the MQI call fails.

Using the NonStop TM/MP (Transaction Manager)

MQSeries for Tandem NSK V2.2.0.1 relies on the transaction management facilities of Tandem's NonStop TM/MP to maintain transaction integrity.

The NonStop TM/MP transaction system provides transactional protection and concurrency, and object-catalog and message integrity.

MQSeries handles TM/MP transactions transparently. If you have a TM/MP transaction in progress when an MQI function is called, any put and get operations with the syncpoint option become part of the same transaction. That is, the updates to the queues occur when the transaction is committed. In the event of volume recovery, all committed transactions are applied to the database files, and uncommitted transactions are backed out. A transaction backout reapplies before-images to database records to undo the effects of a cancelled transaction. Changes do not occur until a commit operation is complete.

If the user application has a transaction in progress and attempts an MQI call out of syncpoint, MQSeries suspends the current, inherited transaction, starts one of its own, commits that transaction, and resumes the original transaction prior to returning control to the user application. Updates to queues resulting from put and get operations occur immediately.

Syncpointing limits

A TMF transaction can have a maximum of 5000 locks. This limitation restricts the number of messages that can be MQPUT or MQGET in a single TMF transaction. To calculate the maximum number of messages (N) of length (LEN) within a single transaction, use the following formula:

$$N = \text{INT}(5000 / (1 + K + \text{INT}(\text{LEN} / 4 \text{ KB})))$$

where K is the number of alternate key files for the queue that is involved in the messaging. This can be changed using the **altmqfls** command.

That is, for:

Message length	K=3	K=2	K=1	K=0
1 KB	N=1250	N=1667	N=2500	N=5000
10 KB	N=833	N=1000	N=1250	N=1667
100 KB	N=172	N=178	N=185	N=192
100 KB	N=172	N=178	N=185	N=192
1 MB	N=19	N=19	N=19	N=19
4 MB	N=4	N=4	N=4	N=4

If N is exceeded, error 2024 (MQRC_SYNCPOINT_LIMIT_REACHED) is returned and the transaction is backed out: no messages are put or got. If an application started the transaction, it must cancel the transaction.

For complex operations, the effective value of N will be smaller. For example, when 4 MB messages are being sent, the batch size for the channel should be no larger than 2 MB to allow for messages to be dequeued and possibly enqueued to the dead letter queue in the event of failure.

Configuration requirements for TM/MP and MQSeries for Tandem NSK

Your Tandem system needs to be configured with TMF (TM/MP) auditing enabled for all volumes that are to contain queue managers or queues. Use the TMFCOM command `status datavols` to determine the status of auditing on any volume on your system. (Note that you have to be SUPER.SUPER to use TMFCOM.) In addition, the TMF audit trails configured for the data volumes that support queue managers must be large enough to allow for the peak rate and size of message traffic expected on all queue managers that use these volumes.

Since misbehaved applications can cause long-running transactions, the TMF system should be configured automatically to cancel long-running transactions. The size of the audit trail, and the time limit on long-running transactions, are application-dependent tuning parameters. The audit trail configured for MQSeries does not need to be configured for dumping to tape.

Monitoring

Use the TMFCOM interface to monitor the status of TMF, with MQSeries running. Use the status tmf and status datavols commands to investigate the general status of TMF, and the status of individual data volumes.

Audit-trail size

Approximate TM/MP audit-trail sizings can be calculated using the following formula. This provides TM/MP audit-trail usage in bytes for a message of a given length:

$$(message\text{-}data\ length + 1\ KB + [1\ KB] + (message\text{-}data\ length - 2\ KB) / 4\ KB \times 4) \times 2$$

where:

- *message-data length* is the length of the MQSeries message.
- 1 KB represents the cumulative size of MQSeries headers, file structures, and key lengths.
- [1 KB] is optional and represents the transmission header (required only if messages are to be sent across a channel).
- $(message\ data\ length - 2KB) / 4KB \times 4$ represents the overhead incurred if the overflow file is used. For this to occur, the message length must be greater than 2 KB. (If this value is negative, ignore it; do not subtract it.)
- $\times 2$ represents the TMF pre-image and post-image audit-trail-space usage.

Troubleshooting

EMS events or FFST reports indicating that BEGINTRANSACTION commands have been disabled by TMF usually mean that the audit trail is filled. This can occur because the audit trail is too small, or because a badly behaved application has held a long-running transaction and TMF has not terminated it in time.

In this instance:

- Increase the size of the audit trail, or
- Identify the cause of the long-running transaction and correct it, or
- Reconfigure TMF to terminate long-running transactions after a shorter period.

EMS events and FFST reports indicating that TMF is not running indicate a configuration problem with TMF that must be corrected before running the queue manager again. In general, the MQSeries queue manager requires TMF to be running correctly to operate in any capacity. Although messages are not lost or corrupted, the queue manager is not able to operate without TMF.

TM/MP configuration requirements

Chapter 12. Recovery and restart

A messaging system ensures that messages entered into the system are delivered to their destination. A messaging system must also provide a method of tracking the messages in the system, and of recovering messages if the system fails for any reason.

MQSeries for Tandem NSK ensures that messages are not lost by using the Tandem NonStop Transaction Manager (TM/MP). TM/MP provides transaction protection, queue-file consistency, and queue-file recovery.

The TM/MP subsystem manages the complex operations for current transactions and database consistency, both user operations and MQSeries operations, making these operations transparent to both users and application programs.

A recovery restores the queue manager to the state it was in when the queue manager stopped. Any transactions that are in process are rolled back, removing from the queues any messages that were not committed at the time the queue manager stopped. Recovery restores all persistent messages; nonpersistent messages are lost during the process.

The remainder of this chapter introduces the concepts of recovery and restart in more detail and explains how to recover if you experience any problems. It covers the following topics:

- “Fault tolerance and recovery”
- “Backing up and restoring” on page 180

Fault tolerance and recovery

If you properly configure the MQSeries V2.2.0.1 product and the NSK system software and hardware (for example, all components are configured as redundant or mirrored devices or process pairs as prescribed by Tandem), the failure of any single hardware or software component does not result in loss, duplication or corruption of data or the permanent loss (that is, requiring outside intervention to restore) of any function of the system. MQSeries for Tandem NSK V2.2.0.1 can recover from a single point of failure while maintaining data integrity as specified above.

Repeated consecutive failure (for example, fail-recovery looping) of the same software component is trapped once a configured maximum number of failures is exceeded. In such instances, or in the case of multiple-point failure, the MQSeries product cannot preserve queue integrity.

Backing up and restoring

Periodically, you might want to make a backup of your queue manager data to provide protection against possible corruption due to hardware failures.

Backing up MQSeries

To back up a queue manager's data, you must:

1. Ensure that the queue manager is not running.

If your queue manager is running, stop it with the **endmqm** command.

Note: If you try to make a backup of a running queue manager, the backup might not be consistent due to updates in progress when the files were copied.

2. Locate the volumes and subvolumes under which the queue manager stores its data

You can use the information in the configuration files to determine these directories. For more information, see Chapter 13, "Configuration files" on page 183.

Note: If you have difficulty understanding the names that appear in the directory it is because the names are transformed to ensure that they are compatible with the platform on which you are using MQSeries. For more information about name transformations, see "Volume structure" on page 70.

3. Make copies of all the queue manager's data and log file subvolumes.

Ensure that you do not overlook any of the files.

Restoring MQSeries

To restore a backup of a queue manager's data, you must:

1. Ensure that the queue manager is not running.
2. Locate the subvolumes under which the queue manager stores its data. This information is located in the configuration file.
3. Empty the subvolumes into which you are going to place the backed up data.
4. Copy the backed up queue manager data into the correct places.

Check the resulting directory structure to ensure that you have all of the required directories.

Check that the MQSeries and queue manager configuration files are consistent so that MQSeries can look in the correct places for the restored data.

If the data was backed up and restored correctly, the queue manager starts.

Recovery and restart of MQSS Servers

The MQSS Server processes are Tandem NSK process pairs. This means that they are designed to continue to provide their services in the event of a failure of a single CPU, or of the process itself. In the case of a single failure (for example, the CPU that contains the primary MQSS Server process fails, or the primary MQSS Server process itself fails) the backup MQSS Server process takes over as the new primary without interruption of queue manager processing.

In normal single-point-of-failure situations, therefore, no recovery actions specific to the MQSS Server are required. A message is logged to the home terminal and the message log file by an MQSS Server whenever the backup has to be restarted by the primary, or the backup takes over as primary.

In the case of a more serious failure (for example, an environmental failure that prevents initialization of the primary or backup MQSS Servers) the TS/MP Pathmon process attempts to restart the MQSS Server up to 10 times.

The MQSS Server accesses only databases that are protected by TM/MP, so that in the event of failures affecting access to the disks, the protection provided by TM/MP and the DP2 disk subsystem can be relied upon.

The MQSS Server can be individually stopped and restarted by use of TS/MP Pathcom commands if necessary, though this is not normally required. The **strmqm** command automatically starts all MQSS Server classes that have names that begin with the character string MQS-STATUS. On queue manager shutdown, all active MQSS Server classes coordinate their shutdown and, when all active queue manager connections are closed, any MQSS Server involved with those connections shuts down.

Disaster recovery using RDF

The following procedures should be used to bring into operation a queue manager on the backup site, if a disaster makes the primary site unusable:

1. Ensure that RDF has completed updating the databases.
2. Use FUP to set the audit flag on for the following files:
 - a. All files in the <qmgr>M subvolume
 - b. All files in the <qmgr>D subvolume, except QMINI, UQMINI and UMQSINI, PATHCTL, TRACEOPT, and SHUTDOWN
3. Set your default volume to the <qmgr>D subvolume. Run up PATHMON manually, run PATHCOM against it, and load the Pathway configuration for the queue manager.
4. Perform an INFO command on all objects.
 - a. Replace all instances of the primary node name with the node name of this system.
 - b. Verify that the CPU numbers assigned to the server classes are still valid for this backup site.
 - c. Verify that the Home Terminal and Out file names are valid for the backup site. If they are not, change them.
 - d. Verify that any alternate TCP/IP process name specified is valid for this system.
 - e. If the backup site is itself not configured for RDF operation, remove any PARAM MQRDF settings from the EC server class definitions.
 - f. After verifying the Pathway configuration, save it back to disk using the shutdown2 command and exit from Pathcom. If the home terminal name has been changed, modify the QMINI file for the queue manager to match the Pathway configuration. If necessary, change the TCP/IP listener ports configuration in the QMINI file.
 - g. Use **strmqm** to start the queue manager.
 - h. Using **runmqsc** verify the channel configuration, and adjust if necessary.

Recovery and restart of MQSS Servers

- i. If you attempt to bring up the same channels as were running before, the channel configuration on the remote queue managers might also have to be changed unless the backup system can be reconfigured to use the same IP address or hostname, for TCP/IP channels, or the same SNAX/APC and ICE resource names (for example, process name, LU names, and so on) for SNA channels.
- j. Be prepared for channel synchronization or sequence errors, particularly if the primary site channels were running at the time of the disaster. RDF does not ensure that the databases on the backup site are up to date (in lockstep with the primary) so data can be lost as a result of a complete disaster. To minimize the chances of this, ensure that your RDF configuration can handle the volume of database updates associated with your message flow.

Chapter 13. Configuration files

MQSeries for Tandem NSK uses *configuration files* to hold basic product configuration information. This chapter describes what configuration files are and how you can use them to change the way that queue managers operate. It contains the following sections:

- “What are configuration files?”
- “MQSeries configuration file (MQSINI)”
- “Queue manager configuration file (QMINI)” on page 185
- “Editing configuration files” on page 193

What are configuration files?

Configuration files define optional values for individual queue managers and for MQSeries on the node as a whole. These files are referred to as *ini* files or *stanza* files. A configuration file contains one or more stanzas, where a stanza is a group of lines in the file that together have a common function or define part of a system. For example, there are stanzas associated with logs, channels, and installable services.

Configuration files can be modified automatically by commands that change the configuration of queue managers on the node and also by editing them manually. In general, however, configuration files should not be modified manually while queue managers are running.

There are two types of configuration file:

- The *MQSeries configuration file*, MQSINI, which specifies values for MQSeries on the node as a whole. There is normally one MQSeries configuration file per node.
- *Queue manager configuration files*, QMINI, which specify values for specific queue managers. There is one queue manager configuration file for each queue manager on the node.

MQSeries configuration file (MQSINI)

The MQSeries configuration file, MQSINI, contains information relevant to all the queue managers on an MQSeries installation node. It is created automatically during installation. In particular, the MQSeries configuration file is used to locate the data associated with each queue manager. The MQSeries configuration file is located in the ZMQSSYS subvolume, by default \$SYSTEM.ZMQSSYS.MQSINI. An environment variable, MQMACHINIFILE, is provided for use on systems where the MQSeries configuration file does not have the default name or location.

What the MQSeries configuration file contains

The MQSINI file contains installation-wide defaults, the names of the queue managers, the name of the default queue manager, and the location of the files associated with each of them. The following stanzas can appear in MQSINI:

AllQueueManagers

Specifies values for installation-wide file locations and volumes.

DefaultQueueManager

Specifies the default queue manager for the installation. This queue manager processes MQSC commands when a queue manager name is not explicitly specified. The stanza is automatically updated if you create a new default queue manager. If you inadvertently create a default queue manager and then want to revert to the original, you must alter this stanza manually.

QueueManager

There is one such stanza for each queue manager. The QueueManager stanza specifies the queue manager name and the location of the files associated with that queue manager. The names of these files are based on the queue manager name but are transformed if the queue manager name is not a valid file name.

Figure 35 shows an example MQSINI file.

```

*****#
** Module Name: MQSINI                *#
** Type      : MQSeries machine-wide ini file      *#
# Function   : Define configuration data for all queue managers *#
**                *#
*****#
** Notes    :                *#
** 1) This file defines configuration data for all queue managers *#
**                *#
*****#
AllQueueManagers:
  MQSVolume=$DATA0                /Volume for the installation
  MQSExePath=$DATA0.ZMQSEXE       /Location of product executables
  QMDefaultVolume=$DATA0          /Default volume for queue manager creation
QueueManager:
  Name=MT01                        /A queue manager called MT01
  QMVolume=$DATA0                 /Volume of the queue manager
  QMSubvolume=MT01                /Subvolume prefix for the queue manager
DefaultQueueManager:
  Name=MT01                        /Name of the default queue manager (optional)

```

Figure 35. Example MQSeries configuration file (MQSINI). The MQSINI file is initialized during installation with the volume and subvolume information you provide.

Note: Because the MQSeries configuration file is used to locate the data associated with queue managers, a nonexistent or incorrect configuration file can cause some or all MQSeries commands to fail. Also, applications cannot connect to a queue manager that is not defined in the MQSeries configuration file.

Queue manager configuration file (QMINI)

A queue manager configuration file, QMINI, contains information relevant to a specific queue manager. There is one queue manager configuration file for each queue manager. It is created automatically when the queue manager with which it is associated is created.

The file is held in the subvolume of the queue manager. For example, the path and name for a configuration file for a queue manager called QMNAME could be \$VOLUME.QMNAME.QMINI.

Note: The queue manager name can be up to 48 characters in length. A subvolume name is generated based on the queue manager name. This process is known as *name transformation*, and ensures the name is both valid and unique.

What the queue manager configuration file contains

The stanzas that can appear in a queue manager configuration file, QMINI, are as follows:

Configuration

This stanza defines the global configurations for the queue manager.

The following entries can be modified:

```
DefaultMQSSName
HomeTerminalName
PathmonProcName
EMSCollectorName
MinIdleMCALU62Responders
MinIdleMCATCPResponders
MinIdleMCACallers
MinIdleLQMAgents
```

For more information about these entries, see “Configurable queue-manager properties” on page 62. Other entries in this stanza must not be changed.

DefaultProcess

This stanza defines the default values used for MQSeries processes. Entries in this stanza must not be changed.

ECBoss

This stanza defines the configuration of the MQSeries EC Boss process. The ExpectedNumECs entry defines the number of EC processes for this queue manager. This value must correspond with the PATHWAY configuration for the queue manager. For more information, see “Configurable queue-manager properties” on page 62. Other entries in this stanza must not be changed.

EC

The MCAAgentPriority and LQMAgentPriority entries of the EC stanza, which control the process priorities of agent processes, can be modified. For more information, see “Configurable queue-manager properties” on page 62. Other entries in this stanza must not be modified.

The following stanzas define the specific operating parameters for each MQSeries process type. Typically, you do not need to change the values of these parameters. However, see “Minimizing resource usage for remote operations” on page 187.

MCACaller

MCATCPResponder

MCALU62Responder

MQIServer

LQMAgent

ChannellInitiator

TCPListener

Authority

Provides the recommended mechanism for enabling and disabling the OAM for a queue manager. Set the MQAUTH flag to On or Off to enable or disable the OAM without having to add and remove the Service and Service Component stanzas.

Service

Specifies the name of one of the installable services, and the number of entry points to that service. There is one stanza for each service. These services are available:

- Authorization service
- Name service

The Object Authority Manager (OAM) is enabled by default: the authorization service stanza and its associated ServiceComponent stanza are present in QMINI by default.

You can disable the OAM simply by setting the MQAUTH flag in the Authority stanza to Off and restarting the queue manager. Alternatively, you can:

1. Delete the queue manager (using the **dltmqm** command)
2. Create the queue manager again (using the **crtmqm** command) with the MQSNOAUT environment variable set.
3. Delete the authorization service stanzas from QMINI.

The name service stanza must be added manually to QMINI if you want to enable the supplied name service.

ServiceComponent

These stanzas define the service component associated with a particular service. There can be more than one service component stanza for each service, but each service component stanza must match the corresponding service stanza. See the *MQSeries Programmable System Management* manual for more information.

TuningParameters

This stanza defines internal tuning parameters used by the local queue manager agents. You should not change these values.

Channels

This stanza contains information about the channels. As well as defining the maximum number of channels (MaxChannels) that can be defined for the queue

manager, a second entry (`MaxActiveChannels`) limits the number of channels that can be active simultaneously. `MaxActiveChannels` must not be greater than `MaxChannels`. The channels stanza also contains an entry (`ChanInitDiscInterval`) that can be used to tune the performance of the channel initiator. For more information about these entries, see “Configurable queue-manager properties” on page 62. Other entries in this stanza must not be modified.

See the *MQSeries Intercommunication* book for more information about channels.

TCPConfig

Specifies network-protocol configuration parameters. These stanzas override the default parameters for channels. Only stanzas representing changed default values are actually present.

`TCPKeepAlive`, if specified, causes TCP/IP periodically to check that the other end of the connection is still available. If it is not, the channel is closed.

See the *MQSeries Intercommunication* book for more information.

For information about modifying the `TCPPort`, `TCPNumListenerPorts`, and `TCPListenerPort` entries, see “Configurable queue-manager properties” on page 62.

Minimizing resource usage for remote operations

If a queue manager is known to be required to deal only with messages smaller than 4 MB, some resource can be saved by altering the `ExtPoolSize` parameter for the components used in remote operations.

These components are:

MCACaller
MCATCPResponder
MCALU62Responder
MQIServer

The default value of `ExtPoolSize` for these process types is 5000000. If these are reduced to 256000, for example, memory-resource usage by these processes is significantly decreased. However, messages greater than 256 KB will not be able to be sent over channels.

Care should be exercised when altering these parameters. The queue manager must be stopped and started before any changes take effect.

Queue manager configuration file

```
*****#
#* Module Name: QMINI                                     *#
#* Type       : MQSeries queue manager configuration file *#
# Function    : Define the configuration of a single queue manager *#
#*          *#
*****#
#* Notes     :                                           *#
#* 1) This file defines the configuration of the queue manager *#
#*          *#
*****#
Configuration:
  PathmonProcName=$p01p
  DefaultMQSSName=$p01s
  ServerClassName=MQS-ECBOSS
  EMSCollectorName=$0
  HomeTerminalName=$ZTN0.#PTY001C
  ShutdownFileName=SHUTDOWN
  TraceOptionsFileName=TRACEOPT
  RuntimeFileName=RUNTIME
  StatableFileName=STATABLE
  ChannelDefFileName=CHDEFS
  DefaultCCSID=819
  DefaultTraceOptions=0
  MaxIdleAgents=10
  MinIdleMCALU62Responders=0
  MinIdleMCATCPResponders=0
  MinIdleMCA callers=0
  MinIdleLQMAgents=1
  MaxIdleAgentReuse=10
DefaultProcess:
  ExeFileName=DEFAULT
  TraceVolSubvol=$DATA1.p101L
  TracePrefix=TR
  ErrorVolSubvol=$DATA1.p101L
  ErrorPrefix=ER
  DebugMode=0
  IPCCTimeOut=10000
  IPCCMemSetSize=32000
  MemSetSize=16000
  ExtPoolSize=256000
  IniPoolSize=256000
  Priority=175
```

Figure 36 (Part 1 of 5). Example queue manager configuration file (QMINI)

```
ECBoss:
  ExeFileName=MQECBOSS
  TraceVolSubvol=$DATA1.p101L
  TracePrefix=TR
  ErrorVolSubvol=$DATA1.p101L
  ErrorPrefix=ER
  DebugMode=0
  IPCCTimeOut=10000
  IPCCMemSetSize=32000
  MemSetSize=16000
  ExtPoolSize=5000000
  IniPoolSize=256000
  Priority=175
  ExpectedNumECs=1
EC:
  ExeFileName=MQEC
  TraceVolSubvol=$DATA1.p101L
  TracePrefix=TR
  ErrorVolSubvol=$DATA1.p101L
  ErrorPrefix=ER
  DebugMode=0
  IPCCTimeOut=10000
  IPCCMemSetSize=32000
  MemSetSize=16000
  ExtPoolSize=256000
  IniPoolSize=256000
  Priority=175
  LQMAgentExe=MLQMAG
  MCACallTerExe=MQMCACAL
  MCATCPResponderExe=MQTCPRES
  MCALU62ResponderExe=MLU6RES
  MCAAgentPriority=165
  LQMAgentPriority=165
  StopProcessTimer=3000
  IdleProcessTimer=3000
```

Figure 36 (Part 2 of 5). Example queue manager configuration file (QMNI)

Queue manager configuration file

```
MCACaller:
  ExeFileName=MQMCACAL
  TraceVolSubvol=$DATA1.p101L
  TracePrefix=TR
  ErrorVolSubvol=$DATA1.p101L
  ErrorPrefix=ER
  DebugMode=0
  IPCCTimeOut=10000
  IPCCMemSetSize=32000
  MemSetSize=16000
  ExtPoolSize=5000000
  IniPoolSize=256000
  Priority=175
MCATCPResponder:
  ExeFileName=MQTCPRES
  TraceVolSubvol=$DATA1.p101L
  TracePrefix=TR
  ErrorVolSubvol=$DATA1.p101L
  ErrorPrefix=ER
  DebugMode=0
  IPCCTimeOut=10000
  IPCCMemSetSize=32000
  MemSetSize=16000
  ExtPoolSize=5000000
  IniPoolSize=256000
  Priority=175
MCALU62Responder:
  ExeFileName=MQLU6RES
  TraceVolSubvol=$DATA1.p101L
  TracePrefix=TR
  ErrorVolSubvol=$DATA1.p101L
  ErrorPrefix=ER
  DebugMode=0
  IPCCTimeOut=10000
  IPCCMemSetSize=32000
  MemSetSize=16000
  ExtPoolSize=5000000
  IniPoolSize=256000
  Priority=175
```

Figure 36 (Part 3 of 5). Example queue manager configuration file (QMNI)

```
MQIServer:
  ExeFileName=MQMQISER
  TraceVolSubvol=$DATA1.p101L
  TracePrefix=TR
  ErrorVolSubvol=$DATA1.p101L
  ErrorPrefix=ER
  DebugMode=0
  IPCCTimeOut=10000
  IPCCMemSetSize=32000
  MemSetSize=16000
  ExtPoolSize=5000000
  IniPoolSize=256000
  Priority=175
LQMAgent:
  ExeFileName=MLQMAG
  TraceVolSubvol=$DATA1.p101L
  TracePrefix=TR
  ErrorVolSubvol=$DATA1.p101L
  ErrorPrefix=ER
  DebugMode=0
  IPCCTimeOut=50
  IPCCMemSetSize=32000
  MemSetSize=16000
  ExtPoolSize=120000
  IniPoolSize=200000
  Priority=175
ChannelInitiator:
  ExeFileName=RUNMQCHI
  TraceVolSubvol=$DATA1.p101L
  TracePrefix=TR
  ErrorVolSubvol=$DATA1.p101L
  ErrorPrefix=ER
  DebugMode=0
  IPCCTimeOut=10000
  IPCCMemSetSize=32000
  MemSetSize=16000
  ExtPoolSize=256000
  IniPoolSize=256000
  Priority=175
TCPLListener:
  ExeFileName=RUNMQLSR
  TraceVolSubvol=$DATA1.p101L
  TracePrefix=TR
  ErrorVolSubvol=$DATA1.p101L
  ErrorPrefix=ER
  DebugMode=0
  IPCCTimeOut=10000
  IPCCMemSetSize=32000
  MemSetSize=16000
  ExtPoolSize=256000
  IniPoolSize=256000
  Priority=175
```

Figure 36 (Part 4 of 5). Example queue manager configuration file (QMIMI)

Queue manager configuration file

```
| Authority:
|   MQAUTH=On
| Service:
|   Service=AuthorizationService
|   EntryPoints=9
| ServiceComponent:
|   Service=AuthorizationService
|   Name=MQSeries.TANDEM.auth.service
|   Module=MQOAM
|   ComponentDataSize=0
|   ComponentID=0
| TuningParameters:
|   KernelMemSetSize=32000
|   ObjCatMemSetSize=32000
|   QueueMemSetSize=16000
|   MQGETActiveQPoll=50
|   MQGETInactiveQPoll=1000
| Channels:
|   RetryAll=1
|   MaxChannels=10
|   MaxActiveChannels=10
|   MaxTries=3
|   MaxTriesInterval=10
|   ChanInitDiscInterval=10
| TCPConfig:
|   TCPPort=1414
|   TCPNumListenerPorts=1
|   TCPListenerPort=1414
|   TCPKeepAlive=1
```

Figure 36 (Part 5 of 5). Example queue manager configuration file (QMNI)

Editing configuration files

You can edit the default configuration files to alter the system defaults. However, before editing any configuration file, ensure that you have a backup that you can restore if necessary, and that any affected queue managers are stopped.

You might have to edit your configuration files if, for example:

- You lose a configuration file (recover from backup, if possible).
- You need to change the distribution of your queue manager across CPUs.
- You need to change your default queue manager (for example, if you accidentally delete the existing queue manager).
- You are advised to do so by your IBM Support Center.

For more information, see “Configurable queue-manager properties” on page 62.

Implementing changes to configuration files

If you edit a configuration file, the changes are not implemented immediately by the queue manager. Changes made to the MQSeries configuration file (MQSINI) take effect only when MQSeries queue managers are created or started. Changes made to a queue manager configuration file (QMINI) take effect when the queue manager is started. If the queue manager is running when you make the changes, you must stop and then restart the queue manager for any changes to be recognized by the system.

Recommendations for configuration files

When you create a new queue manager, you should:

- Back up the MQSeries configuration file
- Back up the new queue manager configuration file

Editing configuration files

Chapter 14. Problem determination

This chapter provides troubleshooting information for MQSeries for Tandem NSK. To determine a problem, you should list the symptoms and then trace them back to the cause.

Performance problems caused by the limitations of your hardware cannot be solved immediately. If you believe that the cause of the problem is in the MQSeries code, contact your IBM Support Center. This chapter contains these sections:

- “Making a preliminary check”
- “Common programming errors” on page 198
- “What to do next” on page 199
- “Application design considerations” on page 202
- “Effect of message length” on page 202
- “Error logs” on page 207
- “Dead-letter queues” on page 210
- “Configuration files and problem determination” on page 210
- “Using MQSeries trace” on page 210
- “First Failure Support Technology™ (FFST)” on page 212

Making a preliminary check

The cause of a problem can be in:

- MQSeries
- Your network
- An application
- The Tandem system software

The sections that follow provide questions that you might want to consider. Answer the questions and make a note of any issues that might be relevant to the problem.

Has MQSeries run successfully previously?

If MQSeries has not successfully run previously, you might not have set it up correctly. See Chapter 2, “Installing MQSeries for Tandem NSK Version 2.2.0.1” on page 19 to check that you have carried out all the steps correctly.

Are there any error messages?

MQSeries uses error logs to capture messages concerning the operation of MQSeries itself, any queue managers that you start, and error data coming from the channels that are in use. Check the error logs for any messages that have been recorded that are associated with your problem.

See “Error logs” on page 207 for information about the contents of the error logs and their locations.

Are there any return codes explaining the problem?

If your application gets a return code indicating that a Message Queue Interface (MQI) call has failed, refer to the *MQSeries Application Programming Reference* manual for a description of that return code.

Can you reproduce the problem?

If you can reproduce the problem, consider the following questions:

- Is the problem caused by a command or an equivalent administration request?
Does the operation work if it is entered by another method? If the command works if it is entered on the command line, but not otherwise, check that the command server has not stopped, and that the queue definition of the SYSTEM.ADMIN.COMMAND.QUEUE has not been changed.
- Is the problem caused by a program?
- Can you identify any application that always seems to be running in the system when the problem occurs? If so, examine the application for errors.

Have any changes been made since the last successful run?

When you are considering changes that might recently have been made, think about the MQSeries system, and also about the other programs it interfaces with, the hardware, and any new applications. Consider also the possibility that a new application that you are not aware of might have been run on the system.

- Have you changed, added, or deleted any queue definitions?
- Have you changed or added any channel definitions? Changes may have been made to either MQSeries channel definitions or any underlying communications definitions required by your application.
- Do your applications deal with return codes that they might get as a result of any changes you have made?

Has the application run successfully before?

If the problem appears to involve one particular application, consider whether the application has run successfully before.

Consider the following questions:

- Have any changes been made to the application since it last ran successfully?

If so, can the error exist in the new or modified part of the application. Check the changes and see if you can find an obvious reason for the problem. Is it possible to retry using a back level of the application?

- Have all the functions of the application been fully exercised previously?

Does the problem occur when part of the application that has never been invoked before is used for the first time? If so, the error might exist in that part of the application. Analyze what the application was doing when it failed, and check the source code in that part of the program for errors.

If a program has run successfully on previous occasions, check the current queue status, and the files that were being processed when the error occurred. It is possible that they contain some unusual data value that causes a rarely used path in the program to be invoked?

- Does the application check all return codes?

Has your MQSeries system been changed, such that your application does not check the return codes it receives as a result of the change. For example, does your application assume that the queues it accesses can be shared? If a queue has been redefined as exclusive, can your application deal with return codes indicating that it can no longer access that queue?

- Does the application run on other MQSeries systems?

Is there a difference in the way that this MQSeries system is set up which is causing the problem? For example, have the queues been defined with the same message length or priority?

- Have you set PARAM SAVE-ENVIRONMENT ON?

If not, you will receive MQRC 2058 on MQCONN calls. Set the PARAM in your application environment, as described in “TACL environment variables” on page 26.

If the application has not run successfully previously

If your application has not yet run successfully, you should examine it carefully for any errors.

Before you look at the code, and depending upon which programming language the code is written in, examine the output from the translator, or the compiler and linkage editor, if applicable, to see if any errors are reported.

If your application fails to translate, compile, or link-edit into the load library, it cannot run. See the *MQSeries Application Programming Reference* manual for information about building your application.

If the documentation shows that each of these steps was accomplished without error, you should consider the coding logic of the application. Do the symptoms of the problem indicate the function that is failing and, therefore, the piece of code in error? See “Common programming errors” on page 198 for some examples of common errors that cause problems with MQSeries applications.

Does the problem affect specific parts of the network?

You might be able to identify specific parts of the network that are affected by the problem (remote queues, for example). If the link to a remote message queue manager is not working, the messages cannot flow to a remote queue.

Check that the connection between the two systems is available, and that the intercommunication component of MQSeries has been started.

Check that messages are reaching the transmission queue, and check the local queue definition of the transmission queue and any remote queues.

Have you made any network-related changes, or changed any MQSeries definitions, that might account for the problem?

Common programming errors

Does the problem occur at specific times of the day?

If the problem occurs at specific times of day, it could be that it is dependent on system loading. Typically, peak system loading is at mid-morning and mid-afternoon, so these are the times when load-dependent problems are most likely to occur. (If your MQSeries network extends across more than one time zone, peak system loading might seem to occur at some other time of day.)

Is the problem intermittent?

An intermittent problem could be caused by failing to take into account the fact that processes can run independently of each other. For example, a program may issue an MQGET call, without specifying a wait option, before an earlier process has completed. An intermittent problem may also be seen if your application tries to get a message from a queue while the call that put the message is in-doubt (that is, before it has been committed or backed out).

Have you applied any service updates?

If a service update has been applied to MQSeries, check that the update action completed successfully and that no error message was produced.

- Did the update have any special instructions?
- Was any test run to verify that the update had been applied correctly and completely?
- Does the problem still exist if MQSeries is restored to the previous service level?
- If the installation was successful, check with the IBM Support Center for any patch error.
- If a patch has been applied to any other program, consider the effect it might have on the way MQSeries interfaces with it.

Common programming errors

The errors in the following list illustrate the most common causes of problems encountered while running MQSeries programs. You should consider the possibility that the problem with your MQSeries system could be caused by one or more of these errors:

- Assuming that queues can be shared, when they are in fact exclusive.
- Passing incorrect parameters in an MQI call.
- Passing insufficient parameters in an MQI call. This may mean that MQI cannot set up completion and reason codes for your application to process.
- Failing to check return codes from MQI requests.
- Passing variables with incorrect lengths specified.
- Passing parameters in the wrong order.
- Failing to initialize *MsgId* and *CorrelId* correctly.
- Failing to issue BEGINTRANSACTION when MQPMO_NO_SYNCPOINT is specified on the MQPUT command.

Problems with commands

You should be careful when including special characters, such as back slash (\) and double quotation marks ("), in descriptive text for some commands. If you use either of these characters in descriptive text, precede them with a \. That is, enter \\ or \" if you want \ or " in your text.

What to do next

When you have established that no changes have been made to your system, and that there are no problems with your application programs, choose the option that best describes the symptoms of your problem.

- "Have you obtained incorrect output?"
- "Have you failed to receive a response from a PCF command?"
- "Does the problem affect only remote queues?" on page 201
- "Is your application or MQSeries for Tandem NSK running slowly?" on page 201

Have you obtained incorrect output?

In this book, "incorrect output" refers to your application:

- Not receiving a message that it was expecting.
- Receiving a message containing unexpected or corrupted information.
- Receiving a message that it was not expecting, for example, one that was destined for a different application.

In all cases, check that any queue or queue manager aliases that your applications are using are correctly specified and accommodate any changes that have been made to your network.

If an MQSeries error message is generated, all of which are prefixed with the letters "AMQ," you should look in the error log. See "Error logs" on page 207 for further information.

Have you failed to receive a response from a PCF command?

If you have issued a command but you have not received a response, consider the following questions:

- Is the command server running?

Work with the **dspmqcsv** command to check the status of the command server. If the response to this command indicates that the command server is not running, use the **strmqcsv** command to start it. If the response to the command indicates that the SYSTEM.ADMIN.COMMAND.QUEUE is not enabled for MQGET requests, enable the queue for MQGET requests.

- Has a reply been sent to the dead-letter queue?

The dead-letter queue header structure contains a reason or feedback code describing the problem. See the *MQSeries Application Programming Reference* manual for information about the dead-letter queue header structure (MQDLH).

If the dead-letter queue contains messages, you can use the supplied browse sample application (AMQSBCG) to browse the messages using the MQGET

What next

call. The sample application steps through all the messages on a named queue for a named queue manager, displaying both the message descriptor and the message context fields for all the messages on the named queue.

- Has a message been sent to the error log?
See “Error logs” on page 207 for further information.
- Are the queues enabled for put and get operations?
- Is the *WaitInterval* long enough?

If your MQGET call has timed out, a completion code of MQCC_FAILED and a reason code of MQRC_NO_MSG_AVAILABLE are returned. (See the *MQSeries Application Programming Reference* manual for information about the *WaitInterval* field, and completion and reason codes from MQGET.)

- If you are using your own application program to put commands onto the SYSTEM.ADMIN.COMMAND.QUEUE, do you need to commit a transaction?
Unless you have specifically excluded your request message from syncpoint, you need to commit a transaction before attempting to receive reply messages.
- Are the MAXDEPTH and MAXMSGL attributes of your queues set sufficiently high?
- Are you using the *CorrelId* and *MsgId* fields correctly?

Set the values of *MsgId* and *CorrelId* in your application to ensure that you receive all messages from the queue.

Try stopping the command server and then restarting it, responding to any error messages that are produced.

If the system still does not respond, the problem could be with the queue manager. Try stopping the queue manager and then restarting. If the problem still occurs after restart, contact your IBM Support Center for help.

Are some of your queues failing?

If you suspect that the problem occurs with only a subset of queues, check the local queues that you think are having problems:

1. Display the information about each queue. You can use the MQSC command DISPLAY QUEUE to display the information.
2. Use the data displayed to do the following checks:
 - If CURDEPTH is at MAXDEPTH, this indicates that the queue is not being processed. Check that all applications are running normally.
 - If CURDEPTH is not at MAXDEPTH, check the following queue attributes to ensure that they are correct:
 - If triggering is being used:
 - Is the trigger monitor running?
 - Is the trigger depth too great? That is, does it generate a trigger event often enough?
 - Is the process name correct?
 - Is the process available and operational?
 - Can the queue be shared? If not, another application could already have it open for input.

- Is the queue enabled appropriately for GET and PUT?
- If there are no application processes getting messages from the queue, determine why this is so. It could be because the applications need to be started, a connection has been disrupted, or the MQOPEN call has failed for some reason.

Check the queue attributes IPPROCS and OPPROCS. These attributes indicate whether the queue has been opened for input and output. If a value is zero, it indicates that no operations of that type can occur. Note that the values may have changed and that the queue was open but is now closed.

You need to check the status at the time you expect to put or get a message.

If you are unable to solve the problem, contact your IBM Support Center for help.

Does the problem affect only remote queues?

If the problem affects only remote queues, check the following:

- Check that required channels have been started and are triggerable, and that any required initiators are running.
- Check that the programs that should be putting messages to the remote queues have not reported problems.
- If you use triggering to start the distributed queuing process, check that the transmission queue has triggering set on. Also, check that the trigger monitor is running.
- Check the error logs for messages indicating channel errors or problems.
- If necessary, start the channel manually. See the *MQSeries Intercommunication* book for information about how to do this.

For information about how to define channels, see Appendix J, “Setting up communications” on page 333 and the *MQSeries Intercommunication* book.

Is your application or MQSeries for Tandem NSK running slowly?

If your application is running slowly, this could indicate that it is in a loop, or waiting for a resource that is not available.

This could also be caused by a performance problem. Perhaps it is because your system is operating near the limits of its capacity. This type of problem is probably worst at peak system load times, typically at mid-morning and mid-afternoon. (If your network extends across more than one time zone, peak system load might seem to occur at some other time.)

Examine the priority of application and queue manager processes using the STATUS command. A loop causes the priority of the process to be reduced gradually to zero by NSK.

Check that each of the CPUs in the NSK system is being utilized fully. If some processors are only lightly loaded, your NSK system needs balancing. Consider adding ECs to other processors to distribute MQSeries workload.

A performance problem may be caused by a limitation of your hardware.

Application design considerations

Note: After a fresh install of MQSeries or a cold load of the Tandem NSK system, MQSeries executables might take longer to run than expected when they are first invoked. This is because the Tandem NSK operating system goes through a “fixup” phase, during which it ensures that all external declarations are resolved.

If you find that performance degradation is not dependent on system loading, but happens sometimes when the system is lightly loaded, a poorly designed application program is probably to blame. This could manifest itself as a problem that only occurs when certain queues are accessed.

The following symptoms might indicate that MQSeries is running slowly:

- Your system is slow to respond to MQSeries commands.
- Repeated displays of the queue depth indicate that the queue is being processed slowly for an application with which you would expect a large amount of queue activity.

If the performance of your system is still degraded after reviewing the above possible causes, the problem may lie with MQSeries for Tandem NSK itself. If you suspect this, you need to contact your IBM Support Center for assistance.

Application design considerations

There are a number of ways in which poor program design can affect performance. These can be difficult to detect because the program can appear to perform well, while impacting the performance of other tasks. Several problems specific to programs making MQSeries calls are discussed in the following sections.

For more information about application design, see the *MQSeries Application Programming Guide*.

Effect of message length

Although MQSeries allows messages to hold up to 4 MB of data, the amount of data in a message affects the performance of the application that processes the message. To achieve the best performance from your application, you should send only the essential data in a message; for example, in a request to debit a bank account, the only information that may need to be passed from the client to the server application is the account number and the amount of the debit.

Searching for a particular message

The MQGET call usually retrieves the first message from a queue. If you use the message and correlation identifiers (*MsgId* and *CorrelId*) in the message descriptor to specify a particular message, the queue manager has to search the queue until it finds that message. Using the MQGET call in this way affects the performance of your application. You can disable message retrieval by *MsgId* or *CorrelId* or both for a queue using the **altmqfls** command. For more information, see “altmqfls (Alter queue file attributes)” on page 220.

Queues that contain messages of different lengths

If the messages on a queue are of different lengths, to determine the size of a message, your application could use the MQGET call with the *BufferLength* field set to zero so that, even though the call fails, it returns the size of the message data. The application could then repeat the call, specifying the identifier of the message it measured in its first call and a buffer of the correct size. However, if there are other applications serving the same queue, you might find that the performance of your application is reduced because its second MQGET call spends time searching for a message that another application has retrieved in the time between your two calls.

If your application cannot use messages of a fixed length, another solution to this problem is to use the MQINQ call to find the maximum size of messages that the queue can accept, then use this value in your MQGET call. The maximum size of messages for a queue is stored in the *MaxMsgLength* attribute of the queue. This method could use large amounts of storage, however, because the value of this queue attribute could be as high as 4 MB, the maximum allowed by MQSeries for Tandem NSK.

Frequency of syncpoints

Programs that issue numerous MQPUT calls within syncpoint, without committing them, can cause performance problems. Affected queues can fill up with messages that are currently inaccessible, while other tasks might be waiting to get these messages. This has implications in terms of: storage; TMF audit trail usage; and processes tied up with tasks that are attempting to get messages.

Use of the MQPUT1 call

Use the MQPUT1 call only if you have a single message to put on a queue. If you want to put more than one message, use the MQOPEN call, followed by a series of MQPUT calls and a single MQCLOSE call.

Incorrect output

The term “incorrect output” can be interpreted in many different ways. For the purpose of problem determination within this book, the meaning is explained in “Have you obtained incorrect output?” on page 199.

Two types of incorrect output are discussed in this section:

- Messages that do not appear when you are expecting them
- Messages that contain the wrong information, or information that has been corrupted

Additional problems that you might find if your application includes the use of distributed queues are also discussed.

Messages that do not appear on the queue

If messages do not appear when you are expecting them, check for the following:

- Has the message been put on the queue successfully?
- Has the queue been defined correctly. For example, is MAXMSGL sufficiently large?

Incorrect output

- Is the queue enabled for putting?
- Is the queue already full? This could mean that an application was unable to put the required message on the queue.
- Are you able to get any messages from the queue?
- Do you need to take a syncpoint?

If messages are being put or retrieved within syncpoint, they are not available to other tasks until the unit of recovery has been committed.

- Is your wait interval long enough?
You can set the wait interval as an option for the MQGET call. You should ensure that you are waiting long enough for a response.
- Are you waiting for a specific message that is identified by a message or correlation identifier (*MsgId* or *CorrelId*)?

Check that you are waiting for a message with the correct *MsgId* or *CorrelId*. A successful MQGET call sets both these values to that of the message retrieved, so you may need to reset these values in order to get another message successfully.

Also, check whether you can get other messages from the queue.

- Can other applications get messages from the queue?
- Was the message you are expecting defined as persistent?
If not, and MQSeries has been restarted, the message has been lost.
- Has another application got exclusive access to the queue?

If you are unable to find anything wrong with the queue, and MQSeries is running, make the following checks on the process that you expected to put the message on to the queue:

- Did the application get started?
If it should have been triggered, check that the correct trigger options were specified.
- Did the application stop?
- Is a trigger monitor running?
- Was the trigger process defined correctly?
- Did the application complete correctly?
Look for evidence of an abnormal end in the job log.
- Did the application commit its changes, or were they backed out?

If multiple transactions are serving the queue, they can conflict with one another. For example, suppose one transaction issues an MQGET call with a buffer length of zero to find out the length of the message, and then issues a specific MQGET call specifying the *MsgId* of that message. However, in the meantime, another transaction issues a successful MQGET call for that message, so the first application receives a reason code of MQRC_NO_MSG_AVAILABLE. Applications that are expected to run in a multiserver environment must be designed to cope with this situation.

Consider that the message could have been received, but that your application failed to process it in some way. For example, did an error in the expected format of the message cause your program to reject it? If this is the case, refer to “Messages that contain unexpected or corrupted information” on page 205.

Messages that contain unexpected or corrupted information

If the information contained in the message is not what your application was expecting, or has been corrupted in some way, consider the following points:

- Has your application, or the application that put the message onto the queue, changed?

Ensure that all changes are simultaneously reflected on all systems that need to be aware of the change.

For example, the format of the message data may have been changed, in which case, both applications must be recompiled to pick up the changes. If one application has not been recompiled, the data will appear corrupt to the other.

- Is an application sending messages to the wrong queue?

Check that the messages your application is receiving are not really intended for an application servicing a different queue. If necessary, change your security definitions to prevent unauthorized applications from putting messages on to the wrong queues.

If your application has used an alias queue, check that the alias points to the correct queue.

- Has the trigger information been specified correctly for this queue?

Check that your application should have been started; or should a different application have been started?

If these checks do not enable you to solve the problem, you should check your application logic, both for the program sending the message, and for the program receiving it.

Problems with incorrect output when using distributed queues

If your application uses distributed queues, you should also consider the following points:

- Has MQSeries been correctly installed on both the sending and receiving systems, and correctly configured for distributed queuing?

- Are the links available between the two systems?

Check that both systems are available, and connected to MQSeries. Check that the connection between the two systems, and the channels between the two queue managers, are active.

- Is triggering set on in the sending system?
- Is the message you are waiting for a reply message from a remote system?

Check that triggering is activated in the remote system.

- Is the queue already full?

Incorrect output

This could mean that an application was unable to put the required message onto the queue. If this is so, check if the message has been put onto the dead-letter queue.

The dead-letter queue header contains a reason or feedback code explaining why the message could not be put onto the target queue. See the *MQSeries Application Programming Reference* manual for information about the dead-letter queue header structure.

- Is there a mismatch between the sending and receiving queue managers?

For example, the message length could be longer than the receiving queue manager can handle.

- Are the channel definitions of the sending and receiving channels compatible?

For example, a mismatch in sequence number wrap stops the distributed queuing component. See the *MQSeries Intercommunication* book for more information about distributed queuing.

- Have you started a TCP/IP listener?

If you are using TCP/IP as a communications protocol for MQSeries communications to the Tandem, a TCP/IP listener process must be running. See “Specifying and controlling TCP/IP listeners” on page 48 for more information.

- Is the TCP/IP listener listening on the correct TCP/IP port?

The TCP/IP listener listens on a port defined on a `TCPListenerPort` entry in the `TCPConfig` stanza of the `QMINI` file for your queue manager. See “TCP/IP ports listened on by the queue manager” on page 66 for more information.

- Is the TCP/IP process name correct ?

If you are using the TCP/IP communications protocol, is your Tandem system using the default process name (`$ztc0`) for the TCP/IP process? If not, you must alter some of the server classes in your MQSeries pathway to enable the correct process name to be used by MQSeries channels. See “Reconfiguring a queue manager for a nondefault TCP/IP process” on page 67 for more information.

- Is the SNA channel defined with `AUTOSTART(ENABLED)`?

If:

- You are running MQSeries channels using SNA as a communications protocol.

and

- The channel type on Tandem is one that is waiting to be initiated from a remote MQSeries system (for example, a `RECEIVER`).

and

- The remote system is having problems starting the channel.

an LU 62 responder process might not be running for your channel. Check that the channel is defined with `AUTOSTART(ENABLED)`. See “SNA channels” on page 333 for more information.

- Is data conversion involved? If the data formats between the sending and receiving applications differ, data conversion is necessary. Automatic

conversion occurs when the MQGET is issued if the format is recognized as one of the built-in formats.

If the data format is not recognized as a built-in format, a data conversion exit can be used to allow you to perform the translation with your own routines. Check that your routine is being loaded correctly.

See the *MQSeries Application Programming Guide* for more information about data conversion.

Error logs

MQSeries for Tandem NSK uses a number of error logs to capture messages concerning the operation of MQSeries itself, any queue managers that you start, and error data coming from the channels that are in use.

The location of the error logs depends on whether the queue manager name is known and whether the error is associated with a client.

- If the queue manager name is known and the queue manager is available:
`<QMVOL>.<SUBVOL>L.MQERRLG1`
- If the queue manager is not available:
`<MQSVOL>.ZMQSSYS.MQERRLG1`
- First Failure Symptom Trap (FFST) in
`<QMVOL>.<SUBVOL>.FDnnnnn`
- see “How to examine the FFSTs” on page 212.

Log files

The error log subvolume can contain up to three error log files named:

- MQERRLG1
- MQERRLG2
- MQERRLG3

After you have created a queue manager, three error log files are created when they are needed by the queue manager. These files are called MQERRLG1, MQERRLG2, and MQERRLG3, and are placed in the subvolume of each queue manager that you create.

As error or log messages are generated they are placed in MQERRLG1. When MQERRLG1 is filled it is copied to MQERRLG2. Before the copy, MQERRLG2 is copied to MQERRLG3. The previous contents, if any, of MQERRLG3 are discarded.

The latest error messages are thus always placed in MQERRLG1, the other files being used to maintain a history of error messages.

All messages relating to channels are also placed in the appropriate queue manager's errors files unless the name of their queue manager is unknown or the queue manager is unavailable. When the queue manager name is unavailable or its name cannot be determined, channel-related messages are placed in the system error log (ZMQSSYS.MQERRLG1).

Error logs

To examine the contents of any error log file, you can use either the `fup copy` command or your usual Tandem NSK editor in read-only mode. (If you open the error log in update mode, error messages might be lost.)

Early errors

There are a number of special cases where the above error logs have not yet been established and an error occurs. MQSeries attempts to record any such errors in an error log. The location of the log depends on how much of a queue manager has been established.

If, due to a corrupt configuration file for example, no location information can be determined, errors are logged to an error file that is created at installation time on the ZMQSSYS subvolume in the file MQERRLG1.

For further information about configuration files, see Chapter 13, "Configuration files" on page 183.

Operator messages

In MQSeries for Tandem NSK, operator messages identify normal errors, typically caused directly by users doing things like using parameters that are not valid on a command. These messages are written to the associated window, if any, and are also written to a file in the queue manager subvolume.

Errors that can be associated with a particular queue manager are logged to MQERRLG1 in the queue manager's log subvolume. Those that cannot be linked to a defined and operational queue manager are logged in the MQERRLG1 file located in subvolume ZMQSSYS.

Example error log

This example shows part of an MQSeries for Tandem NSK error log:

```

...
08/01/95 11:41:56 AMQ8003: MQSeries queue manager started.
EXPLANATION: MQSeries queue manager Janet started.
ACTION: None.
-----
08/01/95 11:56:52 AMQ9002: Channel program started.
EXPLANATION: Channel program 'JANET' started.
ACTION: None.
-----
08/01/95 11:57:26 AMQ9208: Error on receive from host 'camelot
(9.20.12.34)'.
EXPLANATION: An error occurred receiving data from 'camelot
(9.20.12.34)' over TCP/IP. This may be due to a communications failure.
ACTION: Record the TCP/IP return code 232 (X'E8') and tell the
systems administrator.
-----
08/01/95 11:57:27 AMQ9999: Channel program ended abnormally.
EXPLANATION: Channel program 'JANET' ended abnormally.
ACTION: Look at previous error messages for channel program
'JANET' in the error files to determine the cause of the failure.
-----
08/01/95 14:28:57 AMQ8004: MQSeries queue manager ended.
EXPLANATION: MQSeries queue manager Janet ended.
ACTION: None.
-----
08/02/95 15:02:49 AMQ9002: Channel program started.
EXPLANATION: Channel program 'JANET' started.
ACTION: None.
-----
08/02/95 15:02:51 AMQ9001: Channel program ended normally.
EXPLANATION: Channel program 'JANET' ended normally.
ACTION: None.
-----
08/02/95 15:09:27 AMQ7030: Request to quiesce the queue manager
accepted. The queue manager will stop when there is no further
work for it to perform.
EXPLANATION: You have requested that the queue manager end when
there is no more work for it. In the meantime, it will refuse
new applications that attempt to start, although it allows those
already running to complete their work.
ACTION: None.
-----
08/02/95 15:09:32 AMQ8004: MQSeries queue manager ended.
EXPLANATION: MQSeries queue manager Janet ended.
ACTION: None.
...

```

EMS events

An EMS event is generated for each error entry made in the MQERRLG1 file. For more information about EMS events, see “Event Management Service (EMS) events” on page 171.

Dead-letter queues

Messages that cannot be delivered for some reason are placed on the dead-letter queue. You can check whether the queue contains any messages by issuing an MQSC DISPLAY QUEUE command. If the queue contains messages, you can use the provided browse sample application (MQSBCG0E) to browse messages on the queue using the MQGET call. The sample application steps through all the messages on a named queue for a named queue manager, displaying both the message descriptor and the message context fields for all the messages on the named queue.

You must decide how to dispose of any messages found on the dead-letter queue, depending on the reasons for the messages being put on the queue.

Problems may occur if you do not have a dead-letter queue on each queue manager you are using. The supplied file amqscoma (located in subvolume ZMQSSMPL), which you use as input to the runmqsc command, creates the default objects for a queue manager, including a dead-letter queue called SYSTEM.DEAD.LETTER.QUEUE. When you have created this dead-letter queue, you must alter the DEADQ attribute of the queue manager using **runmqsc**.

Configuration files and problem determination

Configuration file errors typically prevent queue managers from being found and result in “queue manager unavailable” type errors.

There are several checks you can make on the configuration files:

- Ensure that the configuration files exist.
- Ensure that they have appropriate permissions.
- Ensure that the MQSeries configuration file references the correct queue manager and directories.

Using MQSeries trace

MQSeries for Tandem NSK uses the following commands for the trace facility:

- **strmqtrc** – see “strmqtrc (Start MQSeries trace)” on page 273
- **dspmqtrc** – see “dspmqtrc (Display MQSeries formatted trace output)” on page 248
- **endmqtrc** – see “endmqtrc (End MQSeries trace)” on page 255

The trace facility uses one file for each entity being traced, with the trace information being recorded in the appropriate file.

Trace options are specified in the QMINI file.

Note: With MQSeries for Tandem NSK, tracing can also be controlled via the Queue Manager menu of the Message Queue Management (MQM) facility.

Trace file names

Trace file names are constructed in the error log subvolume as follows:

TRccpppp

where *ccpppp* is the process identifier (PID) of the process producing the trace.

The PID is made up of:

cc, the CPU number.

pppp, the Process number.

If the tracing utility encounters a trace file of an identical process identifier that has not been deleted, it replaces the final character of the process number with a letter, giving 26 processes of the same PID the opportunity to write output. For example, the first trace file for PID 00, 0315 would be TR000315. For a second process started on completion of process 00, 0315 with the same PID, the trace file would be TR00031A.

Note: Because of this restriction, trace files should be purged from the system as soon as they have been examined.

Sample trace data

The following sample is an extract from a trace:

ID	ELAPSED_MSEC	DELTA_MSEC	APPL	SYSCALL	KERNEL	INTERRUPT
...						
30d	0	0	MQS CEI	Exit!. 12484.1	xcsWaitEventSem	rc=10806020
30d	0	0	MQS CEI	Exit! 12484.1	zcpReceiveOnLink	rc=20805311
30d	0	0	MQS FNC	Entry 12484.1	zxcProcessChildren	
30d	0	0	MQS CEI	Entry. 12484.1	xcsRequestMutexSem	
30d	1	0	MQS CEI	Entry.. 12484.1	xcsHSHMEMBtoPTR	
30d	1	0	MQS CEI	Exit... 12484.1	xcsHSHMEMBtoPTR	rc=00000000
30d	1	0	MQS FNC	Entry.. 12484.1	xllSemGetVal	
30d	1	0	MQS FNC	Exit... 12484.1	xllSemGetVal	rc=00000000
30d	1	0	MQS FNC	Entry.. 12484.1	xllSemReq	
30d	1	0	MQS FNC	Exit... 12484.1	xllSemReq	rc=00000000
30d	1	0	MQS CEI	Exit.. 12484.1	xcsRequestMutexSem	rc=00000000
30d	2	0	MQS CEI	Entry. 12484.1	xcsReleaseMutexSem	
30d	2	0	MQS CEI	Entry.. 12484.1	xcsHSHMEMBtoPTR	
30d	2	0	MQS CEI	Exit... 12484.1	xcsHSHMEMBtoPTR	rc=00000000
30d	2	0	MQS FNC	Entry.. 12484.1	xllSemRel	
30d	2	0	MQS FNC	Exit... 12484.1	xllSemRel	rc=00000000
30d	2	0	MQS CEI	Exit.. 12484.1	xcsReleaseMutexSem	rc=00000000
30d	2	0	MQS CEI	Entry. 12484.1	xcsHSHMEMBtoPTR	
...						

Figure 37. Sample trace

Notes:

1. In this example the data is truncated. In a real trace, the complete function names and return codes are present.
2. The return codes are given as values, not literals.

First Failure Support Technology™ (FFST)

FFST errors are normally severe, and indicate either a configuration problem with the system or an MQSeries internal error. In most cases, the queue manager remains operational, though there may be a brief interruption of service to some or all applications. FFSTs are referenced in the file ZMQSSYS.MQSYSLOG.

How to examine the FFSTs

The files are named FDnnnnn, where:

nnnnn Is the process ID reporting the error

When a process creates an FFST report, it also generates an EMS event.

A typical FFST report is shown in Figure 38.

```

+-----+
| MQSeries First Failure Symptom Report |
| ===== |
| Date/Time            :- February 1 12:23:26    1999 |
| Host Name            :- \HURSLEY |
| PIDS                :- 5697A17 |
| LVLS                :- 221 |
| Product Long Name   :- MQSeries Version 2 for TANDEM NSK |
| Vendor              :- IBM |
| Probe Id            :- RM020011 |
| Application Name    :- MQM |
| Component           :- rrxOpenSync |
| Build Date          :- Feb 5 1998 |
| Exe File Name       :- \HURSLEY.$DATA0.ZMQSEXE.MQMCACAL |
| UserID              :- MQM.MANAGER |
| Process File Name   :- \HURSLEY.$Z734:15441941 |
| Node number         :- 1 |
| CPU                 :- 0 |
| PIN                 :- 339 |
| QueueManager       :- MT01 |
| Major Errorcode    :- xecF_E_UNEXPECTED_RC |
| Minor Errorcode    :- Unknown(A) |
| Probe Type         :- MSGAMQ6118 |
| Probe severity     :- Severity 2: error |
| Probe Description   :- AMQ6118: An internal MQSeries error has occurred. |
| Text                :- Error creating synch file |
| |
| Arith1              :- 10 (0xa) |
| Comment1            :- error 0000000010 in function 0000000020 |
+-----+

```

Figure 38. Sample First Failure Symptom Report

However, there is one set of problems that they may be able to solve. If the FFST shows “out of resource” or “out of space on device” descriptions, it is likely that the relevant system limit is exceeded.

To resolve the problem, increase the appropriate limit and restart the queue manager. See “Configuration of other NonStop Kernel resources” on page 27 for further details.

Part 2. Reference

Chapter 15. The MQSeries control commands

This chapter contains reference material for the control commands used with MQSeries for Tandem NSK.

Control commands summary

The following control commands are supported by MQSeries for Tandem NSK via TACL macros and compiled programs:

- almqfls (alter queue file attributes)
- almqusr (alter MQSeries user information)
- cleanrdf (RDF housekeeping utility)
- cnv1520 (convert V1.5.1 definitions to V2.2.0.1)
- cnvclchl (convert client channel definitions)
- cnvmsgs (convert V1.5.1 messages to V2.2.0.1)
- crtmqcvx (data conversion)
- crtmqm (create queue manager)
- dlrmqm (delete queue manager)
- dspmqaut (display authority)
- dspmqcsv (display command server)
- dspmqfls (display MQSeries file attributes)
- dspmqtrc (display MQSeries formatted trace output)
- dspmqusr (display MQSeries user information)
- endmqcsv (end command server)
- endmqm (end queue manager)
- endmqtrc (end MQSeries trace)
- instmqm (install MQSeries for Tandem NSK)
- runmqchi (run channel initiator)
- runmqchl (run channel)
- runmqdlq (run dead-letter queue handler)
- runmqslr (run TCP/IP listener)
- runmqsc (run MQSeries commands)
- runmqtrm (start trigger monitor)
- setmqaut (set/reset authority)
- strmqcsv (start command server)
- strmqm (start queue manager)
- strmqtrc (start MQSeries trace)
- upgmqm (upgrade V2.2 queue manager)

Detailed descriptions of these commands are provided in the remainder of this chapter.

Notes:

1. Flags, which are single-character identifiers preceded by a dash (for example, -v on the **runmqsc** command), must be specified in lowercase.
2. Usage messages are displayed if control commands are invoked with -?, ?, or with no parameters when parameters are expected.

Using names

The names for the following MQSeries objects can be a maximum of 48 characters:

- Queue managers
- Queues
- Process definitions

The maximum length of channel names is 20 characters.

The characters that can be used for all MQSeries names are:

- Uppercase A - Z
- Lowercase a - z
- Numerics 0 - 9
- Period (.)
- Underscore (_)
- Forward slash (/)
- Percent sign (%)

Notes:

1. Forward slash and percent are special characters. If you use either of these characters in a name, the name must be enclosed in double quotation marks whenever it is used.
2. Leading or embedded blanks are not allowed.
3. National language characters are not allowed.
4. Names may be enclosed in double quotation marks, but this is essential only if special characters are included in the name.

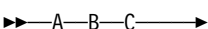
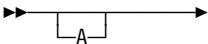
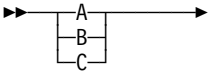
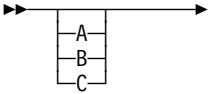
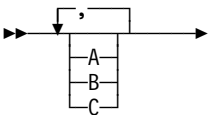
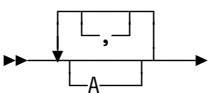
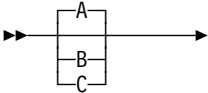
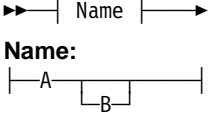
How to read syntax diagrams

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 10. 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. Name: —A— —B—
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.

altmqfls (Alter queue file attributes)

Purpose

You use the **altmqfls** command to:

- Move the message files that belong to a predefined local queue to a different volume to distribute disk I/O across volumes.

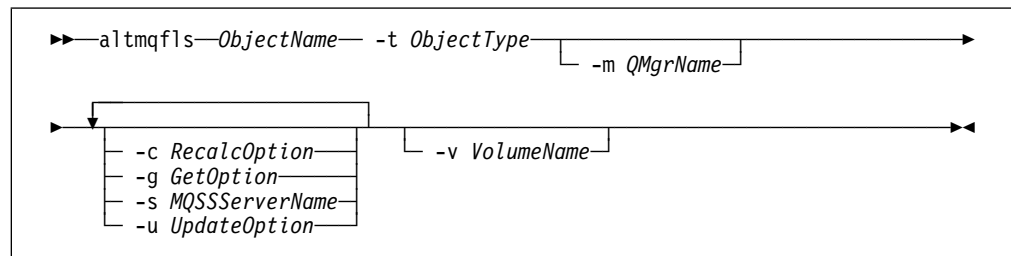
or

- Change the recalculation, update, and message-retrieval options for an object, and the responsible MQSS Server for an object.

A single **altmqfls** command cannot be used for both purposes.

By changing the recalculation, update, and message-retrieval options, you can reduce the I/O activity associated with accessing a queue. Changing the MQSS Server responsible for an object is a way of distributing the processing load of maintaining status data across your system.

Syntax



Required parameters

ObjectName

Is the name of the permanent local queue whose message files are to be relocated. The queue must not be open, nor must it contain uncommitted messages.

-t *ObjectType*

Identifies a permanent local queue. *ObjectType* may be ql, qlocal, QL, or QLOCAL. It must be specified.

Optional parameters

-c *RecalcOption*

Is the recalculation option. It must be one of:

OPEN	Recalculate CURDEPTH on every MQOPEN.
CLOSE	Recalculate CURDEPTH on every MQCLOSE.
BOTH	Recalculate CURDEPTH on every MQOPEN and every MQCLOSE.
FIRSTOPEN	Recalculate CURDEPTH whenever the first opener opens the queue.
LASTCLOSE	Recalculate CURDEPTH whenever the last opener closes the queue.
FIRSTLAST	Recalculate CURDEPTH whenever the first opener opens the queue and whenever the last opener closes the queue.
NEVER	Never recalculate CURDEPTH.
<i>n</i>	Recalculate CURDEPTH every <i>n</i> hours, where <i>n</i> is an integer.

-g *GetOption*

Is the message-retrieval option. It must be set to NONE or to any combination of the characters MCB for retrieval by *MsgId* (M), *CorrelId* (C), or both (B).

-m *QMgrName*

Is the name of the queue manager to which the local queue belongs. The queue manager must have been started. If no queue manager name is specified, the default queue manager is used.

-s *MQSSServerName*

Is the name of an MQSS Server process that is to be responsible for the status data for this object.

-v *VolumeName*

Is a Tandem volume name (for example, \$DEV). This value is required if you are using **altnmqfls** to move message files to a different volume.

Return codes

- 0** Command completed normally
- 10** Command completed but not entirely as expected
- 20** An error occurred during processing

Examples

In the following example, message files belonging to the local queue `flint.queue`, owned by queue manager `target.queue.mgr`, are moved to volume `$DATA3`.

```
altnmqfls -m target.queue.mgr -t ql -v $DATA3 flint.queue
```

In the following example, the recalculation interval, message-retrieval options, and MQSS Server process name are being changed for an object:

```
altnmqfls -m target.queue.mgr -t ql -c NEVER -g CB -s $TQS2 flint.queue
```

This command results in local queue `flint.queue`, which belongs to `target.queue.mgr`:

- Supporting MQGET by *CorrelId* or by both *MsgId* and *CorrelId*
- Being moved to an MQSS Server with a process name of `$TQS2`
- Never having its CURDEPTH statistics recalculated

Related commands

dspmqfls Display MQSeries files

altmqsr (Alter MQSeries user information)

Purpose

Use the **altmqsr** command to define or remove a principal corresponding to a Tandem NSK user ID that will have access to MQSeries.

Syntax

```
▶▶altmqsr -m QMgrName -p PrincipalName [-u TandemUserId] [-r]▶▶
```

Description

You can use this command to:

- Create a principal (that is, to grant access to a queue manager to a Tandem NSK user ID).
- Remove a principal (that is, to revoke access to a queue manager from a Tandem user ID).
- Change a principal definition.

When `-u TandemUserId` is specified, **altmqsr** creates a principal, if one does not already exist, or changes the existing definition. *TandemUserId* can be specified as a Tandem Administrative user ID or, if SAFEGUARD is running, as a SAFEGUARD alias.

When `-r` is specified, the principal is deleted from the principal database.

You must specify either `-u TandemUserId` or `-r`.

Required parameters

- m** *QMgrName*
Is the name of the queue manager to which the principal belongs.
- p** *PrincipalName*
Is the name of the principal to be created, changed, or removed.
- u** *TandemUserId*
Is the Tandem NSK Administrative user ID or SAFEGUARD alias to be associated with the principal definition.
- r** Specifies that the principal definition is to be removed from the queue manager.

Return codes

- 0** Successful operation
- 36** Invalid arguments supplied
- 69** Storage not available
- 71** Unexpected error

Examples

To add a principal mquser1 mapping to a Tandem user ID mqtest.fred:

```
altmqusr -m MT02 -p mquser1 -u mqtest.fred
```

To add a principal mquser2 mapping to group group.user01:

```
altmqusr -m MT02 -p mquser2 -u group.user01
```

To see the results of these commands, use the **dspmqsusr** command, as described in “dspmqsusr (Display MQSeries user information)” on page 249.

To remove principal mquser1:

```
altmqusr -m MT02 -p mquser1 -r
```

Related commands

dspmqsusr Display MQSeries user information

cleanrdf (Perform RDF housekeeping)

Purpose

Use the **cleanrdf** utility to perform routine housekeeping on the primary system queue manager in an RDF environment. The **cleanrdf** utility completes the removal of files that have been logically deleted on both the primary and backup systems. The utility also duplicates some nonaudited databases to the correct location on the backup system.

Note that the utility invoked by **cleanrdf** traverses the entire object catalog and message database, so some degradation of performance is likely to occur while the utility is running.

Syntax

```
cleanrdf -b BkupSystem [-m QMgrName]
```

Required parameters

-b *BkupSystem*

Is the Tandem NSK system name of the RDF backup site for this queue manager. *BkupSystem* is specified in the form *lname* (as is standard in the Tandem NSK environment).

Optional parameters

-m *QMgrName*

Is the name of the queue manager for which **cleanrdf** is to be run. If no queue-manager name is specified, **cleanrdf** is run against the default queue manager.

Return codes

- 0** Command completed normally
- 20** An error occurred during processing

Examples

In the following example, **cleanrdf** is run against the queue manager `test.queue.mgr`. Tandem NSK node `\HAWK` has been configured as the backup RDF site for this queue manager.

```
cleanrdf -b \HAWK -m test.queue.mgr
```

cnv1520 (Convert V1.5.1 definitions to V2.2.0.1)

Purpose

Use the **cnv1520** command to convert MQSeries for Tandem NSK Version 1.5.1 queue and channel definitions for use with MQSeries for Tandem NSK Version 2.2.0.1. The output from this command is a file of MQSeries commands that you can supply as input to **runmqsc**.

Syntax

```
▶▶—cnv1520— [ -c ChdefFile ] [ -q QdefFile ] [ -o CommandFile ]▶▶
```

Optional parameters

- c** *ChdefFile*
Identifies the file containing the channel definitions to be converted. The default file-name is CHANDESC.
- q** *QdefFile*
Identifies the file containing the queue definitions to be converted. The default file-name is QDESC.
- o** *CommandFile*
Identifies the output file of MQSeries commands (MQSC) created by **cnv1520**. The default file-name is CNVMQSC.

Examples

The following command converts MQSeries for Tandem NSK V1.5.1 channel and queue definitions from their V1.5.1 representations in the files CHANDESC and QDESC to an MQSC definition file (in this case, CNVMQSC by default), which can be input to a V2.2.0.1 queue manager via **runmqsc**:

```
$DATA0 MQSDATA 96> cnv1520 -c CHANDESC -q QDESC

*****/
*                                          */
* CHANNEL DEFINITIONS                    */
*                                          */
*****/
MQSC file CNVMQSC opened
Opened file CHANDESC

Processing channel - VSE2.ET01.SDR0.0001

Channel processing completed normally

*****/
*                                          */
* QUEUE DEFINITIONS                      */
*                                          */
*****/
MQSC file CNVMQSC opened
Opened file QDESC

- ANNE.VSE2.SDR0.LOCAL
- DEAD.QUEUE

Queue processing completed normally
```

Related commands

cnvmqsc Convert V1.5.1 messages to V2.2.0.1.

cnvclchl (Convert client channel definitions)

Purpose

Use the **cnvclchl** command to convert the client channel definition file, created for CLNTCONN channels by MQSC, from a Tandem structured file to an unstructured format acceptable to MQSeries clients.

Syntax

```
▶▶—cnvclchl— -m QMgrName — [ -o OutputFile ] —▶▶
```

Required parameters

-m *QMgrName*
Identifies the queue manager that owns the channel definitions file (CCHDEFS) to be converted. This value is required.

Optional parameters

-o *OutputFile*
Identifies the file that will contain the converted definitions. The default file-name is AMQCLCHL.

Examples

The following command converts the Tandem structured client channel definition file for queue manager MT01 to an unstructured file. Two client connection channel definitions are contained in the output file AMQCLCHL, SYSTEM.DEF.CLNTCONN and SOLARIS_TO_TANDEM:

```
$DATA0 ZMQSEXE 91> cnvclchl -m MT01

MQSeries client channel table being converted

Opening TANDEM v2.0 CLNTCONN table

Opening UNIX v2.0 CLNTCONN table AMQCLCHL for output

Writing UNIX v2.0 CLNTCONN table entry for SOLARIS_TO_TANDEM
Writing UNIX v2.0 CLNTCONN table entry for SYSTEM.DEF.CLNTCONN

Closing TANDEM v2.0 CLNTCONN table
Closing UNIX v2.0 CLNTCONN table
MQSeries client channel table conversion complete.
```



```

$DATA0 MQSDATA 111> cnvmsgs -m MT01 -q QDESC -v

*****/
*
* PROCESSING MESSAGES
*
*****/
Opened file QDESC
-- Connect to Queue Manager MT01
QDESC

-----
Queue - ANNE.VSE2.SDRC.LOCAL
Queue has 1 available messages
Queue has 0 delivered messages
Process queue? (y/n): y
Process deleted messages? (y/n): n
Opened file $DATA0.MQSDATA.QFILE

--- Opening queue ANNE.VSE2.SDRC.LOCAL

--- Closing queue ANNE.VSE2.SDRC.LOCAL

MESSAGE SUMMARY:
    total messages read from file: 1
        available messages: 1
        deleted messages: 0
    total messages expected: 1
        messages converted: 1
        messages failed: 0

-----
Queue - DEAD.QUEUE
Queue has 0 available messages
Queue has 0 delivered messages
Process queue? (y/n): n

-----
Queue - VM03.TQ.SDRC.0002
Queue has 0 available messages
Queue has 0 delivered messages
Process queue? (y/n): n

-- Disconnect from Queue Manager MT01
QDESC

-----

Message processing completed normally
$DATA0 MQSDATA 112>

```

Related commands

cnv1520 Convert V1.5.1 definitions to V2.2.0.1.

crtmqcvx (Data conversion)

Purpose

Use the **crtmqcvx** command to create a fragment of code that performs data conversion on data type structures. The command generates a C function that can be used in an exit to convert your C structures.

The command reads an input file containing a structure or structures to be converted. It then writes an output file containing a code fragment or fragments to convert those structures.

For further information about this command and how to use it, refer to the *MQSeries Application Programming Guide*.

Syntax

```
▶▶—crtmqcvx—SourceFile—TargetFile—◀◀
```

Required parameters

SourceFile

Specifies the input file containing the C structures to be converted.

TargetFile

Specifies the output file containing the code fragments generated to convert the structures.

Return codes

- 0** Command completed normally
- 10** Command completed with unexpected results
- 20** An error occurred during processing

Examples

The following example shows the results of using the data conversion command against a source C structure. The command issued is:

```
crtmqcvx source target
```

The input file, source looks like this:

```

/* This is a test C structure which can be converted by the */
/* crtmqcvx utility */

struct my_structure
{
    int    code;
    MQLONG value;
};

```

The output file, target, produced by the command is shown below. You can use these code fragments in your applications to convert data structures. However, if you do so, you should understand that the fragment uses macros supplied in the MQSeries header file MQSVMHTH in subvolume ZMQSLIB.

```

MQLONG Convertmy_structure(
    PMQBYTE *in_cursor,
    PMQBYTE *out_cursor,
    PMQBYTE in_lastbyte,
    PMQBYTE out_lastbyte,
    MQHCONN hConn,
    MQLONG  opts,
    MQLONG  MsgEncoding,
    MQLONG  ReqEncoding,
    MQLONG  MsgCCSID,
    MQLONG  ReqCCSID,
    MQLONG  CompCode,
    MQLONG  Reason)
{
    MQLONG ReturnCode = MQRC_NONE;

    ConvertLong(1); /* code */

    AlignLong();
    ConvertLong(1); /* value */

Fail:
    return(ReturnCode);
}

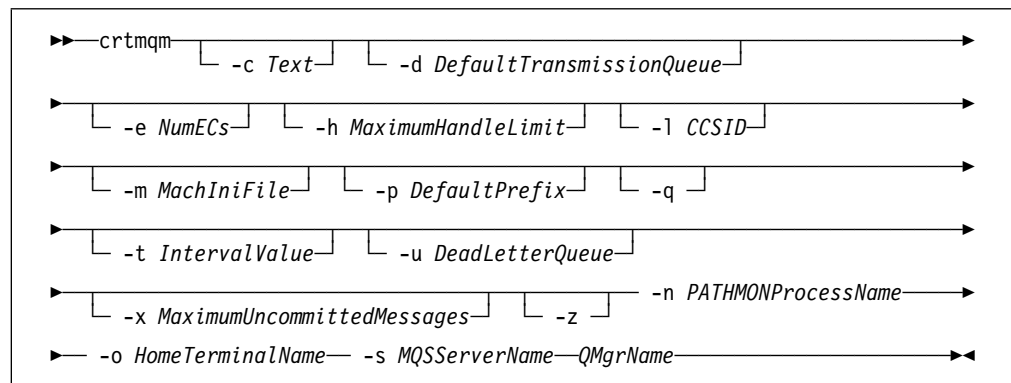
```

crtmqm (Create queue manager)

Purpose

Use the **crtmqm** command to create a local queue manager. Once a queue manager has been created, use the **strmqm** command to start it.

Syntax



Required parameters

-n *PATHMONProcessName*

The process name of the TS/MP PATHMON process for the queue manager. This process name must be unique in the system.

-o *HomeTerminalName*

Home terminal device name. (\$DDDD.#SS). For example, \$TRM1.#A.

-s *MQSServerName*

The process name to be given to the default MQSS Server for the queue manager. The process name must be unique in the system.

QMgrName

The name of the queue manager to be created. The name can contain up to 48 characters. This must be the last item in the command.

Optional parameters

-c *Text*

Some text (up to 64 characters) that describes this queue manager. The default is all blanks.

If special characters are required, the description must be enclosed in double quotation marks.

-d *DefaultTransmissionQueue*

The name of the local transmission queue that remote messages are placed on if a transmission queue is not explicitly defined for their destination. There is no default.

-e *NumECs*

The number of EC processes in the queue manager. The default is 1.

-h *MaximumHandleLimit*

The maximum number of handles that any one application can have open at the same time.

Specify a value in the range 1 through 999 999 999. The default value is 256.

-I *CCSID*

Qmgr CCSID. The default value is 819.

-m *MachIniFile*

Overrides the default MQSINI file location and that specified in the environment variable MQMACHINIFILE.

-p *DefaultPrefix*

The volume for the queue manager. Overrides the QMDefaultVolume entry in the MQSINI file.

-q Specifies that this queue manager is to be made the default queue manager. The new queue manager replaces any existing queue manager as the default.

If you accidentally use this flag and wish to revert to an existing queue manager as the default queue manager, you can edit the DefaultQueueManager stanza in the MQSeries configuration file. See Chapter 13, "Configuration files" on page 183 for information about configuration files.

-t *IntervalValue*

The trigger-time interval in milliseconds for all queues controlled by this queue manager. This value specifies the time after the receipt of a trigger-generating message when triggering is suspended. That is, if the arrival of a message on a queue causes a trigger message to be put on the initiation queue, any message arriving on the same queue within the specified interval does not generate another trigger message.

You can use the trigger time interval to ensure that your application is allowed sufficient time to deal with a trigger condition before it is alerted to deal with another on the same queue. You may wish to see all trigger events that happen; if so, set a low or zero value in this field.

Specify a value in the range 0 through 999 999 999. The default is 999 999 999 milliseconds, a time of more than 11 days. Allowing the default to be taken effectively means that triggering is disabled after the first trigger

message. However, triggering can be reenabled by an application servicing the queue using an alter queue command to reset the trigger attribute.

-u *DeadLetterQueue*

The name of the local queue that is to be used as the dead-letter (undelivered-message) queue. Messages are put on this queue if they cannot be routed to their correct destination.

By default, there is no dead-letter queue.

-x *MaximumUncommittedMessages*

Specifies the maximum number of uncommitted messages under any one syncpoint. That is, the sum of:

- The number of messages that can be retrieved from queues
- The number of messages that can be put on queues
- Any trigger messages generated within this unit of work

This limit does not apply to messages that are retrieved or put outside syncpoint control.

Specify a value in the range 1 through 10 000. The default value is 1000 uncommitted messages.

-z Suppresses error messages.

This flag is normally used within MQSeries to suppress unwanted error messages. As use of this flag could result in loss of information, you are recommended not to use it when entering commands on a command line.

Return codes

- 0** Queue manager created
- 8** Queue manager already exists
- 49** Queue manager stopping
- 69** Storage not available
- 70** Queue space not available
- 71** Unexpected error
- 72** Queue manager name error
- 111** Queue manager created. However, there was a problem processing the default queue manager definition in the product configuration file. The default queue manager specification may be incorrect.

Examples

1. This command creates a default queue manager named `Paint.queue.manager`, which is given a description of `Paint Shop`.

```
crtmqm -c "Paint Shop" -n $PANT -o $TRM1.#A -s $PNT1 Paint.queue.manager
```

2. In this example, another queue manager, `travel`, is created. The trigger interval is defined as 5000 milliseconds (or 5 seconds) and its dead-letter queue is specified as `SYSTEM.DEAD.LETTER.QUEUE`.

```
crtmqm -t 5000 -u SYSTEM.DEAD.LETTER.QUEUE -n $TRAV -o $TRM1.#A -s $TRV1 travel
```

Once a trigger event is generated, further trigger events are disabled for five seconds.

Related commands

strmqm	Start queue manager
endmqm	End queue manager
dltmqm	Delete queue manager

dltmqm (Delete queue manager)

Purpose

Use the **dltmqm** command to delete a specified queue manager. All objects associated with this queue manager are also deleted. Before you can delete a queue manager you must end it using the **endmqm** command.

Syntax

```
▶▶ dltmqm [-z] QMgrName ▶▶
```

Required parameters

QMGrName

Specifies the name of the queue manager to be deleted.

Optional parameters

-z Suppresses error messages.

Return codes

- 0** Queue manager deleted
- 5** Queue manager running
- 16** Queue manager does not exist
- 69** Storage not available
- 71** Unexpected error
- 72** Queue manager name error
- 112** Queue manager deleted. However, there was a problem processing the default queue manager definition in the product configuration file. The default queue manager specification may be incorrect.

Examples

1. The following command deletes the queue manager `saturn.queue.manager`.

```
dltmqm saturn.queue.manager
```

2. The following command deletes the queue manager `travel` and also suppresses any messages caused by the command.

```
dltmqm -z travel
```

Related commands

crtmqm Create queue manager

strmqm Start queue manager

endmqm End queue manager

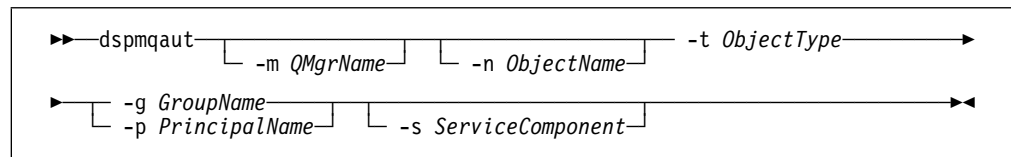
dspmqaout (Display authority)

Purpose

Use the **dspmqaout** command to display the current authorizations to a specified object. Only one group may be specified.

If a user ID is a member of more than one group, examine the authorizations of each group to determine all the authorizations that apply to the user ID.

Syntax



Required parameters

-g *GroupName*

Specifies the name of the user group on which the inquiry is to be made. You can specify only **one** name, which must be the name of an existing user group. You must specify either **-g** *GroupName* or **-p** *PrincipalName*.

-p *PrincipalName*

Specifies the name of the principal for which the authorizations to the specified object are to be displayed. You must specify either **-g** *GroupName* or **-p** *PrincipalName*.

-t *ObjectType*

Specifies the type of object on which the inquiry is to be made. Possible values are:

queue or q	A queue or queues matching the object type parameter
qmgr	A queue manager object
process or prcs	A process

Optional parameters

-m *QMgrName*

Specifies the name of the queue manager on which the inquiry is to be made.

-n *ObjectName*

Specifies the name of the object on which the inquiry is to be made.

This is a required parameter **unless** it is the queue manager itself.

You must specify the name of a queue manager, queue, or process definition.

-s *ServiceComponent*

This parameter applies only if you are using installable authorization services, otherwise it is ignored.

If installable authorization services are supported, this parameter specifies the name of the authorization service to which the authorizations apply. This

parameter is optional; if it is not specified, the authorization update is made to the first installable component for the service.

Returned parameters

This command returns an authorization list, which can contain none, one, or more authorization parameters. Each authorization parameter returned means that any user ID in the specified group has the authority to perform the operation defined by that parameter.

Table 11 shows the authorities that can be given to the different object types.

Authority	Queue	Process	Qmgr
all	√	√	√
alladm	√	√	√
allmqi	√	√	√
altusr			√
browse	√		
chg	√	√	√
chgaut	√	√	√
clr	√		
connect			√
crt	√	√	√
dlt	√	√	√
dsp	√	√	√
get	√		
inq	√	√	√
passall	√		
passid	√		
put	√		
set	√	√	√
setall	√		√
setid	√		√

The following list defines the authorizations associated with each parameter:

- all** Use all operations relevant to the object.
- alladm** Perform all administration operations relevant to the object.
- allmqi** Use all MQI calls relevant to the object.
- altusr** Specify an alternate user ID on an MQI call.
- browse** Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.

chg	Change the attributes of the specified object, using the appropriate command set.
chgaut	Specify authorizations for other groups of users on the object, using the setmqaut command.
clr	Clear a queue (PCF command Clear queue only).
connect	Connect the application to the specified queue manager by issuing an MQCONN call.
crt	Create objects of the specified type, using the appropriate command set.
dlt	Delete the specified object, using the appropriate command set.
dsp	Display the attributes of the specified object, using the appropriate command set.
get	Retrieve a message from a queue by issuing an MQGET call.
inq	Make an inquiry on a specific queue by issuing an MQINQ call.
passall	Pass all context.
passid	Pass the identity context.
put	Put a message on a specific queue by issuing an MQPUT call.
set	Set attributes on a queue from the MQI by issuing an MQSET call.
setall	Set all context on a queue.
setid	Set the identity context on a queue.

The authorizations for administration operations, where supported, apply to these command sets:

- Control commands
- MQSC commands
- PCF commands

Return codes

0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
69	Storage not available
71	Unexpected error
72	Queue manager name error
133	Unknown object name
145	Unexpected object name
146	Object name missing
147	Object type missing
148	Invalid object type
149	Entity name missing

Examples

The following example shows a command to display the authorizations on queue manager saturn.queue.manager associated with user group staff:

```
dspmqaout -m saturn.queue.manager -t qmgr -g staff
```

The results from this command are:

```
Entity staff has the following authorizations for object :  
  get  
  browse  
  put  
  inq  
  set  
  connect  
  altusr  
  passid  
  passall  
  setid
```

Related commands

setmqaut Set or reset authority

dspmqcsv (Display command server)

Purpose

Use the **dspmqcsv** command to display the status of the command server for the specified queue manager.

The status can be one of the following:

- Starting
- Running
- Running with SYSTEM.ADMIN.COMMAND.QUEUE not enabled for gets
- Ending
- Stopped

Syntax

```
▶▶—dspmqcsv—QMgrName—————▶▶
```

Required parameters

QMgrName

Specifies the name of the local queue manager for which the command server status is being requested.

Return codes

- 0** Command completed normally
- 10** Command completed with unexpected results
- 20** An error occurred during processing

Examples

The following command displays the status of the command server associated with `venus.q.mgr`:

```
dspmqcsv venus.q.mgr
```

Related commands

- strmqcsv** Start a command server
- endmqcsv** End a command server

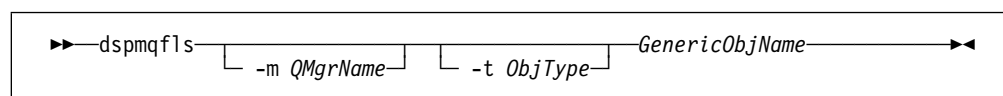
dspmqls (Display MQSeries file attributes)

Purpose

Use the **dspmqls** command to display the real file system name for all MQSeries objects that match a specified criterion. You can use this command to identify the files associated with a particular MQSeries object. This is useful for backing up specific objects. See “Volume structure” on page 70 for further information about name transformation.

You can also use the **dspmqls** command to display the current settings for the recalculation, update, supported retrieval criteria, and MQSS Server process-name options of an object.

Syntax



Required parameters

GenericObjName

Specifies the name of the MQSeries object. The name is a string with no flag and is a required parameter. If the name is omitted an error is returned.

This parameter supports a wild card character * at the end of the string.

Optional parameters

-m *QMgrName*

Specifies the name of the queue manager for which files are to be examined. If this parameter is omitted, the command operates on the default queue manager.

-t *ObjType*

Specifies the MQSeries object type. The following list shows the valid object types. The abbreviated name is shown first followed by the full name.

* or all	All object types; this is the default
q or queue	A queue or queues matching the object name parameter
ql or qlocal	A local queue
qa or qalias	An alias queue
qr or qremote	A remote queue
qm or qmodel	A model queue
qmgr	A queue manager object
prcs or process	A process

Note: The **dspmqls** command displays the names of all the files for the queue.

Return codes

- 0 Command completed normally
- 10 Command completed but not entirely as expected
- 20 An error occurred during processing

Examples

- The following command displays the details of all objects with names beginning SYSTEM.ADMIN that are defined on the default queue manager.

```
dspmqls SYSTEM.ADMIN*
```

- The following command displays file details for all processes with names beginning PROC defined on queue manager RADIUS.

```
dspmqls -m RADIUS -t prcs PROC*
```

- The following command displays file information for MY.LOCAL.QUEUE:

```
dspmqls -m MT02 -t q MY.LOCAL.QUEUE
```

```
MQSeries Display MQ Files
```

```
CONNECTING.
```

```
QLOCAL MY.LOCAL.QUEUE
  $DATA0.MT02M.QMYXLOCA
  $DATA0.MT02M.OMYXLOCA
  $DATA0.MT02D.TMYXLOCA
```

```
MQSS Server      : DEFAULT
Recalc Option    : FIRSTOPEN
Update Option    : LASTCLOSE
Get Option       : MCB
```

- QMYXLOCA is the queue file, OMYXLOCA is the overflow file for the queue, and TMYXLOCA is the touch file in the queue manager's data directory.
 - This queue has Get Option MCB, meaning it supports message retrieval by *MsgID*, *CorrelId*, or both.
 - Recalc option FIRSTOPEN is the default, and means that the CURDEPTH count for the queue will be updated on the FIRST MQOPEN.
 - Update Option LASTCLOSE means the object catalog will be updated on the last MQCLOSE.
- The following example shows two **altmqfls** commands, which move the queue files to a new volume (\$data4) and limit message-retrieval options to MQGET with *MsgId*. The **dspmqls** command displays the results of the **altmqfls** commands.

```
|  
| altmqfls -m MT02 -t ql -v $data4 MY.LOCAL.QUEUE  
| altmqfls -m MT02 -t ql -g M MY.LOCAL.QUEUE  
  
| dspmqls -m MT02 -t q MY.LOCAL.QUEUE  
  
| MQSeries Display MQ Files  
  
| CONNECTING.  
| QLOCAL MY.LOCAL.QUEUE  
| $data4.MT02M.QMYXLOCA  
| $data4.MT02M.OMYXLOCA  
| $DATA0.MT02D.TMYXLOCA  
  
| MQSS Server : DEFAULT  
| Recalc Option : FIRSTOPEN  
| Update Option : LASTCLOSE  
| Get Option : M  
|
```

Related commands

altmqfls Alter queue volume

dspmqtrc (Display MQSeries formatted trace output)

Purpose

Use the **dspmqtrc** command to display MQSeries formatted trace output. For more information about using MQSeries trace, see “Using MQSeries trace” on page 210.

Syntax

```
▶▶—dspmqtrc— -t FormatTemplate —InputFileName◀◀
```

Required parameters

InputFileName

Specifies the name of the file containing the unformatted trace. For example \$DATA.MQTRACE.AMQ12345..

-t *FormatTemplate*

Specifies the name of the template file containing details of how to display the trace. A trace-format template file, AMQTRC, is provided in subvolume ZMQSSMPL.

Related commands

endmqtrc End MQSeries trace
strmqtrc Start MQSeries trace

dspmqsqr (Display MQSeries user information)

Purpose

Use the **dspmqsqr** command to display information about a specified principal, or all principals for the queue manager.

Syntax

```
▶▶ dspmqsqr -m QMgrName [-p PrincipalName] ▶▶
```

Description

You can use this command to:

- Display all principals, or a particular principal, defined for a queue manager.
- Display the Tandem NSK Administrative and SAFEGUARD file-sharing groups corresponding to the Tandem NSK user ID associated with each principal.

Required parameters

-m *QMgrName*

Is the name of the queue manager to which the principals belong.

Optional parameters

-p *PrincipalName*

Is the name of the principal to be displayed.

Return codes

0 Successful operation
36 Invalid arguments supplied
69 Storage not available
71 Unexpected error

dspmqsqr

Examples

1. This example shows **dspmqsqr** for a newly created queue manager:

```
dspmqsqr -m MT02
```

Principal	Userid	Username	Alias	GroupName	GroupType
	0.1				
NOBODY	0.0				
mqm	20.255	MQM.MANAGER	n	MQM	a

The principal database contains the principal mqm, which maps to the user name of the user who created the queue manager.

2. This example shows output from **dspmqsqr** after additional principals have been added with **altmqsr**:

```
dspmqsqr -m MT02
```

Principal	Userid	Username	Alias	GroupName	GroupType
	0.1				
NOBODY	0.0				
mqm	20.255	MQM.MANAGER	n	MQM	a
mquser1	50.3	MQTEST.FRED	n	MQTEST	a
				MQM	s
mquser2	1.1	GROUP.USER01	n	GROUP	a

Principal mquser1, which maps to Tandem user ID MQTEST.FRED, has been added. FRED is a member of group MQTEST and a member of group MQM using SAFEGUARD aliasing.

Principal mquser2 maps to Tandem user ID GROUP.USER01.

Related commands

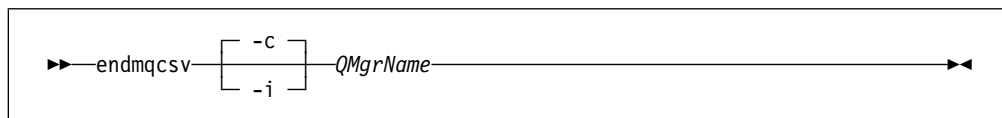
altmqsr Alter MQSeries user information

endmqcsv (End command server)

Purpose

Use the **endmqcsv** command to stop the command server on the specified queue manager.

Syntax



Required parameters

QMgrName

Specifies the name of the queue manager for which the command server is to be ended.

Optional parameters

- c Specifies that the command server is to be stopped in a controlled manner. The command server is allowed to complete the processing of any command message that it has already started. No new message is read from the command queue.

This is the default.
- i Specifies that the command server is to be stopped immediately. Actions associated with a command message currently being processed may not be completed.

Return codes

- 0** Command completed normally
- 10** Command completed with unexpected results
- 20** An error occurred during processing

Examples

1. The following command stops the command server on queue manager saturn.queue.manager:

```
endmqcsv -c saturn.queue.manager
```

The command server can complete processing any command it has already started before it stops. Any new commands received remain unprocessed in the command queue until the command server is restarted.

2. The following command stops the command server on queue manager pluto immediately:

```
endmqcsv -i pluto
```

Related commands

- | | |
|-----------------|--|
| strmqcsv | Start a command server |
| dspmqcsv | Display the status of a command server |

endmqm (End queue manager)

Purpose

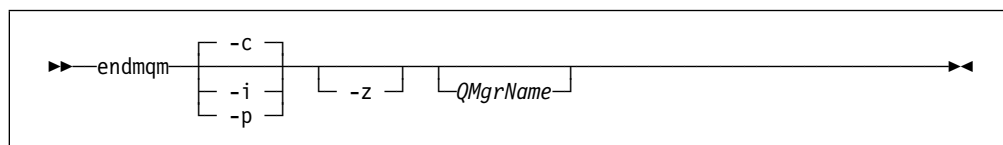
Use the **endmqm** command to end (stop) a specified local queue manager. This command stops a queue manager in one of three modes:

- Normal or quiesced shutdown
- Immediate shutdown
- Preemptive shutdown

The attributes of the queue manager and the objects associated with it are not affected. You can restart the queue manager using the **strmqm** (Start queue manager) command.

To delete a queue manager, you must stop it and then use the **dltmqm** (Delete queue manager) command.

Syntax



Optional parameters

QMgrName

Is the name of the message queue manager to be stopped. If no name is specified, the default queue manager is stopped.

- c Controlled (or quiesced) shutdown. The queue manager stops but only after all applications have disconnected. Any MQI calls currently being processed are completed. This is the default.
- i Immediate shutdown. The queue manager stops after it has completed all the MQI calls currently being processed. Any MQI requests issued after the command has been issued fail. Any incomplete units of work are rolled back when the queue manager is next started.
- p Preemptive shutdown.

Use this type of shutdown only in exceptional circumstances. For example, when a queue manager does not stop as a result of a normal **endmqm** command.

The queue manager stops without waiting for applications to disconnect or for MQI calls to complete. This can give unpredictable results for MQI applications. All processes in the queue manager that fail to stop are terminated 30 seconds after the command is issued.

- z Suppresses error messages on the command.

endmqm

Return codes

0	Queue manager ended
16	Queue manager does not exist
36	Invalid arguments
40	Queue manager not available
69	Storage not available
71	Unexpected error
72	Queue manager name error

Examples

The following examples show commands that end (stop) the specified queue managers.

1. This command ends the default queue manager in a controlled way. All applications currently connected are allowed to disconnect.

```
endmqm
```

2. This command ends the queue manager named `saturn.queue.manager` immediately. All current MQI calls complete, but no new ones are allowed.

```
endmqm -i saturn.queue.manager
```

Related commands

crtmqm	Create a queue manager
strmqm	Start a queue manager
dltmqm	Delete a queue manager

endmqtrc (End MQSeries trace)

Purpose

Use the **endmqtrc** command to end tracing for a specified queue manager.

For more information about using MQSeries trace, see “Using MQSeries trace” on page 210.

Syntax

```

▶▶—endmqtrc — -a _____
                |
                | —m QMgrName _____
                |
                | —e _____
                |
                |_____▶▶

```

Required parameters

-m *QMgrName*

Is the name of the queue manager for which tracing is to be ended.

A queue manager name can be specified on the same command as the **-e** flag.

-a If this flag is specified, all tracing is ended.

This flag **must** be specified alone.

Optional parameters

-e If this flag is specified, early tracing is ended on the named queue manager.

Return codes

AMQ5611

This message is issued if arguments that are not valid are supplied to the command.

Examples

This command ends tracing of data for a queue manager called QM1.

```
endmqtrc -m QM1
```

Related commands

dspmqtrc Display formatted trace output
strmqtrc Start MQSeries trace

instmqm

instmqm (Install MQSeries for Tandem NSK)

Purpose

Use the **instmqm** command to install MQSeries for Tandem NSK or update licenses.

Syntax

```
▶▶ instmqm [-i] ▶▶
```

Optional parameters

-i Invokes **instmqm** for license updates.

runmqchi (Run channel initiator)

Purpose

Use the **runmqchi** command to run a channel initiator process. For more information about the use of this command, refer to the *MQSeries Intercommunication* book.

Syntax

```

▶▶—runmqchi — [ -q InitiationQName ] [ -m QMgrName ] ▶▶

```

Optional parameters

-q *InitiationQName*

Specifies the name of the initiation queue to be processed by this channel initiator. If no value is specified, SYSTEM.CHANNEL.INITQ is used.

-m *QMgrName*

Specifies the name of the queue manager on which the initiation queue exists. If the name is omitted, the default queue manager is used.

Return codes

- 0** Command completed normally
- 10** Command completed with unexpected results
- 20** An error occurred during processing

If errors occur that result in return codes of either 10 or 20, you should review the queue manager error log that the channel is associated with for the error messages. You should also review the system error log, as problems that occur before the channel is associated with the queue manager are recorded there. For more information about error logs, see “Error logs” on page 207.

runmqchl (Run channel)

Purpose

Use the **runmqchl** command to start either a sender (SDR), requester (RQSTR), or fully qualified server channel. On MQSeries for Tandem NSK, **runmqchl** can also be used to start LU 6.2 responder processes for listening-type SNA channels that are not AUTOSTART(ENABLED) or that have just been defined.

The channel runs asynchronously. To stop the channel, issue the MQSC command STOP CHANNEL.

Syntax

```
▶▶—runmqchl— -c ChannelName — [ -m QMgrName ] ▶▶
```

Required parameters

-c *ChannelName*
Specifies the name of the channel to start.

Optional parameters

-m *QMgrName*
Specifies the name of the queue manager with which this channel is associated. If no name is specified, the default queue manager is used.

Return codes

- 0** Command completed normally
- 10** Command completed with unexpected results
- 20** An error occurred during processing

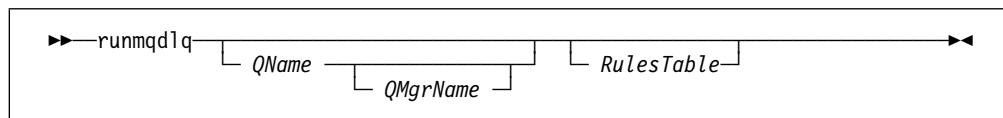
If return codes 10 or 20 are generated, review the error log of the associated queue manager for the error messages. You should also review the @SYSTEM error log because problems that occur before the channel is associated with the queue manager are recorded there.

runmqdlq (Run dead-letter queue handler)

Purpose

Use the **runmqdlq** command to start the dead-letter queue (DLQ) handler, a utility that processes messages on a dead-letter queue.

Syntax



Optional parameters

QName Is the name of the dead-letter queue to be processed.

If you specify a *QName* value, it overrides any INPUTQ value specified in a rules table. If no (nonblank) name is specified either on input to **runmqdlq** or in the rules table, the dead-letter queue associated with the queue manager named on the *QMgrName* parameter is processed.

QMgrName

Is the name of the queue manager that owns the queue to be processed.

If you specify a *QMgrName* value, it overrides any INPUTQM value specified in a rules table. If no (nonblank) name is specified either on input to **runmqdlq** or in the rules table, the queue is assumed to belong to the default queue manager.

RulesTable

Is the name of the file containing the rules table, which must contain at least one rule.

By default, the **runmqdlq** command takes its input from the standard IN file. When the command is processed, the results and a summary are put into a report that is sent to the standard OUT file. Alternatively, by redirecting the input from a file, you can apply a rules table to the specified queue.

If no rules table is specified on input to **runmqdlq**, rules and actions must be specified interactively.

In this case, the DLQ handler:

- Reads its input from the keyboard.
- Does not start to process the named queue until it receives an end_of_file (ctrl-Y) character.

The MQSC rules for comment lines and for joining lines also apply to the DLQ handler input parameters.

For more information about rules tables and how to construct them, see “DLQ handler rules table” on page 158.

runmqlsr (Run listener)

Purpose

The runmqlsr (Run listener) command runs a TCP/IP listener process.

Syntax

```

runmqlsr [-t tcp] [-p Port] [-m QMgrName]

```

Description

When run from a TACL prompt, **runmqlsr** does not allow the TACL (that is, it is run waited). The TACL prompt returns only if there is a failure or the listener stops. If the terminal (TACL) is stopped before **runmqlsr**, the listener is unable to access its home terminal or out file. Before **runmqlsr** is invoked, all PARAMs (such as MQEMSEVENTS) must be defined.

For these reasons, you are recommended to start and stop the listener from the queue manager's PATHWAY, which gives a greater degree of control.

Optional parameters

-p Port Port number for TCP/IP. If a value is not specified, the port number specified on a TCPListenerPort entry in the TCPConfig stanza in the QMINI file is used. The default value is 1414. If multiple listener ports are defined in QMINI, the next available port is used.

If none of the ports specified in QMINI is free, or the port specified on the **runmqlsr** command is not available, **runmqlsr** fails.

-m QMgrName
Specifies the name of the queue manager. If no name is specified, the command operates on the default queue manager.

-t tcp Identifies TCP/IP as the transmission protocol. This is the only valid value (and the default) in MQSeries for Tandem NSK.

Return codes

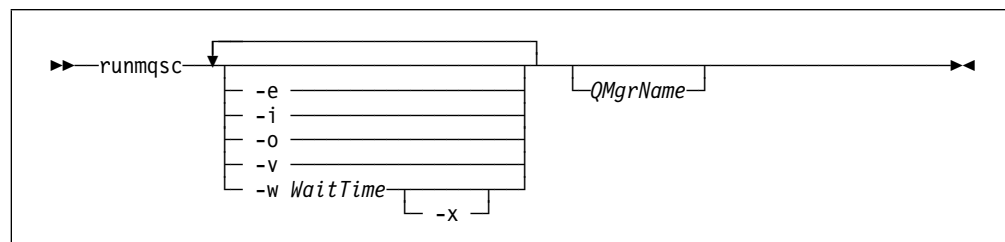
0 Command completed normally
10 Command completed with unexpected results
20 An error occurred during processing

runmqsc (Run MQSeries commands)

Purpose

Use the **runmqsc** command to issue MQSC commands to a queue manager. MQSC commands enable you to perform administration tasks, for example defining, altering, or deleting a local queue object. MQSC commands and their syntax are described in the *MQSeries Command Reference*.

Syntax



Description

You can invoke the **runmqsc** command in three modes:

- Verify mode** MQSC commands are verified but not actually run. An output report is generated indicating the success or failure of each command. This mode is only available on a local queue manager.
- Direct mode** MQSC commands are sent directly to a local queue manager.
- Indirect mode** MQSC commands are run on a remote queue manager. These commands are put on the command queue on a remote queue manager and are run in the order in which they were queued. Reports from the commands are returned to the local queue manager.

The **runmqsc** command takes its input from the standard IN file. When the commands are processed, the results and a summary are put into a report that is sent to the standard OUT file.

By taking the standard IN file from the keyboard, you can enter MQSC commands interactively.

By redirecting the input from a file you can run a sequence of frequently-used commands contained in the file. You can also redirect the output report to a file.

Note: To run this command, your user ID must belong to user group mqm.

Optional parameters

- e** Prevents source text for the MQSC commands from being copied into a report. This is useful when you enter commands interactively.
- i** Input file name
- o** Output file name
- v** Specifies verification mode; this verifies the specified commands without performing the actions. This mode is available locally only. The **-w** and **-x** flags are ignored if they are specified at the same time.

-w *WaitTime*

Specifies indirect mode, that is, the MQSC commands are to be run on another queue manager. You must have the required channel and transmission queues set up for this. See “Preparing channels and transmission queues for remote administration” on page 127 for more information.

WaitTime Specifies the time, in seconds, that **runmqsc** waits for replies. Any replies received after this are discarded, however, the MQSC commands are still run. Specify a time between 1 and 999 999 seconds.

Each command is sent as an Escape PCF to the command queue (SYSTEM.ADMIN.COMMAND.QUEUE) of the target queue manager.

The replies are received on queue SYSTEM.MQSC.REPLY.QUEUE and the outcome is added to the report. This can be defined as either a local queue or a model queue.

Indirect mode operation is performed through the default queue manager.

This flag is ignored if the **-v** flag is specified.

- x** Specifies that the target queue manager is running under MVS/ESA. This flag applies only in indirect mode. The **-w** flag must also be specified. In indirect mode, the MQSC commands are written in a form suitable for the MQSeries for MVS/ESA command queue.

QMgrName

Specifies the name of the target queue manager on which the MQSC commands are to be run. If omitted, the MQSC commands run on the default queue manager.

Return codes

- 00** MQSC command file processed successfully.
- 10** MQSC command file processed with errors-report contains reasons for failing commands.
- 20** Error-MQSC command file not run.

Examples

1. Enter this command at the TACL prompt:

```
runmqsc
```

Now you can enter MQSC commands directly. No queue manager name was specified, therefore the MQSC commands are processed on the default queue manager.

2. The following example shows how to specify that MQSC commands are verified only:

```
runmqsc -i $SYSTEM.CONFIG.MQSCIN -v BANK
```

This verifies the MQSC command file \$SYSTEM.CONFIG.MQSCIN. The queue manager name is BANK. The output is displayed in the current window.

3. This command runs an MQSC command file against the queue manager called BANK.

```
runmqsc -i MQSCFILE -o $TEST.MQ.MQSCOUT BANK
```

In this example, the output is directed to file \$TEST.MQ.MQSCOUT. The input file is MQSCFILE in the current subvolume.

runmqtrm (Start trigger monitor)

Purpose

Use the **runmqtrm** command to invoke a trigger monitor. For further information about using trigger monitors, refer to the *MQSeries Application Programming Guide*.

Syntax

```
▶▶—runmqtrm [ -m QMgrName ] [ -q InitiationQName ]▶▶
```

Optional parameters

-m *QMgrName*

Specifies the name of the queue manager on which the trigger monitor operates. If this parameter is omitted, the trigger monitor operates on the default queue manager.

-q *InitiationQName*

Specifies the name of the initiation queue to be processed. If this parameter is omitted, SYSTEM.DEFAULT.INITIATION.QUEUE is used.

Return codes

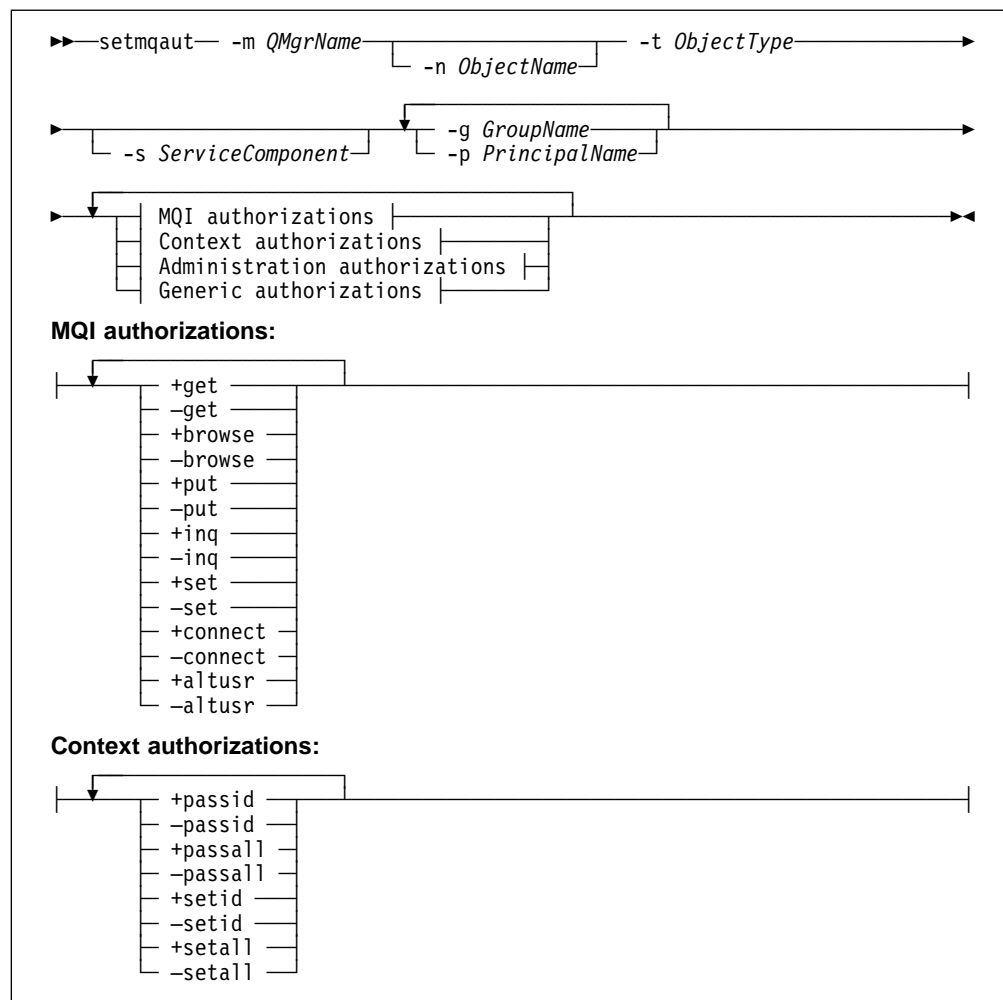
- 0** Not used. The trigger monitor is designed to run continuously and therefore not to end. Hence a value of 0 would not be seen. The value is reserved.
- 10** Trigger monitor interrupted by an error.
- 20** Error—trigger monitor not run.

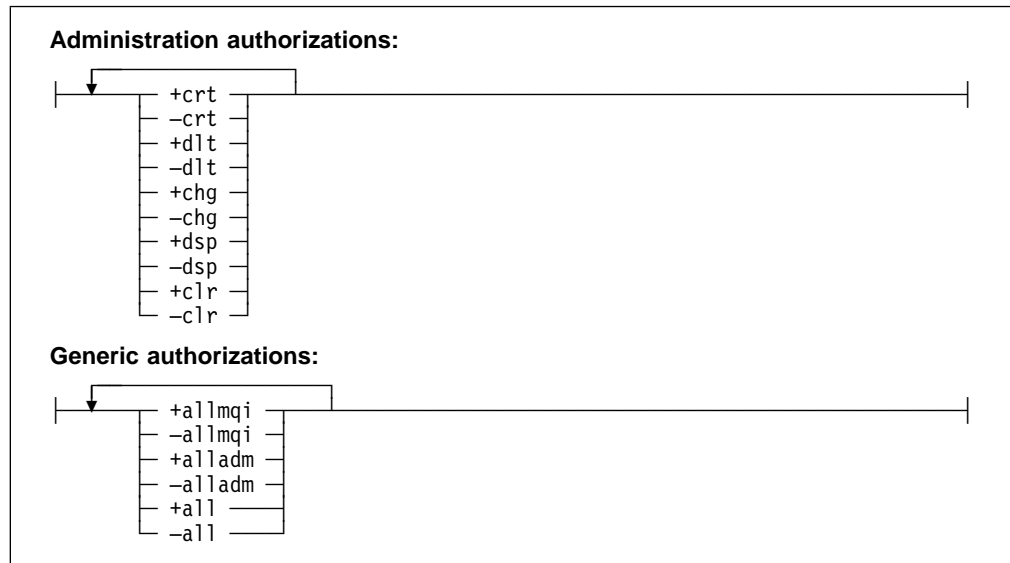
setmqaut (Set/reset authority)

Purpose

Use the **setmqaut** command to change the authorizations to an object or to a class of objects. Authorizations can be granted to, or revoked from, any number of principals or groups.

Syntax





Description

You can use this command both to *set* an authorization, that is, give a user group permission to perform an operation, and to *reset* an authorization, that is, remove the permission to perform an operation. You must specify the user groups to which the authorizations apply and also the queue manager, object type, and object name of the object. You can specify any number of groups in a single command.

The authorizations that can be given are categorized as follows:

- Authorizations for issuing MQI calls
- Authorizations for MQI context
- Authorizations for issuing commands for administration tasks
- Generic authorizations

Each authorization to be changed is specified in an authorization list as part of the command. Each item in the list is a string prefixed by '+' or '-'. For example, if you include +put in the authorization list, you are giving authority to issue MQPUT calls against a queue. Alternatively, if you include -put in the authorization list, you are removing the authorization to issue MQPUT calls.

Authorizations can be specified in any order provided that they do not clash. For example, specifying allmqi with set causes a clash.

You can specify as many groups or authorizations as you require in a single command.

If a user ID is a member of more than one group, the authorizations that apply are the union of the authorizations of each group to which that user ID belongs.

Required parameters

-g *GroupName*

Specifies the name of the user group whose authorizations are to be changed. You can specify more than one group name, but each name must be prefixed by the -g flag.

You must specify at least one principal or group.

-m *QMgrName*

Specifies the name of the queue manager of the object for which the authorizations are to be changed. The name can contain up to 48 characters.

-p *PrincipalName*

Specifies the name of the principal for which the authorizations are to be changed. You can specify more than one principal name, but each name must be prefixed by the -p flag.

You must specify at least one principal or group.

-t *ObjectType*

Specifies the type of object for which the authorizations are to be changed.

Possible values are:

- **q** or **queue**
- **prcs** or **process**
- **qmgr**

Optional parameters

-n *ObjectName*

Specifies the name of the object for which the authorizations are to be changed.

This is a required parameter *unless* it is the queue manager itself. You must specify the name of a queue manager, queue, or process, but must not use a generic name.

-s *ServiceComponent*

This parameter applies only if you are using installable authorization services, otherwise it is ignored.

If installable authorization services are supported, this parameter specifies the name of the authorization service to which the authorizations apply. This parameter is optional; if it is not specified, the authorization update is made to the first installable component for the service.

Authorizations

Specifies the authorizations to be given or removed. Each item in the list is prefixed by a '+' indicating that authority is to be given, or a '-', indicating that authorization is to be removed. For example, to give authority to issue an MQPUT call from the MQI, specify +put in the list. To remove authority to issue an MQPUT call, specify -put.

Table 12 on page 268 shows the authorities that can be given to the different object types.

Authority	Queue	Process	Qmgr
all	√	√	√
alladm	√	√	√
allmqi	√	√	√
altusr			√
browse	√		
chg	√	√	√
clr	√		
connect			√
crt	√	√	√
dlt	√	√	√
dsp	√	√	√
put	√		
inq	√	√	√
get	√		
passall	√		
passid	√		
set	√	√	√
setall	√		√
setid	√		√

Authorizations for MQI calls

- altusr** Use an alternate user ID in a message.
See the *MQSeries Application Programming Guide* for more information about alternate user IDs.
- browse** Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.
- connect** Connect the application to the specified queue manager by issuing an MQCONN call.
- get** Retrieve a message from a queue by issuing an MQGET call.
- inq** Make an inquiry on a specific queue by issuing an MQINQ call.
- put** Put a message on a specific queue by issuing an MQPUT call.
- set** Set attributes on a queue from the MQI by issuing an MQSET call.
- Note:** If you open a queue for multiple options, you have to be authorized for each of them.

Authorizations for context

- passall** Pass all context on the specified queue. All the context fields are copied from the original request.
- passid** Pass identity context on the specified queue. The identity context is the same as that of the request.
- setall** Set all context on the specified queue. This is used by special system utilities.
- setid** Set identity context on the specified queue. This is used by special system utilities.

Authorizations for commands

- chg** Change the attributes of the specified object.
- clr** Clear the specified queue (PCF Clear queue command only).
- crt** Create objects of the specified type.
- dlt** Delete the specified object.
- dsp** Display the attributes of the specified object.

Authorizations for generic operations

- all** Use all operations applicable to the object.
- alladm** Perform all administration operations applicable to the object.
- allmqi** Use all MQI calls applicable to the object.

Return codes

- 0** Successful operation
- 36** Invalid arguments supplied
- 40** Queue manager not available
- 49** Queue manager stopping
- 69** Storage not available
- 71** Unexpected error
- 72** Queue manager name error
- 133** Unknown object name
- 145** Unexpected object name
- 146** Object name missing
- 147** Object type missing
- 148** Invalid object type
- 149** Entity name missing
- 150** Authorization specification missing
- 151** Invalid authorization specification

Examples

1. This example shows a command that specifies that the object on which authorizations are being given is the queue orange.queue on queue manager saturn.queue.manager.

```
setmqaut -m saturn.queue.manager -n orange.queue -t queue -g tango +inq +alladm
```

The authorizations are being given to user group tango and the associated authorization list specifies that user group tango:

- Can issue MQINQ calls.
 - Has authority to perform all administration operations on that object.
2. In this example, the authorization list specifies that user group foxy:
 - Cannot issue any calls from the MQI to the specified queue.
 - Has authority to perform all administration operations on the specified queue.

```
setmqaut -m saturn.queue.manager -n orange.queue -t queue -g foxy -allmqi +alladm
```

Related commands

dspmqaut Display authority

strmqcsv (Start command server)

Purpose

Use the **strmqcsv** command to start the command server for the specified queue manager. This enables MQSeries to process commands sent to the command queue.

Syntax

```
▶▶—strmqcsv—QMgrName————▶▶
```

Required parameters

QMgrName

Specifies the name of the queue manager for which the command server is to be started.

Return codes

- 0** Command completed normally
- 10** Command completed with unexpected results
- 20** An error occurred during processing

Examples

The following command starts a command server for queue manager earth:

```
strmqcsv earth
```

Related commands

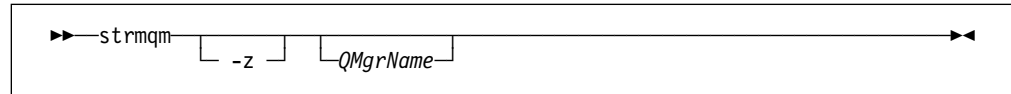
- endmqcsv** End a command server
- dspmqcsv** Display the status of a command server

strmqm (Start queue manager)

Purpose

Use the **strmqm** command to start a local queue manager.

Syntax



Optional parameters

QMgrName

Specifies the name of a local queue manager to be started. If omitted, the default queue manager is started.

-z Suppresses error messages.

This flag is used within MQSeries to suppress unwanted error messages. Because using this flag could result in loss of information, you should not use it when entering commands on a command line.

Return codes

- 0** Queue manager started
- 3** Queue manager being created
- 5** Queue manager running
- 16** Queue manager does not exist
- 49** Queue manager stopping
- 69** Storage not available
- 71** Unexpected error
- 72** Queue manager name error

Examples

The following command starts the queue manager account:

```
strmqm account
```

Related commands

- crtmqm** Create a queue manager
- dltmqm** Delete a queue manager
- endmqm** End a queue manager

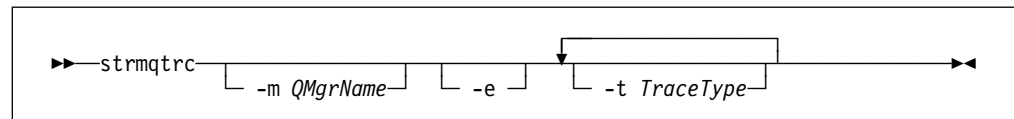
strmqtrc (Start MQSeries trace)

Purpose

Use the **strmqtrc** command to enable tracing. This command can be run whether tracing is enabled or not. If tracing is already enabled, the trace options in effect are modified to those specified on the latest invocation of the command.

For more information about using MQSeries trace, see “Using MQSeries trace” on page 210.

Syntax



Optional parameters

-m *QMgrName*

Is the name of the queue manager to be traced. If no name is specified, the default queue manager is used.

The specified queue manager does not have to be running or even to exist. Consequently, it is possible to trace the creation or startup of a queue manager.

A queue manager name can be specified on the same command as the **-e** flag. If more than one trace specification applies to a given entity being traced, the trace includes all of the specified options.

- e** If this flag is specified, early tracing is requested. This involves trace information being written, before the processes know to which MQSeries component they belong. Any process, belonging to any component of any queue manager, traces its early processing if this flag is specified. The default, if this flag is not specified, is not to perform early tracing.

-t *TraceType*

Defines which points during processing can be traced. One or more of the following options can be supplied:

all Output data for every trace point in the system. This is also the default if the **-t** flag is not specified.

api Output data for trace points associated with the MQI and major queue manager components.

comms

Output data for trace points associated with data flowing over communications networks.

csflows

Output data for trace points associated with processing flow in common services.

lqmflows

Output data for trace points associated with processing flow in the local queue manager.

remoteflows

Output data for trace points associated with processing flow in the communications component.

otherflows

Output data for trace points associated with processing flow in other components.

csdata

Output data for trace points associated with internal data buffers in common services.

lqmdata

Output data for trace points associated with internal data buffers in the local queue manager.

remotedata

Output data for trace points associated with internal data buffers in the communications component.

otherdata

Output data for trace points associated with internal data buffers in other components.

versiondata

Output data for trace points associated with the version of MQSeries running.

commentary

Output data for trace points associated with comments in the MQSeries components.

If this flag is omitted, all trace points are enabled and a full trace generated.

Note: If multiple trace types are supplied, each *must* have its own -t flag. Any number of -t flags can be specified, provided that each has a valid trace type associated with it.

It is not an error to specify the same trace type on multiple -t flags.

Return codes

AMQ7024

This message is issued if arguments that are not valid are supplied to the command.

AMQ8304

The maximum number of nine concurrent traces is already running.

Examples

This command enables tracing of data from common services and the local queue manager, for a queue manager called QM1.

```
strmqtrc -m QM1 -t csdata -t lqmdata
```


Related commands

- dspmqtrc** Display formatted trace output
- endmqtrc** End MQSeries trace

upgmqm (Upgrade V2.2 queue manager)

Purpose

This command upgrades a V2.2 queue manager for use with MQSeries for Tandem NSK V2.2.0.1. The utility invoked by **upgmqm** sends progress messages to the terminal from which it is invoked. When the utility completes, the named queue manager is ready for use with MQSeries for Tandem NSK V2.2.0.1. Queue manager attributes new in V2.2.0.1 are set to their default values. You can alter these in the usual way.

Note: If a V2.2 queue manager is not upgraded using **upgmqm**, no control commands will work for that queue manager. This includes **dltmqm**, which will fail with FFSTs trying to open the principal database. A queue manager from V2.2 no longer needed under V2.2.0.1 must be removed prior to the code upgrade, or upgraded using **upgmqm**, then removed.

Syntax

```
▶▶ upgmqm -m QMgrName -s DefaultMQSSServer -p SubvolumePath ▶▶
```

Required parameters

-m *QMgrName*

Specifies the name of the queue manager to which the **upgmqm** utility is to be applied.

-s *DefaultMQSSServer*

A unique process name for the default MQSS Server for the queue manager.

-p *SubvolumePath*

A subvolume path (\$VOL.SUBVOL) that the upgrade utility can use for working files. This subvolume **must** be on the same volume as the queue manager. Specify only the subvolume part of the path; do not specify the volume name. The **upgmqm** command checks that the subvolume does not already exist, before accepting the subvolume as valid.

Examples

This example upgrades V2.2 queue manager Myv2qm with a default MQSS Server name of \$MYSS and uses subvolume \$VOL.scratch for the working files (where \$VOL is the location of the queue manager):

```
upgmqm -m Myv2qm -s $MYSS -p scratch
```

Part 3. Appendixes

Appendix A. MQSeries for Tandem NSK at a glance

Program and part number

- 5697–A17 MQSeries for Tandem NSK, Version 2 Release 2.0.1, part number 83H8731.

Hardware requirements

Minimum hardware requirements are:

- Any of the Tandem NSK range of machines supported by Tandem NSK D3x, D4x, or G0x.
- Specific hardware in support of user-selected network transport protocols.

You are also recommended to have one or more mirrored data disks with specified space requirements for TMF audit space and the MQSeries database.

Software requirements

Minimum software requirements are:

- Tandem NSK D3x, D4x, G02, or later G0x operating systems, including TM/MP (TMF), ENSCRIBE, and EMS.
- TS/MP (PATHWAY) to match operating system.
- SCF for configuration, command, and control of TCP and SNA network transports.

For SNA connectivity:

- SNAX/APC and SNAX/XF or SNAX/APN to match operating system

or

- Insession ICE to match operating system

For TCP/IP connectivity:

- TCP/IP to match operating system.

Transaction logging is maintained with the Tandem TM/MP (TMF) product.

Security

MQSeries for Tandem NSK uses the security features of the NSK file system, which provide file-level access control to USER and GROUP for read, write, execute, and purge operations. SAFEGUARD is not required for the use of MQSeries for Tandem NSK; however, the product is compatible with a SAFEGUARD environment.

License management

All MQSeries resources are owned by a single user ID in group MQM. To administer MQSeries with either the SCOBOL menus or **runmqsc**, you must be logged in with a user ID assigned or linked to the MQM group.

Maintenance functions

MQSeries functions with:

- The Message Queue Management (MQM) facility using SCOBOL requester configuration screens in a PATHWAY environment.
- The **runmqsc** command-line interface.
- SCF utility for configuration, command and control functionality to maintain TCP/IP and SNA environments for Tandem network protocol offerings.
- ICE utilities provided with that product for control of ICE LU 6.2 interface.

Compatibility

The MQI for MQSeries for Tandem NSK, V2.2.0.1, is compatible with existing applications running V1.5.1.

Supported compilers

MQSeries for Tandem NSK V2.2.0.1 is built using the Common Runtime Environment (CRE) to link all objects. This method imposes the following requirements on users of versions of the MQI prior to Version 2:

1. All pre-D30 COBOL and C object code must be recompiled with the D30 (or later) compiler to integrate the CRE linkage.
2. All pre-D30 TAL object code must be recompiled with a D30 (or later) compiler and you must ensure that the TAL program is compliant with the special programming considerations specified in the *Common Run-time Environment Programmer's Guide*. More detailed information on each of these programming considerations is provided in the *TAL Programmer's Guide*.
3. For object code produced with native compilers on D40, a separate binding is provided.
4. C programs must use the WIDE memory model (32-bit integers).
5. COBOL programs must conform to the requirements of the CRE.
6. In TAL programs, all integers passed to the MQI functions must be 32 bits (or be cast to 32 bit with the \$INT32() macro).

License management

You must enter the system type and the number of CPUs to define the program entitlement. These parameters can be entered at installation time or at any subsequent time in the event of a license upgrade being purchased. At startup these values are checked against the physical Tandem machine configuration. If the license registration and program entitlement are insufficient, a warning message is issued.

Language selection

A supplied message text file is encoded in the 7-bit character set that is native to the Tandem NSK operating system. MQSeries for Tandem NSK lets the national language be specified when the product is installed. The message language defaults to U.S. English.

Message persistence

Persistent messages are defined as messages that, once committed, survive a system restart. Nonpersistent messages are messages that do not survive a system restart. To take advantage of the scalability provided by Tandem NSK platforms, all message queues are file based. Both persistent and nonpersistent messages are supported, though there is no performance advantage in using nonpersistent messages.

Internationalization

MQSeries for Tandem NSK lets the CCSID be specified when the queue manager instance is created. The queue manager CCSID defaults to 819. MQSeries for Tandem NSK supports character-set conversion into the configured CCSID of the queue manager. For information about the CCSIDs that can be specified for an MQSeries for Tandem NSK queue manager, including those that provide support for the euro character, see the *MQSeries Application Programming Reference* book.

|
|
|
|

Appendix B. PAK file installation examples

This appendix includes the following PAK file installation examples:

- Nonnative installation (Figure 39)
- Native installation (Figure 40 on page 287)
- UPGRADE installation (Figure 41 on page 290)

For information about the availability of the Tandem UNPAK utility, see the README file supplied with MQSeries for Tandem NSK.

An example PAK file installation (nonnative installation)

```

$DEV2 MQPAKS 291> unpak beta2201, $*.install.instmqm, map names $*.*.* to &
$DEV2 MQPAKS 291> &$dev2.mqpaks.*, listall,myid
UNPAK - File decompression program (Kari Kujansuu/Tandem Finland 1996)
Compression routines: 'zlib' by Jean-loup Gailly and Mark Adler.
(ftp://ftp.uu.net/pub/archiving/zip/zlib/)

Archive version: 1
File Mode RESTORE Program - T9074ACR (26MAY95)
Copyright Tandem Computers Incorporated 1981-1994
Drives: ($Z743)
System: \HAWK Operating System: G02 Tape Version: 3
Backup options: NO AUDITED, BLOCKSIZE 28, NO IGNORE, OPEN, PARTONLY OFF,
INDEXES IMPLICIT
Restore time: 15Dec98 20:21 Backup time: 7Dec98 16:18 Page: 1

Tape: 1 Code EOF Last modif Owner RWEPT Type Rec Block

$DEV2.MQPAKS
INSTMQM 100 425984 4Dec98 18:08 44,1 NNNN

Summary Information

Files restored = 1 Files not restored = 0

$DEV2 MQPAKS 293> instmqm

IBM MQSeries for Tandem NSK, Version 2.2.0.1
Installation and License update program.

```

Figure 39 (Part 1 of 4). An example PAK file installation (nonnative installation)

PAK file installation (nonnative)

| @(#) Licensed Materials - Property of IBM 83H8731,5697-A17 (C) Copyright IBM Co
| rp. 1993, 1997 All Rights Reserved US Government Users Restricted Rights - Use,
| duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

| Product installation selected...
| You may stop the installation by entering
| "quit" at any prompt.
| Where there is a default for a parameter, you may
| select it by pressing the Enter key on its own.

| Phase 1: Collection of license information

| License information
| Enter the system type that you are licensed for.
| The following system types are recognized:
| CLX/R
| CLX800
| K100
| K200
| CYCLONE
| K1000
| K2000
| S7000
| K10000
| K20000
| S70000
| There is no default value for this parameter.

| Please enter your selection: K2000

| Enter the number of CPUs that you are licensed for.
| The valid range for this parameter is 2 to 16.
| There is no default value for this parameter.

| Please enter the number: 4

| Will you be installing from tape or from an archive file?
| Enter TAPE or ARCHIVE.
| The default value for this parameter is "TAPE"

| Please enter the selection: ARCHIVE

| *Figure 39 (Part 2 of 4). An example PAK file installation (nonnative installation)*

| Enter the name of the archive file to be used for installation.
| The file name entered will be validated by opening it.
| If the file cannot be opened you will be given the
| opportunity to correct the name.
| Note: if you are using an archive file, you MUST have
| installed the "unpak" utility in a location that is
| in the default search path for this user.
| There is no default value for this parameter.

| Please enter the file name: BETA2201

| Enter the name of your spooler process.
| The name entered will be validated by opening it.
| If the spooler cannot be opened you will be given the
| opportunity to correct the name.
| The default value for this parameter is "\$S"

| Please enter the spooler name: \$S

| Select the type of installation to be performed.
| The following options are available:
| SCRATCH - a from scratch installation
| UPGRADE - an upgrade from the latest service
| level of MQSeries V2.2.0.0
| The default value for this parameter is "SCRATCH"

| Please enter the type of installation: SCRATCH

| Enter the volume that you will use for installation.
| Enter the volume name in the format "\$VVVVVVV".
| The default value for this parameter is "\$SYSTEM"

| Please enter the volume: \$DEV2

| Enter the default volume that you want Queue Managers to
| be created on.
| Note that the default Queue Manager volume may be changed
| at any time after installation by editing the MQSINI file.
| Enter the volume name in the format "\$VVVVVVV".
| The default value for this parameter is "\$DEV2"

| Please enter the volume: \$DEV2

| Enter the subvolume on \$DEV2 that you will use for executables.
| Enter the subvolume name in the format "VVVVVVVV".
| The default value for this parameter is "ZMQSEXE"

| *Figure 39 (Part 3 of 4). An example PAK file installation (nonnative installation)*

PAK file installation (nonnative)

```
| Please enter the subvolume:
|
| Select the language to be used for administration messages.
| The following languages are available:
| ENUS      - US English
| ESES      - Spanish
| The default value for this parameter is "ENUS"
|
| Please enter the language:
|
| You are installing on D3x-series NSK
| You do not have the option of installing native mode executables.
| License verified.
|
| You have selected the following parameters for installation:
| Archive file for installation:      BETA2201
| Spooler name:                      $$
| Volume for installation:           $DEV2
| Default Queue Manager volume:     $DEV2
| Subvolume for executables:        ZMQSEXE
| Language for messages:             ENUS
| You are installing accelerated (TNS) executables.
| This is not an upgrade to a prior V2.2.0.0 installation.
| Beginning to restore files to $DEV2.
| Ready to restore? (yes or quit):   yes
|
| Restoring product to $DEV2...
| Finished restoring files.
| If the summary information indicates a potential error,
| review the spooler jobs named #instmqm, and if necessary, repeat
| the installation.
| Securing files...
| Finished securing files.
| Creating MQSINI file...
| Finished creating MQSINI file.
| Creating message file...
| Finished creating message file.
| Installation complete.
| $DEV2 MQPAKS 294>
```

| *Figure 39 (Part 4 of 4). An example PAK file installation (nonnative installation)*

An example PAK file installation (native installation)

```
| $DATA00 MQPAKS 27> instmqm  
  
| IBM MQSeries for Tandem NSK, Version 2.2.0.1  
| Installation and License update program.  
  
| @(#) Licensed Materials - Property of IBM 83H8731,5697-A17 (C) Copyright  
| IBM Corp. 1993, 1997 All Rights Reserved US Government Users Restricted Rights -  
| Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.  
  
| Product installation selected...  
| You may stop the installation by entering  
| "quit" at any prompt.  
| Where there is a default for a parameter, you may  
| select it by pressing the Enter key on its own.  
  
| Phase 1: Collection of license information  
  
| License information  
| Enter the system type that you are licensed for.  
| The following system types are recognized:  
| CLX/R  
| CLX800  
| K100  
| K200  
| CYCLONE  
| K1000  
| K2000  
| S7000  
| K10000  
| K20000  
| S70000  
| There is no default value for this parameter.  
  
| Please enter your selection: S7000  
  
| Enter the number of CPUs that you are licensed for.  
| The valid range for this parameter is 2 to 16.  
| There is no default value for this parameter.  
  
| Please enter the number: 16  
  
| Will you be installing from tape or from an archive file?  
| Enter TAPE or ARCHIVE.  
| The default value for this parameter is "TAPE"
```

| *Figure 40 (Part 1 of 3). An example PAK file installation (native installation)*

PAK file installation (native)

```
| Please enter the selection:      ARCHIVE

| Enter the name of the archive file to be used for installation.
| The file name entered will be validated by opening it.
| If the file cannot be opened you will be given the
| opportunity to correct the name.
| Note: if you are using an archive file, you MUST have
| installed the "unpak" utility in a location that is
| in the default search path for this user.
| There is no default value for this parameter.

| Please enter the file name:      beta2201

| Enter the name of your spooler process.
| The name entered will be validated by opening it.
| If the spooler cannot be opened you will be given the
| opportunity to correct the name.
| The default value for this parameter is "$$"

| Please enter the spooler name:

| Select the type of installation to be performed.
| The following options are available:
| SCRATCH      - a from scratch installation
| UPGRADE      - an upgrade from the latest service
|                level of MQSeries V2.2.0.0
| The default value for this parameter is "SCRATCH"

| Please enter the type of installation:      scratch

| Enter the volume that you will use for installation.
| Enter the volume name in the format "$VVVVVVV".
| The default value for this parameter is "$SYSTEM"

| Please enter the volume:          $data00

| Enter the default volume that you want Queue Managers to
| be created on.
| Note that the default Queue Manager volume may be changed
| at any time after installation by editing the MQSINI file.
| Enter the volume name in the format "$VVVVVVV".
| The default value for this parameter is "$DATA00"

| Please enter the volume:

| Enter the subvolume on $DATA00 that you will use for executables.
| Enter the subvolume name in the format "VVVVVVVV".
| The default value for this parameter is "ZMQSEXE"
```

| *Figure 40 (Part 2 of 3). An example PAK file installation (native installation)*

```
| Please enter the subvolume:  
  
| Select the language to be used for administration messages.  
| The following languages are available:  
| ENUS      - US English  
| ESES      - Spanish  
| The default value for this parameter is "ENUS"  
  
| Please enter the language:  
  
| You are installing on G-series NSK  
| You have the option of installing native mode executables.  
| Select the type of executables to be installed.  
| The following options are available:  
| NATIVE    - install native mode (TNS/R) executables  
| NON-NATIVE - install accelerated (TNS) executables  
| The default value for this parameter is "NATIVE"  
  
| Please enter the type of executables:      native  
  
| License verified.  
  
| You have selected the following parameters for installation:  
| Archive file for installation:      BETA2201  
| Spooler name:                      $$  
| Volume for installation:           $DATA00  
| Default Queue Manager volume:     $DATA00  
| Subvolume for executables:        ZMQSEXE  
| Language for messages:            ENUS  
| You are installing native (TNS/R) executables.  
| This is not an upgrade to a prior V2.2.0.0 installation.  
| Beginning to restore files to $DATA00.  
| Ready to restore? (yes or quit):   yes  
  
| Restoring product to $DATA00...  
| Finished restoring files.  
| If the summary information indicates a potential error,  
| review the spooler jobs named #instmqm, and if necessary, repeat  
| the installation.  
| Relinking native executables...  
| Securing files...  
| Finished securing files.  
| Creating MQSINI file...  
| Finished creating MQSINI file.  
| Creating message file...  
| Finished creating message file.  
| Installation complete.
```

| *Figure 40 (Part 3 of 3). An example PAK file installation (native installation)*

An example PAK file installation (UPGRADE installation)

If you select an UPGRADE installation, a check is made for the latest service level of MQSeries for Tandem NSK V2.2. If this check is not satisfied the installation terminates.

An excerpt from an UPGRADE installation from an ARCHIVE is shown in Figure 41.

```
| Select the type of installation to be performed.
| The following options are available:
|   SCRATCH - a from scratch installation
|   UPGRADE - an upgrade from the latest service level of MQSeries V2.2.0.0
| The default value for this parameter is "SCRATCH"
|
| Please enter the type of installation:      UPGRADE
|
| Enter the volume that you installed MQSeries on.
| Enter the volume name in the format "$VVVVVVV".
| The default value for this parameter is "$SYSTEM"
|
| Please enter the volume:      $data0
|
| Enter the subvolume on $DATA0 containing the MQSeries executables.
| Enter the subvolume name in the format "VVVVVVVV".
| The default value for this parameter is "ZMQSEX"
|
| Please enter the subvolume:
|
| Verifying latest service level of V2.2.0.0 is present...
|
| Presence of PTF U456615 has been verified.
| Installation proceeding.
| Select the language to be used for administration messages.
| The following languages are available:
| ENUS      - US English
| ESES      - Spanish
| The default value for this parameter is "ENUS"
|
| Please enter the language:
|
| You are installing on D3x-series NSK
| You do not have the option of installing native mode executables.
| License information updated successfully
| License verified.
```

Figure 41 (Part 1 of 2). An example PAK file installation (UPGRADE install)

```
| You have selected the following parameters for installation:
| Archive file for installation:      $DATA0.MQBETA.BETA2201
| Spooler name:                      $$
| Volume for installation:           $DATA0
| Subvolume for executables:        ZMQSEXE
| Language for messages:             ENUS
| You are installing accelerated (TNS) executables.
| This is an upgrade to a prior V2.2.0.0 installation.
| Beginning to restore files to $DATA0.
| Ready to restore? (yes or quit):   yes

| Restoring product to $DATA0...
```

| *Figure 41 (Part 2 of 2). An example PAK file installation (UPGRADE install)*

PAK file installation (UPGRADE)

Appendix C. System defaults

The sample MQSC command file `amqscoma` contains definitions for the MQSeries for Tandem NSK default and system objects. The default object definitions contain a complete set of attributes for that object. When you create an object, its attributes are inherited from the default object, except the ones you explicitly specify. The system objects are required for the operation of a queue manager or channel. Table 13 lists the objects defined in `amqscoma`.

You should create these objects for each queue manager on a given node. To create these objects, see “Running the supplied MQSC command files” on page 105.

Table 13. Objects included in `amqscoma`

Object Name	Description
SYSTEM.DEFAULT.ALIAS.QUEUE	Default alias queue
SYSTEM.DEFAULT.LOCAL.QUEUE	Default local queue
SYSTEM.DEFAULT.MODEL.QUEUE	Default model queue.
SYSTEM.DEFAULT.REMOTE.QUEUE	Default remote queue.
SYSTEM.DEAD.LETTER.QUEUE	Sample dead-letter (undelivered-message) queue
SYSTEM.DEFAULT.PROCESS	Default process definition
SYSTEM.DEF.SENDER	Default sender channel
SYSTEM.DEF.SERVER	Default server channel
SYSTEM.DEF.RECEIVER	Default receiver channel
SYSTEM.DEF.REQUESTER	Default requester channel
SYSTEM.DEF.SVRCONN	Default server connection channel
SYSTEM.DEF.CLNTCONN	Default client connection channel
SYSTEM.CHANNEL.INITQ	Channel initiation queue
SYSTEM.CHANNEL.SYNCQ	Channel synchronization queue
SYSTEM.DEFAULT.INITIATION.QUEUE	Default initiation queue
SYSTEM.CICS.INITIATION.QUEUE	Default CICS® initiation queue
SYSTEM.ADMIN.COMMAND.QUEUE	Administration command queue. Used for remote MQSC commands, and PCF commands.
SYSTEM.MQSC.REPLY.QUEUE	MQSC reply-to-queue. This a model queue that creates a temporary dynamic queue for replies to remote MQSC commands.
SYSTEM.ADMIN.QMGR.EVENT	Event queue for queue manager events.
SYSTEM.ADMIN.PERFM.EVENT	Event queue for performance events.
SYSTEM.ADMIN.CHANNEL.EVENT	Event queue for channel events.

System defaults

Appendix D. Stopping and removing queue managers manually

If the normal methods for stopping and removing queue managers fail, you can resort to the more drastic methods described here.

Stopping a queue manager manually

The normal method of stopping queue managers, using the **endmqm** command, should work even in the event of failures within the queue manager. In exceptional circumstances, if this method of stopping a queue manager fails, use the following procedure to stop it manually:

1. Find the process IDs of the queue manager programs that are still running.
2. FUP LISTOPENS on the TRACEOPT file in the queue manager's data subvolume gives CPU, PIN of processes belonging to the queue manager.
3. End the queue manager processes that are still running. Use the **STOP** command, together with the process IDs discovered in the previous step.

End the processes in the following order:

- a. MQECBOSS – EC Boss
- b. MQEC – ECs
- c. Any other processes that are still running

Note: Manual ending of the queue manager may result in FFSTs being taken, and the production of FD files. This should not be regarded as a defect in the queue manager.

The queue manager should restart normally, even if it was ended by using the preceding method.

If you want to delete the queue manager after stopping it manually, use the **dltmqm** command as normal. If, for some reason, this command fails to delete the queue manager, the manual process detailed in “Removing queue managers manually” can be used.

Removing queue managers manually

To remove queue managers manually:

1. Ensure that there are no queue manager processes running for the queue manager you want to remove.
2. Edit the MQSINI file to remove the queue manager stanza and if necessary, modify the default queue manager stanza. Note the location of the queue manager files before deleting the stanza.
3. Delete all files in all subvolumes of the queue manager using the FUP PURGE command. For example, FUP PURGE \$VOL.QMSVOL*.*.

Removing queue managers manually

Appendix E. MQSC supported by MQSeries for Tandem NSK

This appendix lists the MQSeries commands (MQSC) supported by MQSeries for Tandem NSK. For information about the syntax of these commands, see the *MQSeries Command Reference* book.

ALTER CHANNEL	Change channel attributes.
ALTER PROCESS	Change process attributes.
ALTER QALIAS	Change attributes of an alias queue.
ALTER QLOCAL	Change attributes of a local queue.
ALTER QMGR	Change queue manager attributes.
ALTER QMODEL	Change attributes of a model queue.
ALTER QREMOTE	Change attributes of a local definition of a remote queue, a queue-manager alias, or a reply-to queue alias.
CLEAR QLOCAL	Clear messages from a local queue.
DEFINE CHANNEL	Create a channel definition.
DEFINE PROCESS	Create a process definition.
DEFINE QALIAS	Create an alias-queue definition.
DEFINE QLOCAL	Create a local-queue definition.
DEFINE QMODEL	Create a model-queue definition.
DEFINE QREMOTE	Create a local definition of a remote queue, a queue-manager alias, or a reply-to-queue alias.
DELETE CHANNEL	Delete a channel definition.
DELETE PROCESS	Delete a process definition.
DELETE QALIAS	Delete an alias-queue definition.
DELETE QLOCAL	Delete a local-queue definition.
DELETE QMODEL	Delete a model-queue definition.
DELETE QREMOTE	Delete a local definition of a remote queue.
DISPLAY CHANNEL	Display a channel definition.
DISPLAY CHSTATUS	Display the status of one or more channels.
DISPLAY PROCESS	Display a process definition.
DISPLAY QMGR	Display queue-manager attributes
DISPLAY QUEUE	Display queue attributes
PING CHANNEL	Test a channel.
PING QMGR	Test whether queue manager is responding to commands.
RESET CHANNEL	Reset the message sequence number for a channel.
RESOLVE CHANNEL	Resolve in-doubt messages on sender or server channel.

MQSeries commands

START CHANNEL	Start a channel or LU 6.2 responder process
STOP CHANNEL	Stop a channel

If you build MQSC commands into a script, there must be no more than 72 characters on each line.

Attributes of MQSC

This section provides information about MQSC attributes that is specific to MQSeries for Tandem NSK. See also Appendix J, "Setting up communications" on page 333.

APPLTYPE

The process attribute APPLTYPE can have the value NSK on these MQSC commands:

```
ALTER PROCESS
DISPLAY PROCESS
DEFINE PROCESS
```

AUTOSTART

The channel attribute AUTOSTART(ENABLED), which is required for SNA channels that are to accept incoming requests, indicates that the LU 6.2 responder process for the channel will be started at queue-manager startup. Triggering of SNA channels succeeds only if AUTOSTART(ENABLED) is specified and a responder process is already running for the channel.

AUTOSTART(ENABLED) can be specified on the DEFINE CHANNEL command for RCVR, RQSTR, SVR, and SVRCONN channels.

AUTOSTART(DISABLED) is the default value.

USERDATA for triggered programs

Data passed to the trigger monitor via the USERDATA attribute of MQSC DEFINE PROCESS or ALTER PROCESS must be in double quotation marks if it is a string containing spaces. For example, if this USERDATA

```
-o $DISK.VOLUME.PROGRAM -d 1
```

is to be passed to the trigger monitor, it must be specified on input to MQSC in double quotation marks, as follows:

```
'" -o $DISK.VOLUME.PROGRAM -d 1"'
```

If you display the process definition via MQSC, it appears as follows:

```
"-o $DISK.VOLUME.PROGRAM -d 1"
```

CLEAR QLOCAL considerations

The CLEAR QLOCAL command is unable to detect when uncommitted messages are being backed out from a queue (that TMF is in the process of abending a transaction), so might not purge all messages on the queue. Therefore, you are recommended to verify that the queue files are not open by a TMF-backout process before running CLEAR QLOCAL. You can use the **dspmqls** command to locate the queue files for this purpose.

CLEAR QLOCAL

Appendix F. Application Programming Reference

The following sections are specific to MQSeries for Tandem NSK, and should be used in conjunction with the *MQSeries Application Programming Reference* manual.

Elementary data types – TAL programming language

The elementary data types for the TAL programming language are:

Data Type	Representation
MQBYTE	STRING
MQBYTE24	BEGIN STRING BYTE [0:23];END
MQBYTE32	BEGIN STRING BYTE [0:31];END
MQCHAR	STRING
MQCHAR4	BEGIN STRING BYTE [0:3];END
MQCHAR8	BEGIN STRING BYTE [0:7]; END
MQCHAR12	BEGIN STRING BYTE [0:11];END
MQCHAR28	BEGIN STRING BYTE [0:27];END
MQCHAR32	BEGIN STRING BYTE [0:31];END
MQCHAR48	BEGIN STRING BYTE [0:47];END
MQCHAR64	BEGIN STRING BYTE [0:63];END
MQCHAR128	BEGIN STRING BYTE [0:127];END
MQCHAR256	BEGIN STRING BYTE [0:255];END
MQHCONN	INT(32)
MQHOBJ	INT(32)
MQLONG	INT(32)

Structure data types – TAL programming language

This section describes TAL declarations for the following MQSeries structure data types:

MQDLH	Dead letter header
MQGMO	Get message options
MQMD	Message descriptor
MQOD	Object descriptor
MQPMO	Put message options
MQTM	Trigger message
MQTMC2	Trigger message 2
MQXQH	Transmission queue header

MQDLH – Dead Letter Header

The MQDLH structure describes the information that is prefixed to the application message data of messages on the dead-letter (undelivered-message) queue.

MQDLH – TAL declaration

```
STRUCT      MQDLH^DEF (*);
BEGIN
STRUCT      STRUCID;
BEGIN STRING BYTE [0:3]; END;
INT(32)     VERSION;
INT(32)     REASON;
STRUCT      DESTQNAME;
BEGIN STRING BYTE [0:47]; END;
STRUCT      DESTQMGRNAME;
BEGIN STRING BYTE [0:47]; END;
INT(32)     ENCODING;
INT(32)     CODEDCHARSETID;
STRUCT      FORMAT;
BEGIN STRING BYTE [0:7]; END;
INT(32)     PUTAPPLTYPE;
STRUCT      PUTAPPLNAME;
BEGIN STRING BYTE [0:27]; END;
STRUCT      PUTDATE;
BEGIN STRING BYTE [0:7]; END;
STRUCT      PUTTIME;
BEGIN STRING BYTE [0:7]; END;
END;
```

MQGMO – Get Message Options

The MQGMO structure is an input/output parameter of the MGET call. Please note the following information about the MQGMO_SET_SIGNAL, MQGMO_WAIT, MQGMO_LOCK, MQGMO_UNLOCK, and MQGMO_NO_SYNCPOINT options in MQSeries for Tandem NSK:

MQGMO_SET_SIGNAL

The MQGMO_SET_SIGNAL option applies to MQSeries for Tandem NSK only. It cannot be specified in conjunction with the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT, MQGMO_BROWSE_MSG_UNDER_CURSOR, MQGMO_MSG_UNDER_CURSOR, and MQGMO_WAIT options.

For more information about MQGMO_SET_SIGNAL, see “Signal notification IPC message” on page 308.

MQGMO_WAIT

Only one nonbrowse MQGET call is activated for any given message. Nonbrowse MQGET calls are not given priority over browse MQGET calls waiting for the same message.

MQGMO_NO_SYNCPOINT

If MQPUT is issued outside a Tandem TMF transaction *without* the MQPMO_NO_SYNCPOINT option, the reason code MQRC_UNIT_OF_WORK_NOT_STARTED is returned.

MQGMO_LOCK, MQGMO_UNLOCK

The MQGMO_LOCK and MQGMO_UNLOCK options are not supported by MQSeries for Tandem NSK. If specified, they return the error MQRC_OPTIONS_ERROR.

MQGMO – TAL declaration

```

STRUCT      MQGMO^DEF (*);
BEGIN
    STRUCT      STRUCID;
BEGIN STRING BYTE [0:3]; END;
INT(32)      VERSION;
INT(32)      OPTIONS;
INT(32)      WAITINTERVAL;
INT(32)      SIGNAL1;
INT(32)      SIGNAL2;
STRUCT      RESOLVEDQNAME;
BEGIN STRING BYTE [0:47]; END;
END;

```

MQMD – Message Descriptor

The MQMD structure contains the control information that describes a message. Please note the following information about the *BackoutCount* field in MQSeries for Tandem NSK:

BackoutCount

A backout count is maintained for each message. This count is an estimate of the number of times, within a single queue manager association, a message has been returned to an application on consecutive nonbrowse MQGET calls, and subsequently backed out under TMF control. The backout count is not saved to disk, and is therefore not guaranteed to be accurate. The backout count is reset to zero when an implicit or explicit MQDISC is issued, when MQGET returns a different message, and at queue manager restart.

Structure data types (TAL)

MQMD – TAL declaration

```
STRUCT      MQMD^DEF (*);
  BEGIN
    STRUCT      STRUCID;
    BEGIN STRING BYTE [0:3]; END;
    INT(32)      VERSION;
    INT(32)      REPORTOPTIONS;
    INT(32)      MSGTYPE;
    INT(32)      EXPIRY;
    INT(32)      FEEDBACK;
    INT(32)      ENCODING;
    INT(32)      CODEDCHARSETID;
    STRUCT      FORMAT;
    BEGIN STRING BYTE [0:7]; END;
    INT(32)      PRIORITY;
    INT(32)      PERSISTENCE;
    STRUCT      MSGID;
    BEGIN STRING BYTE [0:23]; END;
    STRUCT      CORRELID;
    BEGIN STRING BYTE [0:23]; END;
    INT(32)      BACKOUTCOUNT;
    STRUCT      REPLYTOQ;
    BEGIN STRING BYTE [0:47]; END;
    STRUCT      REPLYTOQMGR;
    BEGIN STRING BYTE [0:47]; END;
    STRUCT      USERIDENTIFIER;
    BEGIN STRING BYTE [0:11]; END;
    STRUCT      ACCOUNTINGTOKEN;
    BEGIN STRING BYTE [0:31]; END;
    STRUCT      APPLIDENTITYDATA;
    BEGIN STRING BYTE [0:31]; END;
    INT(32)      PUTAPPLTYPE;
    STRUCT      PUTAPPLNAME;
    BEGIN STRING BYTE [0:27]; END;
    STRUCT      PUTDATE;
    BEGIN STRING BYTE [0:7]; END;
    STRUCT      PUTTIME;
    BEGIN STRING BYTE [0:7]; END;
    STRUCT      APPLORIGINDATA;
    BEGIN STRING BYTE [0:3]; END;
  END;
```

MQOD – Object Descriptor

The MQOD structure, which is an input/output parameter of the MQOPEN and MQPUT1 calls, is used to specify an object by name.

MQOD – TAL declaration

```
STRUCT      MQOD^DEF (*);BEGINSTRUCT      STRUCID;
BEGIN STRING BYTE [0:3]; END;INT(32)      VERSION;
INT(32)      OBJECTTYPE;STRUCT
  OBJECTNAME;
  BEGIN STRING BYTE [0:47]; END;STRUCT      OBJECTQMGRNAME;
  BEGIN STRING BYTE [0:47]; END;STRUCT      DYNAMICQNAME;
  BEGIN STRING BYTE [0:47]; END;STRUCT      ALTERNATEUSERID;
  BEGIN STRING BYTE [0:11]; END;
```

MQPMO – Put Message Options

The MQPMO structure is an input/output parameter of the MQPUT and MQPUT1 calls. Please note the following information about the MQPMO_NO_SYNCPOINT option in MQSeries for Tandem NSK:

MQPMO_NO_SYNCPOINT

If MQPUT is issued outside a Tandem TMF transaction *without* the MQPMO_NO_SYNCPOINT option, the reason code MQRC_UNIT_OF_WORK_NOT_STARTED is returned.

MQPMO – TAL declaration

```
STRUCT      MQPMO^DEF (*);
BEGIN
STRUCT      STRUCID;
BEGIN STRING BYTE [0:3]; END;
INT(32)     VERSION;
INT(32)     OPTIONS;
INT(32)     TIMEOUT;
INT(32)     CONTEXT;
INT(32)     KNOWNDESTCOUNT;
INT(32)     UNKNOWNDESTCOUNT;
INT(32)     INVALIDDESTCOUNT;
STRUCT      RESOLVEDQNAME;
BEGIN STRING BYTE [0:47]; END;
STRUCT      RESOLVEDQMGRNAME;
BEGIN STRING BYTE [0:47]; END;
END;
```

MQTM – Trigger Message

The MQTM structure describes the data in the trigger message that is sent by the queue manager to a trigger-monitor application when a trigger event occurs for a queue.

MQTM – TAL declaration

```
STRUCT      MQTM^DEF (*);
BEGIN
STRUCT      STRUCID;
BEGIN STRING BYTE [0:3]; END;
INT(32)     VERSION;
STRUCT      QNAME;
BEGIN STRING BYTE [0:47]; END;
STRUCT      PROCESSNAME;
BEGIN STRING BYTE [0:47]; END;
STRUCT      TRIGGERDATA;
BEGIN STRING BYTE [0:63]; END;
INT(32)     APPLTYPE;
STRUCT      APPLID;
BEGIN STRING BYTE [0:255]; END;
STRUCT      ENVDATA;
BEGIN STRING BYTE [0:127]; END;
STRUCT      USERDATA;
BEGIN STRING BYTE [0:127]; END;
END;
```

MQTMC2 – Trigger Message 2

Data passed by a trigger monitor application to a started application can be described by the MQTMC2 structure.

MQTMC2 – TAL declaration

```
STRUCT      MQTMC2^DEF (*);
BEGIN
STRUCT      STRUCID;
BEGIN STRING BYTE [0:3]; END;
STRUCT      VERSION;
BEGIN STRING BYTE [0:3]; END;
STRUCT      QNAME;
BEGIN STRING BYTE [0:47]; END;
STRUCT      PROCESSNAME;
BEGIN STRING BYTE [0:47]; END;
STRUCT      TRIGGERDATA;
BEGIN STRING BYTE [0:63]; END;
STRUCT      APPLTYPE;
BEGIN STRING BYTE [0:3]; END;
STRUCT      APPLID;
BEGIN STRING BYTE [0:255]; END;
STRUCT      ENVDATA;
BEGIN STRING BYTE [0:127]; END;
STRUCT      USERDATA;
BEGIN STRING BYTE [0:127]; END;
STRUCT      QMQRNAME;
BEGIN STRING BYTE [0:47]; END;
END;
```

MQXQH – Transmission Queue Header

The MQXQH structure describes the information that is prefixed to the application message data of messages on transmission queues.

MQXQH – TAL declaration

```
STRUCT      MQXQH^DEF (*);
BEGIN
STRUCT      STRUCID;
BEGIN STRING BYTE [0:3]; END;
INT(32)     VERSION;
STRUCT      REMOTEQNAME;
BEGIN STRING BYTE [0:47]; END;
STRUCT      REMOTEQMGRNAME;
BEGIN STRING BYTE [0:47]; END;
STRUCT      MSGDESC(MQMD^DEF);
END;
```

MQI calls – TAL programming language

This section describes TAL invocations of the following MQI calls:

MQBACK	Back out changes
MQCLOSE	Close object
MQCONN	Connect queue manager
MQDISC	Disconnect queue manager
MQGET	Get message
MQPUT	Put message
MQPUT1	Put one message
MQINQ	Inquire about object attributes
MQSET	Set object attributes
MQOPEN	Open object
MQCMIT	Commit changes
MQSYNC	Synchronize statistics updates

MQBACK – Back Out Changes

The MQBACK call indicates to the queue manager that all message gets and puts since the last syncpoint are to be backed out.

In MQSeries for Tandem NSK, MQBACK always returns a *CompCode* of MQCC_FAILED and a *Reason* of MQRC_ENVIRONMENT_ERROR. Transactions are managed externally through TM/MP.

MQBACK – TAL invocation

```
INT(32) .EXT Hconn;
INT(32) .EXT CC;
INT(32) .EXT Reason;

CALL MQBACK(HConn, CC, Reason);
```

MQCLOSE – Close Object

The MQCLOSE call, which is the inverse of the MQOPEN call, relinquishes access to an object.

MQCLOSE – TAL invocation

```
INT(32) .EXT HConn ;
INT(32) .EXT HObj;
INT(32) Options;
INT(32) .EXT CC;
INT(32) .EXT Reason;

CALL MQCLOSE(HConn, HObj, Options, CC, Reason);
```

MQCONN – Connect Queue Manager

The MQCONN call connects an application program to an MQSeries queue manager.

MQI calls (TAL)

MQCONN – TAL invocation

```
STRING .EXT InQMgr[0:47];  
INT(32) .EXT HConn ;  
INT(32) .EXT CC;  
INT(32) .EXT Reason;  
  
CALL MQCONN(InQMgr, HConn, CC, Reason);
```

MQDISC – Disconnect Queue Manager

The MQDISC call, which is the inverse of MQCONN, breaks the connection between the MQSeries queue manager and the application program.

MQDISC – TAL invocation

```
INT(32) .EXT HConn ;  
INT(32) .EXT CC;  
INT(32) .EXT Reason;  
  
CALL MQDISC(HConn, CC, Reason);
```

MQGET – Get Message

The MQGET call retrieves a message from a local queue that has been opened using the MQOPEN call.

Please note the following information about the operation of the MQGET call in MQSeries for Tandem NSK:

- The message retrieved by the MQGET call is deleted from the queue unless the MQGMO_BROWSE_FIRST option or the MQGMO_BROWSE_NEXT option is specified.
- If MQGET is issued outside a Tandem TMF transaction *without* the MQGMO_NO_SYNCPOINT option, the reason code MQRC_UNIT_OF_WORK_NOT_STARTED is returned.
- If the MQGMO_CONVERT option is specified for an MQGET call, and the message that is retrieved is not in one of the built-in formats (MQFMT_*), the message is passed to the data conversion exit function MQDATA CONVEXIT() for conversion. A single data conversion exit is provided by the product, because the Tandem NSK operating system does not support dynamic linking. The format name of the unconverted message, from the MQMD of the message, is passed to MQDATA CONVEXIT() in the *MsgDesc* parameter.

Signal notification IPC message

For backwards compatibility with MQSeries for Tandem NSK, Version 1.5.1, the signal mode of message-arrival notification is supported. This type of notification is selected by the MQGMO_SET_SIGNAL option in the options field of the Get Message Options structure. If MQGMO_SET_SIGNAL is specified, the following options are not valid:

- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_MSG_UNDER_CURSOR
- MQGMO_LOCK
- MQGMO_UNLOCK
- MQGMO_WAIT

If MQGMO_SET_SIGNAL is specified with any of these options, a *CompCode* of MQCC_FAILED and a *Reason* of MQRC_OPTIONS_ERROR are returned.

The effects of specifying MQGMO_SET_SIGNAL are as follows:

- If a message is available when MQGET is issued, it is returned immediately to the requesting application.
- If no message is available when MQGET is issued, a *CompCode* of MQCC_WARNING and a *Reason* of MQRC_SIGNAL_REQUEST_ACCEPTED are returned. When a message becomes available, an Inter-Process Communication (IPC) message is sent to the \$RECEIVE queue of the process that made the MQGET call.

The format of this IPC message is:

MsgCode (INT)

Identifies the message as a notification. The value is TRIGGER_RESPONSE.

AppTag (LONG)

Is the application tag provided in the *Signal1* field of MQGMO.

The *Signal1* field of MQGMO is significant only when the signal mode of message-arrival notification has been requested. It can be used by an application to associate the IPC notification message with a particular MQGET request.

Status (LONG)

Is the reason Code from MQGET. It can have the following values:

MQRC_NONE

A message satisfying the criteria specified in the MQGET call is available on the queue.

MQRC_NO_MSG_AVAILABLE

The time specified in the *WaitInterval* field has expired.

MQRC_CONNECTION_BROKEN

The queue manager has been stopped.

MQRC_GET_INHIBITED

An operator has inhibited the GET operation for the queue.

MQRC_Q_DELETED

The queue has been deleted.

MQRC_Q_MGR QUIESCING

The queue manager is quiescing, and the MQGET call was issued with the MQGMO_FAIL_IF QUIESCING option.

MQRC_Q_MGR_STOPPING

The queue manager is shutting down.

Only one signal-notification-mode MQGET call can be outstanding for any object handle. If an MQGET with signal notification is specified when there is already a signal-notification MQGET call outstanding for the same object handle, a *CompCode* of MQCC_FAILED and a *Reason* of MQRC_SIGNAL_OUSTANDING are returned.

If the signal notification indicates that a message is available (*Status* is MQRC_NONE), the message is not locked by the Queue Manager; therefore, it is also available to any other application that shares the queue. It is possible,

MQI calls (TAL)

therefore, that the message will not be available by the time the application issues an MQGET call to retrieve or browse the message. The signal notification IPC message is not part of any unit of work (that is, a Tandem TMF transaction), started by either the application or MQSeries.

If the application calls MQCLOSE for a queue with outstanding signal-notification MQGET operations initiated by that application, the outstanding signal notifications are cancelled. If an application calls MQDISC, all outstanding signal notifications initiated by the application are cancelled.

MQGET – TAL invocation

The TAL language invocation is:

```
INT(32) .EXT Hconn;
INT(32) .EXT Hobj;
STRUCT .EXT MsgDesc(MQMD^Def);
STRUCT .EXT GetMsgOpt(MQGM0^Def);
INT(32) .EXT BufferLen;
INT(32) .EXT Buffer[0:BUFFER^LEN];
INT(32) .EXT CC;
INT(32) .EXT Reason;
```

```
CALL MQGET(HConn, Hobj, MsgDesc, GetMsgOpt, BufferLen, Buffer,
           DataLen, CC, Reason);
```

MQPUT – Put Message

The MQPUT call puts a message on an open queue.

If MQPUT is issued outside a Tandem TMF transaction *without* the MQPMO_NO_SYNCPOINT option, the reason code MQRC_UNIT_OF_WORK_NOT_STARTED is returned.

MQPUT – TAL invocation

```
INT(32) .EXT HConn;
INT(32) .EXT Hobj;
STRUCT .EXT MsgDesc(MQMD^Def);
STRUCT .EXT PutMsgOpt(MQPMO^Def);
INT(32) .EXT BufferLen;
STRING .EXT Buffer[0:BUFFER^SIZE];
INT(32) .EXT CC;
INT(32) .EXT Reason;
```

```
CALL MQPUT(HConn, Hobj, MsgDesc, PutMsgOpt, BufferLen, Buffer,
           CC, Reason);
```

MQPUT1 – Put One Message

The MQPUT1 call puts one message on a queue; the queue need not be open.

If MQPUT1 is issued outside a Tandem TMF transaction without the MQPMO_NO_SYNCPOINT option, the reason code MQRC_UNIT_OF_WORK_NOT_STARTED is returned.

MQPUT1 – TAL invocation

```

INT(32) .EXT HConn ;
STRUCT .EXT ObjDesc(MQOD^Def);
STRUCT .EXT MsgDesc(MQMD^Def);
STRUCT .EXT PutMsgOpt(MQPMO^Def);
INT(32) .EXT BufferLen
STRING .EXT Buffer[0:BUFFER^SIZE]
INT(32) .EXT CC;
INT(32) .EXT Reason;

```

```

CALL MQPUT1(HConn, ObjDesc, MsgDesc, PutMsgOpt, BufferLen, Buffer,
            CC, Reason);

```

MQINQ – Inquire About Object Attributes

The MQINQ call returns the attributes of an object.

MQINQ – TAL invocation

```

INT(32) .EXT HConn ;
INT(32) .EXT HObj ;
INT(32) SelectorCount;
INT(32) .EXT Selectors[0:NUM^SELECTORS];
INT(32) IntAttrCount;
INT(32) .EXT IntAttrs[0:NUM^INT^ATTR];
INT(32) CharAttrLength;
STRING .EXT CharAttrs[0:LEN^CHAR^ATTR];
INT(32) .EXT CC;
INT(32) .EXT Reason;

```

```

PROC MQINQ(HConn, HObj, SelectorCount, Selectors, IntAttrCount,
           IntAttrs, CharAttrLength, CharAttrs, CC, Reason)

```

MQSET – Set Object Attributes

The MQSET call is used to change the attributes of a queue.

MQSET – TAL invocation

```

INT(32) .EXT HConn ;
INT(32) .EXT HObj;
INT(32) SelectorCount;
INT(32) .EXT Selectors[0:NUM^SELECTORS];
INT(32) IntAttrCount;
INT(32) .EXT IntAttrs[0:NUM^INT^ATTR];
INT(32) CharAttrLength;
STRING .EXT CharAttrs[0:LEN^CHAR^ATTR];
INT(32) .EXT CC;
INT(32) .EXT Reason;

```

```

CALL MQSET(HConn, HObj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
           CharAttrLength, CharAttrs, CC, Reason);

```

MQOPEN – Open Object

The MQOPEN call establishes access to an object.

MQOPEN – TAL invocation

```
INT(32) .EXT HConn;  
STRUCT .EXT ObjDesc(MQOD^Def);  
INT(32) Options; INT(32) .EXT Hobj;  
INT(32) .EXT CC;  
INT(32) .EXT Reason;  
  
CALL MQOPEN(HConn, ObjDesc, Options, HObj, CC, Reason);
```

MQCMIT – Commit Changes

The MQCMIT call indicates to the queue manager that the application has reached a syncpoint.

In MQSeries for Tandem NSK, the MQCMIT call always returns a *CompCode* of MQCC_FAILED and a *Reason* of MQRC_ENVIRONMENT_ERROR. Transactions are managed externally through TM/MP.

MQCMIT – TAL invocation

```
INT(32) .EXT Hconn;  
INT(32) .EXT CC;  
INT(32) .EXT Reason;
```

MQSYNC – Synchronize statistics updates

The MQSYNC call is included in MQSeries for Tandem NSK for backwards compatibility with MQSeries for Tandem NSK, Version 1.5.1. The call returns a *CompCode* of MQCC_OK and a *Reason* of MQRC_NONE, but performs no function.

MQSYNC – C invocation

```
transaction_id_def TransID;  
int CommitAbort;  
MQLONG CompCode;  
MQLONG Reason;  
  
MQSYNC(&TransID, CommitAbort, &CompCode, &Reason);
```

MQSYNC – COBOL invocation

```
01 TRANSID NATIVE-4.  
01 COMMITABORT NATIVE-4.  
01 COMPCODE NATIVE-4.  
01 REASON NATIVE-4.  
  
CALL 'MQSYNC' USING TRANSID COMMITABORT.
```

MQSYNC – TAL invocation

```
STRING .EXT TransID;  
INT CommitAbort;  
INT(32) .EXT CC;  
INT(32) .EXT Reason;  
  
CALL MQSYNC(TransID, CommitAbort, CC, Reason);
```

Attributes of MQSeries objects

In MQSeries for Tandem NSK, the attributes of all objects are as described in the *MQSeries Application Programming Reference* manual, with the following exceptions and additions.

Attributes of local queues

The *HardenGetBackout* attribute is ignored in MQSeries for Tandem NSK because the backout count is not saved to disk. To maintain compatibility with other MQSeries Version 2 products, the MQIA_HARDEN_GET_BACKOUT selector can be specified on the MQINQ call.

Attributes of process definitions

The *ApplType* attribute of a process definition can have the value MQAT_NSK.

Attributes of queue managers

- The value of the *Platform* attribute is MQPL_NSK.

The associated constant value is:

```
MQPL_NSK          12          X'0000000C'
```

- MaxMsgLength* is 4 MB.
- CommandLevel* is MQCMDL_LEVEL_221.
- SyncPoint* is MQSP_AVAILABLE.
- The value of *CodedCharSetId* is as specified when the queue manager instance was created.

Reason codes

In MQSeries for Tandem NSK, reason codes (MQRC_) are as described in the *MQSeries Application Programming Reference* manual, with the following additions:

Constant	Decimal	Hexadecimal
MQRC_SYNCPOINT_LIMIT_REACHED	2024	X'000007E8'
MQRC_MSG_ID_ERROR	2206	X'0000089E'
MQRC_CORREL_ID_ERROR	2207	X'0000089F'
MQRC_UNIT_OF_WORK_NOT_STARTED	2232	X'000008B8'
MQRC_UOW_CANCELED	2297	X'000008F9'

MQRC_MSG_ID_ERROR and MQRC_CORREL_ID_ERROR are possible return codes from MQGET only. These codes inform the application that a requested retrieval mode is not supported, having been removed using **altmqfls**. For more information, see “altmqfls (Alter queue file attributes)” on page 220.

Reason codes

Appendix G. Building and running applications

The sample programs and the sample compilation and binding scripts, provided in subvolume ZMQSSMPL, illustrate the main features of the MQI in MQSeries for Tandem NSK V2.2.0.1, and demonstrate how to compile and bind an application.

Writing applications

This section describes some minor differences between the standard Version 2 MQI interface, as documented in the *MQSeries Application Programming Reference* book, and the MQI interface for MQSeries for Tandem NSK V2.2.0.1.

Unit of work (transaction) management

Transaction management is performed under the control of Tandem's TM/MP product, rather than by MQSeries itself.

The effects of this difference are:

- The default SYNCPOINT option for the MQPUT and MQGET calls is SYNCPOINT, rather than NO_SYNCPOINT.
- To use the default (SYNCPOINT) option for MQPUT, MQGET and MQPUT1 operations, the application must have an active TM/MP Transaction that defines the unit of work to be committed. An application initiates a TM/MP transaction by calling the BEGINTRANSACTION() function. All MQPUT, MQPUT1 and MQGET operations performed by the application while this transaction is active are within the same unit of work (transaction). Any other database operations performed by the application are also within this UOW. Note that there are system-imposed limits on the number and size of messages that can be written and deleted within a single TM/MP transaction. When the application has completed the UOW, the TM/MP transaction is ended (the UOW is committed) using the ENDTRANSACTION() function. If any error is encountered, the application can cancel the TM/MP transaction (backout the UOW) using the ABORTTRANSACTION() function. Consequently, the standard Version 2 functions MQCMIT() and MQBACK() are not supported on this product. If they are called, an error is returned.
- If an application uses the NO_SYNCPOINT option for MQPUT, MQGET and MQPUT1 operations, MQSeries starts a TM/MP transaction itself, performs the queueing operation, and commits the transaction before returning to the application. Each operation is therefore performed in its own UOW and, once complete, cannot be backed out by the application using TM/MP.
- A TM/MP transaction does not need to be active for MQI calls other than MQGET, MQPUT and MQPUT1.
- Because TM/MP can cause previously performed MQGET, MQPUT, and MQPUT1 operations to be backed out without notification, the current queue-depth counts of queues can become inaccurate. The MQSS Server runs in the background to correct such inaccuracies, and can be configured to do this on a queue-by-queue basis. However, applications should be coded to be resilient to inaccuracies in the CURDEPTH attribute, especially in an environment that might involve backed-out transactions.

Writing applications

- The back-out count attribute cannot be maintained in the same way as on standard Version 2 implementations. Also, the harden backout count attribute is not used.
- The MQRC_SYNCPOINT_LIMIT_REACHED reason code is used by MQSeries for Tandem NSK V2.2.0.1 to inform an application that the system-imposed limit on the number of I/O operations within a single TM/MP transaction has been reached. If the application specified the SYNCPOINT option, it should cancel the transaction (backout the UOW) and retry with a smaller number of operations in that UOW.
- The MQRC_UOW_CANCELED reason code informs the application that the UOW (TM/MP transaction) has been canceled, either by the system itself (TM/MP imposes some system-wide resource-usage thresholds that will cause this), by user action, or by the initiator of the transaction itself.

General design considerations

Please note that:

- The MQI library (bound into the application process) does not open \$RECEIVE and does not open \$TMP (TM/MP transaction pseudo-file) itself, so you may code your application to use these features.
- The MQI library uses a SERVERCLASS_SEND_() call in initial communication with the Queue Manager. While connected, it maintains two process file opens (with the LINKMON process and a Local Queue Manager Agent) and a small number of disk file opens (fewer than 10).

XA interface

No XA interface for unit of work (UOW) coordination is provided. All UOW coordination is performed by TM/MP.

MQGMO_BROWSE_* with MQGMO_LOCK

As a consequence of the use of TM/MP, MQGMO_BROWSE_* with MQGMO_LOCK is not supported.

Triggered applications

Triggered MQSeries applications in the Tandem NSK environment receive user data through environment variables set up in the TACL process that is running. This is because there is a limit to the length of the argument list that can be passed to a Tandem C process.

In order to access this information, triggered applications should contain code similar to the following (see sample `amqsinqa` for more details):

```

MQTMC2 *trig;                /* trigger message structure */
MQTMC2 trigdata;            /* trigger message structure */
char    *applId;
char    *envData;
char    *usrData;
char    *qmName;

/*****
/*
/*  Set the program argument into the trigger message
/*
/*
/*****
trig = (MQTMC2*)argv[1];    /* -> trigger message */

/* get the environment variables and load the rest of the trigger */
memcpy(&trigdata, trig, sizeof(trigdata));

memset(trigdata.ApplId, ' ', sizeof(trigdata.ApplId));
memset(trigdata.EnvData, ' ', sizeof(trigdata.EnvData));
memset(trigdata.UserData, ' ', sizeof(trigdata.UserData));
memset(trigdata.QMgrName, ' ', sizeof(trigdata.QMgrName));

if( (applId = getenv("TRIGAPPLID")) != 0)
{
    strncpy( trigdata.ApplId ,applId, strlen(applId) );
}

if ( (envData = getenv("TRIGENVDATA")) != 0)
{
    strncpy( trigdata.EnvData , envData, strlen(envData) );
}

if ( (usrData = getenv("TRIGUSERDATA")) != 0)
{
    strncpy( trigdata.UserData, usrData, strlen(usrData) );
}

if ( (qmName = getenv("TRIGQMGRNAME")) != 0)
{
    strncpy( trigdata.QMgrName, qmName, strlen(qmName) );
}

trig = &trigdata;

```

Compiling and binding applications

The MQSeries for Tandem NSK V2.2.0.1 MQI is implemented using the Tandem wide memory model (the `int` datatype is 4 bytes) and the Common Runtime Environment (CRE). Applications must be compatible with this environment in order to work correctly. Refer to the sample build files for the correct options for each compiler in order to ensure compatibility.

In particular, TAL and COBOL applications must follow the rules that are required for compatibility with the CRE, documented in the Tandem manuals relating to the CRE.

Note that, for successful use of the `MQGMO_SET_SIGNAL` function of `MQGET`, you must set the `HIGHREQUESTERS` attribute to `ON` in object code for COBOL and TAL applications.

For a nonnative installation, four versions of the MQI library are delivered with MQSeries for Tandem NSK V2.2.0.1, contained in `ZMQSLIB`. For a native installation, native C MQI applications may be statically linked with the `MQILIBNC` library, or may be dynamically linked with the MQSeries private `SRL` in `ZMQSLIB` (called `MQSRLLIB`). COBOL and TAL applications must be linked with the static library `MQILIBNT`. You must ensure that you use the correct library, as shown in Table 14.

Table 14. Using the correct version of the MQI library

Programming Language	Nonnative	Native/Static	Native/Dynamic
TAL or COBOL	MQMLIBT	MQMLIBNT	N/A
C	MQMLIBC	MQMLIBNC	MQSRLLIB

Running applications

In order to be able to connect to a queue manager, the environment of an application program must be correctly defined:

- The PARAM MQDEFAULTPREFIX is mandatory in the environment of all applications.
- If you have chosen an alternative (nondefault) location for your MQSINI file, an application will not be able to connect to the queue manager if the PARAM MQMACHINIFILE is not set correctly.
- TAL and COBOL applications must have the PARAM SAVE-ENVIRONMENT ON defined in their environment, or they will not be able to connect to the queue manager.

An application may run as either low-pin or high-pin. MQSeries executables themselves are configured to run as high-pin.

MQSeries applications are supported in the NSK environment only. No support for OSS applications is provided.

An MQSeries application may run under PATHWAY, from TACL, or as a child process of another process. Applications can even be added to the queue manager PATHWAY configuration itself, provided they behave correctly on queue manager shutdown.

Running applications

Appendix H. MQSeries for Tandem NSK sample programs

The following C and COBOL sample programs are supplied with MQSeries for Tandem NSK, V2.2.0.1:

Description	C source	C executable	COBOL85 source	COBOL85 executable
Read and output message descriptor and context for each message on a queue	amqsbcg0	amqsbcg	No sample	No sample
Echo a message from a message queue to the reply-to queue	amqsecha	amqsech	amq0ech0	amq0ech
Write messages from a queue to stdout, leave messages on the queue (Browse)	amqsgbr0	amqsgbr	amq0gbr0	amq0gbr
Remove messages from the named queue and write to stdout	amqsget0	amqsget	amq0get0	amq0get
Read the triggered queue, respond with queue information	amqsinqa	amqsinq	No sample	No sample
Use a shared input queue	No sample	No sample	amq0inq0	amq0inq
Copy stdin to a message and put the message on a specified queue	amqsput0	amqsput	amq0put0	amq0put
Put a request message on a specified queue and display the replies	amqsreq0	amqsreq	amq0req0	amq0req
(Trigger function) inhibit puts on a named queue and respond with a statement of the result	amqsseta	amqssset	amq0set0	amq0set
Trigger monitor	amqstrg0	amqstrg	No sample	No sample
Sample skeleton for data conversion exit	amqsvfcn	No sample	No sample	No sample
Sample skeleton for channel exit	amqsvchn	No sample	No sample	No sample

The following TAL sample programs are supplied with MQSeries for Tandem NSK V2.2.0.1:

Description	TAL source	TAL executable
Read <i>n</i> messages from a queue	zmqreadt	zmqread
Write <i>n</i> messages of <i>n</i> length to a queue	zmqwritt	zmqwrit

Building C sample programs

The subvolume ZMQSSMPL contains the following TACL macro files to be used for building nonnative sample C applications:

CSAMP Usage: CSAMP *source-code-file-name*

This is a basic macro for compiling a C source file using the include files contained in subvolume ZMQSLIB. For example, to compile the sample AMQSBCG0, use CSAMP AMQSBCG0. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example, AMQSBCGO.

BSAMP Usage: BSAMP *exe-file-name*

This is a basic macro used to bind an object file with the user libraries in ZMQSLIB. For example, to bind the compiled sample AMQSBCG0, use BSAMP AMQSBCG. The macro produces an executable file called *exe-file-nameE*; for example, AMQSBCGE.

COMPALL Usage: COMPALL

This TACL macro compiles each of the sample source code files using the CSAMP macro.

BINDALL Usage: BINDALL

This TACL macro binds each of the sample object files into executables using the BSAMP macro.

BUILDC Usage: BUILDC

This TACL macro compiles and binds all of the C sample files using the macros COMPALL and BINDALL.

For a native install, the following TACL macro files are to be used for building sample MQI applications:

NMCCALL Usage: NMCCALL

Macro to compile all samples native using NMCSAMP.

NMCP SRL Usage: NMCP SRL *source-code-file-name*

Macro to compile user code for inclusion in the MQSeries PSRL.

NMCSAMP Usage: NMCSAMP *source-code-file-name*

This is a basic macro for compiling a C source file using the include files contained in subvolume ZMQSLIB. For example, to compile the sample AMQSBCG0, use NMCSAMP AMQSBCG0. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example, AMQSBCGO.

NMLDSAMP Usage: NMLDSAMP *exe-file-name*

This basic macro links an object file with the static MQI library in ZMQSLIB.

NMLDPSRL Usage: NMLDPSRL *exe-file-name*

This basic macro links an object file with the MQSeries private SRL in ZMQSLIB

NMLDUSRL Usage: NMLDUSRL *object-input-file*, where *object-input-file* is a file containing a list of objects to be linked.

This is a basic macro for linking user code into a relinkable library.

Note: Nonnative applications can connect to native queue managers, and vice versa. All combinations of native and nonnative operation are valid and supported.

Building COBOL sample programs

The subvolume ZMQSSMPL contains the following files to be used for building sample COBOL applications.

COBSAMP Usage: COBSAMP *source-code-file-name*

This is a basic macro for compiling a COBOL source file using the definition files contained in subvolume ZMQSLIB. For example, to compile the program AMQ0GBR0, use COBSAMP AMQ0GBR0. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example AMQ0GBRO.

BCOBSAMP Usage: BCOBSAMP *exe-file-name*

This is a basic macro used to bind an object with the user libraries in ZMQSLIB. For example to bind the compiled sample AMQ0GBRO, use BCOBSAMP AMQ0GBR. The macro produces an executable called *exe-file-name* AMQ0GBR.

CCBSMPLS Usage: CCBSMPLS

This TACL macro compiles each of the COBOL sample source code files.

BCBSMPLS Usage: BIND /IN BCBSMPLS/

This bind input file binds each of the COBOL sample object files into executables.

BUILDLOB Usage: BUILDLOB

This TACL macro compiles and binds all of the COBOL sample files using the macros CCBSMPLS and BCBSMPLS.

Building TAL sample programs

The subvolume ZMQSSMPL contains the following files to be used for building sample TAL programs.

TALSAMP Usage: TALSAMP *source-code-file-name*

This is a basic macro for compiling a TAL source file using the definition files contained in subvolume ZMQSLIB. For example, to compile the program ZMQWRITT, use TALSAMP ZMQWRITT. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example ZMQWRITO.

BTALSAMP Usage: BTALSAMP *exe-file-name*

This is a basic macro used to bind an object with the user libraries in ZMQSLIB. For example to bind the compiled sample ZMQWRITO, use BTALSAMP ZMQWRIT.

CTLSMPLS Usage: CTLSMPLS

This TACL macro compiles each of the TAL sample source code files.

BTLSMPLS Usage: BIND /IN BTLSMPLS/

This bind input file binds each of the TAL sample object files into executables.

BUILDTAL Usage: BUILDTAL

This TACL macro compiles and binds all of the TAL sample files using the macros CTLSMPLS and BTLSMPLS.

Building and using data-conversion exit functions

The way in which you build and use data-conversion exits depends on whether you are using the native or nonnative version of MQSeries.

Nonnative installation

Dynamically bound libraries are not supported by MQSeries for Tandem NSK. Data conversion exits (and channel exits) are implemented by including statically bound stub functions in the MQSeries libraries and executables that can be replaced using the REPLACE bind option.

A data conversion exit **must** be called DATACONVEXIT (see sample AMQSVFCN), and can be bound into the chosen executable (or library) using the TACL macro BEXITE.

Note: This procedure modifies the target executable; you are recommended to make a back-up copy of the target executable or library before using the macro.

Exit functions, once compiled, must be bound directly into the target executable or library to be accessible by MQSeries. The following TACL macro is used for this purpose:

BEXITE Usage: BEXITE *target-executable-or-library*
source-exit-file-or-library

For example, to bind the sample data conversion exit into the sample MQSGETA, follow these steps:

1. Compile the exit function DATA CONVEXIT (CSAMP AMQSVFCN).
2. Compile the get application (CSAMP AMQSGET0).
3. Bind the get application (BSAMP AMQSGET).
4. Bind the exit function into the get application (BEXITE AMQSGET AMQSVFCO).

Alternatively, if all applications are to have this data conversion exit, the following steps would create both a user library and an application with the exit bound in:

1. Compile the exit function DATA CONVEXIT (CSAMP AMQSVFCN).
2. Compile the get application (CSAMP AMQSGET0).
3. Bind the exit function into the user library (BEXITE ZMQSLIB.MQMLIBC AMQSVFCO).
4. Bind the get application with the modified library (BSAMP AMQSGET).

If the data conversion exit is to be used by channels processing within MQSeries, it must also be bound into the caller executable by the system administrator. For example:

```
BEXITE ZMQSEXE.MQMCCAL AMQSVFCO
```

Use the TACL macro BDXALL to bind the data conversion exit into all required MQSeries processes. For example:

```
BDXALL source-exit-file-or-library
```

Native installation

For the native version of MQSeries, data-conversion exits can be implemented by linking with the static MQI library, or by creating a new version of the MQSeries private SRL.

A data-conversion exit must be called DATA CONVEXIT and can be linked with the chosen application and the MQI library by using the TACL macro NMLDEXIT.

```
NMLDEXIT object-file exit-object-file
```

For example, to link the sample data-conversion exit into the sample MQSGETA, follow these steps

1. Compile the exit function DATA CONVEXIT (NMCSAMP AMQSVFCN).
2. Compile the get application (NMCSAMP AMQSGET0).
3. Link the get application with the exit (NMLDEXIT AMQSGET0 AMQSVFCNO).

If all applications are using the MQSeries private SRL, the following steps will create a new version of the MQSeries private SRL containing the new data-conversion exit:

1. Compile the exit function DATA CONVEXIT for use in the SRL (NMCP SRL AMQSVFCN).
2. Compile the get application (NMCSAMP AMQSGET0).

Building channel exits

3. Create a new version of the MQSeries private SRL called NEWMQSRL by linking this data conversion object with the relinkable version of the MQSeries private SRL (MAKEPSRL ZMQSLIB.MQSRLLNK AMQSVFCN NEWMQSRL).
4. Stop all queue managers and applications accessing the current MQSeries private SRL.
5. Relink all MQSeries applications to the new PSRL (NMLDEXES NEWMQUSRL ZMQSEXE).
6. Relink all user applications to the new PSRL, for example AMQSGET (NMLDPSRL AMQSGET).
7. Restart MQSeries and all MQI applications.

Steps 5 and 6 are quite fast, but can be omitted if the new MQSeries PSRL is placed in ZMQSLIB and called QSRLLIB. The steps can be repeated to link to a different MQSeries PSRL.

You are recommended to take a backup of the MQSeries PSRL before you replace it.

Building and using channel exit functions

The way in which you build and use channels exits depends on whether you are using the native or nonnative version of MQSeries.

Nonnative installation

Dynamically bound libraries are not supported by MQSeries for Tandem NSK. Channel exits (and data-conversion exits) are implemented by including statically bound stub functions in the MQSeries libraries and executables which can be replaced using the REPLACE bind option.

A channel exit function must be written in C, **must** be called CHANNELEXIT (see sample AMQSVCHN), and can be bound into the chosen executable (or library) using the TACL macro BEXITE.

Note: This procedure modifies the target executable. Therefore, you are recommended to make a back-up copy of the target executable or library before using the macro.

The function CHANNELEXIT must handle each of the possible exit calls (security, message-retry, message, send, and receive). You may write your own functions to do this.

Use the TACL macro BCHXALL to bind the data conversion exit into all required MQSeries processes. For example:

```
BCHXALL source-exit-file-or-library
```

Native installation

For the native version of MQSeries, channel exits are implemented by creating a new version of the MQSeries private SRL.

A channel-exit function must be written in C, **must** be called CHANNELEXIT (see sample AMQSVCHN), and must be linked into the MQSeries private SRL.

The following steps create a new version of the MQSeries private SRL containing the new channel exit function:

1. Compile the exit function CHANNELEXIT for use in the SRL (NMCPSRL AMQSVFCN).
2. Create a new version of the MQSeries private SRL called (for example) NEWMQSRL by linking this channel exit object file with the relinkable version of the MQSeries private SRL (MAKEPSRL ZMQSLIB.MQSRLLNK AMQSVFCN NEWMQSRL).
3. Stop all queue managers, and applications accessing the current MQSeries private SRL.
4. Relink all MQSeries applications to the new PSRL (NMLDEXES NEWMQUSRL ZMQSEXEXE).
5. Restart MQSeries and all MQI applications.

Steps 5 and 6 are quite fast, but can be omitted if the new MQSeries PSRL is placed in ZMQSLIB and called QSRLLIB. The steps can be repeated to link to a different MQSeries PSRL.

You are recommended to take a backup of the MQSeries PSRL before you replace it.

The function CHANNELEXIT must handle each of the possible exit calls.

Building channel exits

Appendix I. User exits

MQSeries for Tandem NSK V2.2.0.1 supports both channel exit programs and data-conversion exit programs. For information about channel exits, see the *MQSeries Intercommunication* book. For information about data-conversion exits, see the *MQSeries Application Programming Guide* and the *MQSeries Application Programming Reference*.

Note: For native installations, the linking of user-defined exits requires the `-allow_duplicate_procs` option provided with the D44 or later versions of the native linker NLD.

This appendix provides information specific to the use of exit programs in MQSeries for Tandem NSK.

Channel exit programs

In MQSeries for Tandem NSK, a user exit program can be called at the following points in MCA (message channel agent) processing:

- Channel security exit
- Channel message-retry exit
- Channel message exit
- Channel send exit
- Channel receive exit

MQSeries for Tandem NSK supports a single, statically bound channel exit program, whose entry point is `MQCHANNELEXIT()`. MQSeries for Tandem NSK provides a stub function for this exit that acts as a place holder for user-supplied exit code. In the supplied stub function, the *ExitResponse* field in `MQCXP` (channel exit parameter structure) is set to `MQXCC_CLOSE_CHANNEL`, which causes the MCA to close the channel. No other fields in `MQCXP` are modified.

For nonnative installations, you replace the supplied stub function in the MCA executable images with your own user exit code using the Tandem `BIND` utility. For native installations, you replace the supplied stub function in the MQSeries `PSRL` with your own user exit code creating a new version of the MQSeries `PSRL` using Tandem NSK native linker.

Only the Tandem Common Runtime Environment (CRE) interface for the WIDE memory model is supported.

Reusing channel-exit programs

In MQSeries for Tandem NSK, there is a single entry point for all channel exits. In other MQSeries Version 2 products, there are entry points specific to each channel type and function. It is possible to use channel-exit programs written for other MQSeries Version 2 products by calling those programs from `MQCHANNELEXIT()`. To determine the type of exit being called, examine the *ExitId* field of `MQCXP`, then extract the associated exit-program name from the *MsgExit*, *MsgRetryExit*, *ReceiveExit*, *SendExit*, or *SecurityExit* field of `MQCD`.

Channel attributes

The channel attributes that define the names of user exits are:

- Security exit name (SCYEXIT)
- Message-retry exit name (MREXIT)
- Message exit name (MSGEXIT)
- Send exit name (SENDEXIT)
- Receive exit name (RCVEXIT)

If these channel attributes are left blank, the channel user exit is not invoked. If any of the channel attributes is nonblank, the MQCHANNELEXIT() user exit program is invoked for the corresponding function. Note that the text-string value of the channel attribute is not used to determine the name of the user exit program, since only a single entry point, MQCHANNELEXIT(), is supported in MQSeries for Tandem NSK. However, the values of these channel attributes are passed to MQCHANNELEXIT() in the MQCD (channel data) structure. The function of the channel exit (that is, whether the exit corresponds to a Message, Message-retry, Receive, Security or Send Exit) is passed to MQCHANNELEXIT() in the *ChannelExitParms* parameter of the MQCXP (Channel Exit Parameters) structure.

MQSeries for Tandem NSK does not support the following channel attributes:

- CICS Profile Name
- Sequential delivery
- Target system identifier
- Transaction identifier
- Maximum transmission size

Data-conversion exit programs

A data-conversion exit can be called to convert messages of application-defined format. The data-conversion exit is invoked during the processing of an MQGET call in the following two circumstances:

1. A message is retrieved by an application using MQGET specifying the MQGMO_CONVERT option, and the message descriptor (MQMD) contains a *Format* value that does not represent one of the built-in formats (MQFMT_*).
2. A message is retrieved from a transmission queue by an MCA (which uses MQGET internally) for transmission to a remote queue manager over a channel defined with CONVERT(YES), and the message descriptor (MQMD) of the message contains a *Format* value that is not one of the built-in formats (MQFMT_*).

The MQXCNVC call is used by the data-conversion exit program to convert characters from one character set to another.

MQSeries for Tandem NSK supports a single, statically bound data-conversion exit whose entry point is MQDATACONVEXIT(). MQSeries for Tandem NSK provides a stub function for this exit that acts as a placeholder for user-supplied exit code. In the supplied stub function, the *Reason* field in MQDXP (the data-conversion exit parameter structure) is set to MQRC_NOT_CONVERTED. No other parameters are modified. This value causes the caller of MQGET to receive the *CompCode* MQCC_WARNING and the *Reason* code MQRC_NOT_CONVERTED.

You replace the supplied stub version of MQDATA CONVEXIT() with your own data-conversion exit program using the Tandem BIND utility. To support data conversion on receipt of messages (that is, when an application issues MQGET), the MQDATA CONVEXIT() function in the local queue manager (LQM) agent executable image must be replaced. To support data conversion on sending of messages, the MQDATA CONVEXIT() function in the MCA executable image must be replaced.

The interface to MQDATA CONVEXIT(), the associated data structures MQCXP, MQCD, and MQDXP, and the MQXCNVC call are defined in the *MQSeries Application Programming Reference* manual. Guidance information is provided in the *MQSeries Application Programming Guide*. Note that only the Tandem Common Runtime Environment (CRE) interface for the WIDE memory model is supported.

Reusing data-conversion exit programs

In other MQSeries Version 2 products, a data-conversion exit is required for each application-defined format to be supported. The data-conversion exit programs are named according to the *Format* value (from MQMD) of the message to be converted. The format for which conversion is being requested can be determined from the *Format* field of the *MsgDesc* parameter. The appropriate data-conversion exit program can therefore be invoked from MQDATA CONVEXIT(). The parameters supplied to MQDATA CONVEXIT() can be supplied to the invoked data-conversion function.

Data-conversion exit programs

Appendix J. Setting up communications

This appendix describes how to set up communications for MQSeries for Tandem NSK using the SNA and TCP/IP communications protocols. The following examples are provided:

- “SNAX communications example” on page 335
- “ICE communications example” on page 342
- “TCP/IP communications example” on page 345

SNA channels

The following channel attributes are necessary for SNA channels in MQSeries for Tandem NSK V2.2.0.1:

CONNAME

The value of CONNAME depends on whether SNAX or ICE is used as the communications protocol:

If SNAX is used:

CONNAME('\$PPPP.LOCALLU.REMOTELU')

Applies to sender, requester and fully qualified server channels, where:

\$PPPP Is the process name of the SNAX/APC process.
LOCALLU Is the name of the Local LU.
REMOTELU Is the name of the partner LU on the remote machine.

For example:

```
CONNAME(' $BP01.IYAHT080.IYCNVM03')
```

CONNAME('\$PPPP.LOCALLU')

Applies to receiver and non-fully qualified server channels, where:

\$PPPP Is the process name of the SNAX/APC process.
LOCALLU Is the name of the Local LU. This value can be an asterisk (*), indicating any name.

For example:

```
CONNAME(' $BP01.IYAHT080')
```

If ICE is used:

CONNAME('\$PPPP.#OPEN.LOCALLU.REMOTELU')

Applies to sender, requester and fully qualified server channels, where:

\$PPPP Is the process name of the ICE process.
#OPEN Is the ICE open name.
LOCALLU Is the name of the Local LU.
REMOTELU Is the name of the partner LU on the remote machine.

For example:

```
CONNAME(' $ICE.#IYAHT0C.IYAHT0C0.IYCNVM03')
```

CONNAME('\$PPPP.#OPEN.LOCALLU')

Applies to receiver and non fully qualified server channels, where:

\$PPPP Is the process name of the SNAX/APC process.
#OPEN Is the ICE open name.

Setting up communications

LOCALLU Is the name of the Local LU. This value can be an asterisk (*), indicating any name.

For example:

```
CONNNAME('$ICE.#IYAHT0C.IYAHT0C0')
```

MODENAME

Is the SNA mode name. For example, MODENAME(LU62PS).

TPNAME('LOCALTP[.REMOTETP]')

Is the Transaction Process (TP) name.

LOCALTP Is the local name of the TP.

REMOTETP Is the name of the TP on the remote machine. This value is optional. If it is not specified, and the channel is one that initiates a conversation (that is, a sender, requester, or fully qualified server channel) the LOCALTP name is used.

Both the LOCALTP and REMOTETP values can be up to 16 characters in length.

Note: If SNAX is being used to facilitate SNA communications, the values in the LOCALTP field in the TPNAME must match TPs defined to SNAX. If ICE is being used, TPNAMEs do not need to be defined to ICE; they need only be present in the MQSeries channel definitions.

LU 6.2 responder processes

There is no SNA listener process in MQSeries for Tandem NSK. Each channel initiated from a remote system (receiver, server, or requester that has a fully qualified server on the remote system or a requester that has a sender on the remote system) must have its own, unique TP name on which it can listen. This TP name is specified as the LOCALTP value.

Such channels must be defined to MQSC with the attribute AUTOSTART(ENABLED) to ensure that there is an LU 6.2 responder process listening on this TP name whenever the queue manager is started. This LU 6.2 responder process (MQLU6RES) services incoming SNA requests for its particular TP. If the channel is newly defined, or has been recently altered, an LU 6.2 responder process can be started for that channel by issuing either the MQSC command START CHANNEL (using **runmqsc**) or the **runmqchl** control command from the TACL prompt.

SNA channels defined AUTOSTART(DISABLED) do not listen for incoming SNA requests. LU 6.2 responder processes are not started for such channels. A message is logged to MQERRLG1 whenever an LU 6.2 responder process is started.

TCP/IP channels

For information about using a nondefault TCP/IP process for communications via TCP/IP, see "Reconfiguring a queue manager for a nondefault TCP/IP process" on page 67. For information about the TCP/IP ports a queue manager listens on, see "TCP/IP ports listened on by the queue manager" on page 66.

Communications examples

This section provides communications setup examples for SNA (SNAX and ICE) and TCP/IP.

SNAX communications example

This section provides:

- An example SCF configuration file for the SNA line
- Some example SYSGEN parameters to support the line
- An example SCF configuration file for the SNA process definition
- Some example MQSC channel definitions

SCF SNA line configuration file

Here is an example SCF configuration file:

```

==
== SCF configuration file for defining SNA LINE, PUs and LUs to VTAM®
== Line is called $SNA02 and SYSGEN'd into the Tandem system
==

ALLOW ALL
ASSUME LINE $SNA02

ABORT, SUB LU
ABORT, SUB PU
ABORT

DELETE, SUB LU
DELETE, SUB PU
DELETE

==
== ADD $SNA02 LINE DEFINITION
==

ADD LINE $SNA02, STATION SECONDARY, MAXPUS 5, MAXLUS 1024, RECSIZE 2048, &
      CHARACTERSET ASCII, MAXLOCALLUS 256, &
      PUIDBLK %H05D, PUIDNUM %H312FB

==
== ADD REMOTE PU OBJECT, LOCAL IS IMPLICITLY DEFINED AS #ZNT21
==

ADD PU #PU2, ADDRESS 1, MAXLUS 16, RECSIZE 2046, TYPE (13,21), &
      TRRMTADDR 04400045121088, DYNAMIC ON, &
      ASSOCIATESUBDEV $CHAMB.#p2, &
      TRSSAP %H04, &
      CPNAME IYAQCDRM, SNANETID GBIBMIYA

==
== ADD LOCAL LU OBJECT
==

ADD LU #ZNTLU1, TYPE (14,21), RECSIZE 1024, &
      CHARACTERSET ASCII, PUNAME #ZNT21, SNANAME IYAHT080

```

Setting up communications

```
==
== ADD PARTNER LU OBJECTS
==

== spinach (HP)
ADD LU #PU2LU1, TYPE(14,21), PUNAME #PU2, SNANAME IYABT0F0
== stingray (AIX)
ADD LU #PU2LU2, TYPE(14,21), PUNAME #PU2, SNANAME IYA3T995
== coop007 (OS/2)
ADD LU #PU2LU3, TYPE(14,21), PUNAME #PU2, SNANAME IYAFT170
== MVS CICS
ADD LU #PU2LU4, TYPE(14,21), PUNAME #PU2, SNANAME IYCMVM03
== MVS Non-CICS
ADD LU #PU2LU5, TYPE(14,21), PUNAME #PU2, SNANAME IYCNVM03
== finnr100 (NT)
ADD LU #PU2LU6, TYPE(14,21), PUNAME #PU2, SNANAME IYAFT080
== winas18 (AS400)
ADD LU #PU2LU7, TYPE(14,21), PUNAME #PU2, SNANAME IYAFT110
== MQ-Portugese (OS/2)
ADD LU #PU2LU8, TYPE(14,21), PUNAME #PU2, SNANAME IYAHT090
== VSE
ADD LU #PU2LU10, TYPE(14,21), PUNAME #PU2, SNANAME IYZMZSI2

== START UP TOKEN RING ASSOCIATE SUB DEVICE $CHAMB.#P2
== then start the line, pu's and lu's

START LINE $CHAMB, SUB ALL

START
START, SUB PU

STATUS
STATUS, SUB PU
STATUS, SUB LU
```

SYSGEN parameters

The following are CONFTEXT file entries for a SYSGEN to support the SNA and token ring lines:

```

!*****
!
!                               LAN MACRO
!*****
! This macro is used for all 361x LAN controllers
! REQUIRES T9375 SOFTWARE PACKAGE

C3613^MLAM          = MLAM
                    TYPE 56,              SUBTYPE 0,
                    PROGRAM                C9376P00,
                    INTERRUPT              IOP^INTERRUPT^HANDLER,
                    MAXREQUESTSIZE        32000,
                    RSIZE                  32000,
                    BURSTSIZE              16,
                    LINEBUFFERSIZE        32,
                    STARTDOWN #;

!*****
!                               SNAX macro for Token ring lines
!*****
TOKEN^RING^SNAX^MACRO = SNATS
                    TYPE 58,
                    SUBTYPE 4,
                    RSIZE 1024,
                    SUBTYPE 4,
                    FRAMESIZE 1036 # ;

!*****
!                               SNAX MANAGER
!*****
SSCP^MACRO          = SNASVM
                    TYPE 13,              SUBTYPE 5,
                    RSIZE                  256 #;

!*****
!                               LAN CONTROLLER
!*****
LAN1      3616    0,1    %130    ;

!*****      Service manager
SNAX      6999    0,1    %370    ;

!*****      SNAX/Token Ring Pseudocontroller
RING      6997    0,1    %360    ;

!*****      Token Ring Line
$CHAMB    LAN1.0, LAN1.1      C3613^MLAM, NAME #LAN1;

!*****      Configure the SSCP
$SSCP     SNAX.0, SNAX.1 SSCP^MACRO;

!*****      Sna lines for Dummy Controller over Token Ring
$SNA01    RING.0, RING.1 TOKEN^RING^SNAX^MACRO;
$SNA02    RING.2, RING.3 TOKEN^RING^SNAX^MACRO;

```

Setting up communications

SNAX/APC process configuration

The following definitions configure the example APC process (process name \$BP01) via SCF for the SNA line.

Note: The pathway process \$BP01 is created using the Tandem utility APCRUN.

```
==
== SCF Configuration file for SNAX/APC Lus
==

ALLOW ERRORS

ASSUME PROCESS $BP01

ABORT SESSION *
ABORT TPN *
ABORT PTNR-MODE *
ABORT PTNR-LU *
ABORT LU *

DELETE TPN *
DELETE PTNR-MODE *
DELETE PTNR-LU *
DELETE LU *

==
== ADD LOCAL LU
==
ADD LU IYAHT080, SNANAME GBIBMIYA.IYAHT080, SNAXFILENAME $SNA02.#ZNTLU1, &
      MAXSESSION 256, AUTOSTART YES

== TPnames for MQSeries

ADD TPN IYAHT080.INTCRS6A
ADD TPN IYAHT080.DUMMY, GENERALTPREADY yes, SESSIONCONTROL yes, &
      REMOTEATTACHTIMER -1, REMOTEATTACH queue

=== Spinach (HP) Partner LU

ADD PTNR-LU IYAHT080.IYABT0F0, SNANAME GBIBMIYA.IYABT0F0, &
      PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES

ADD PTNR-MODE IYAHT080.IYABT0F0.LU62PS, MODENAME LU62PS, &
      DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
      DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
      DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
      SENDWINDOW 4

ADD TPN IYAHT080.MH01SDRCSDR
ADD TPN IYAHT080.MH01RQSDSDR
ADD TPN IYAHT080.MH01RQSVSVR
ADD TPN IYAHT080.MH01SDRRCRCVR
ADD TPN IYAHT080.MH01RQSVRQSTR
ADD TPN IYAHT080.MH01RQSDRQSTR

==
== Winas18 (AS400) Partner LU
==
```



```

ADD PTNR-LU    IYAHT080.IYAFT110, SNANAME GBIBMIYA.IYAFT110, &
                PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES

ADD PTNR-MODE IYAHT080.IYAFT110.LU62PS, MODENAME LU62PS, &
                DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
                DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
                DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
                SENDWINDOW 4

ADD TPN       IYAHT080.M401SDRCSDR
ADD TPN       IYAHT080.M401RQSDSDR
ADD TPN       IYAHT080.M401RQSVSVR
ADD TPN       IYAHT080.M401SDRCRCVR
ADD TPN       IYAHT080.M401RQSVRQSTR
ADD TPN       IYAHT080.M401RQSDRQSTR

==
== Stingray (AIX) Partner LU
==

ADD PTNR-LU    IYAHT080.IYA3T995, SNANAME GBIBMIYA.IYA3T995, &
                PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES

ADD PTNR-MODE IYAHT080.IYA3T995.LU62PS, MODENAME LU62PS, &
                DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
                DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
                DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
                SENDWINDOW 4

ADD TPN       IYAHT080.MA02SDRCSDR
ADD TPN       IYAHT080.MA02RQSDSDR
ADD TPN       IYAHT080.MA02RQSVSVR
ADD TPN       IYAHT080.MA02SDRCRCVR
ADD TPN       IYAHT080.MA02RQSVRQSTR
ADD TPN       IYAHT080.MA02RQSDRQSTR

==
== coop007 (OS/2) Partner LU
==

ADD PTNR-LU    IYAHT080.IYAFT170, SNANAME GBIBMIYA.IYAFT170, &
                PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES

ADD PTNR-MODE IYAHT080.IYAFT170.LU62PS, MODENAME LU62PS, &
                DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
                DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
                DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
                SENDWINDOW 4

ADD TPN       IYAHT080.M002SDRCSDR
ADD TPN       IYAHT080.M002RQSDSDR
ADD TPN       IYAHT080.M002RQSVSVR
ADD TPN       IYAHT080.M002SDRCRCVR
ADD TPN       IYAHT080.M002RQSVRQSTR
ADD TPN       IYAHT080.M002RQSDRQSTR

```

Setting up communications

```
==
== MQ-Portugese (OS/2) Partner LU
==

ADD PTNR-LU    IYAHT080.IYAHT090, SNANAME GBIBMIYA.IYAHT090, &
                PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES

ADD PTNR-MODE IYAHT080.IYAHT090.LU62PS, MODENAME LU62PS, &
                DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
                DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
                DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
                SENDWINDOW 4

==
== finnrr100 (NT) Partner LU
==

ADD PTNR-LU    IYAHT080.IYAFT080, SNANAME GBIBMIYA.IYAFT080, &
                PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES

ADD PTNR-MODE IYAHT080.IYAFT080.LU62PS, MODENAME LU62PS, &
                DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
                DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
                DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
                SENDWINDOW 4

ADD TPN        IYAHT080.MW01SDRCS DR
ADD TPN        IYAHT080.MW01RQSDSDR
ADD TPN        IYAHT080.MW01RQSVSVR
ADD TPN        IYAHT080.MW01SDRCRCVR
ADD TPN        IYAHT080.MW01RQSVRQSTR
ADD TPN        IYAHT080.MW01RQSDRQSTR

==
== MVS CICS          Partner LU
==

ADD PTNR-LU    IYAHT080.IYCMVM03, SNANAME GBIBMIYA.IYCMVM03, &
                PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES

ADD PTNR-MODE IYAHT080.IYCMVM03.LU62PS, MODENAME LU62PS, &
                DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
                DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
                DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
                SENDWINDOW 4

ADD TPN        IYAHT080.VM03SDRCS DR
ADD TPN        IYAHT080.VM03RQSDSDR
ADD TPN        IYAHT080.VM03RQSVSVR
ADD TPN        IYAHT080.VM03SDRCRCVR
ADD TPN        IYAHT080.VM03RQSVRQSTR
ADD TPN        IYAHT080.VM03RQSDRQSTR

==
== MVS Non CICS Partner LU
==

ADD PTNR-LU    IYAHT080.IYCNVM03, SNANAME GBIBMIYA.IYCNVM03, &
                PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES
```

```
ADD PTNR-MODE IYAHT080.IYCNVM03.LU62PS, MODENAME LU62PS, &
              DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
              DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
              DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
              SENDWINDOW 4
```

```
ADD TPN      IYAHT080.VM03NCMSDRCSDR
ADD TPN      IYAHT080.VM03NCMRQSDSDR
ADD TPN      IYAHT080.VM03NCMRQSVSVR
ADD TPN      IYAHT080.VM03NCMSDRRCVR
ADD TPN      IYAHT080.VM03NCMRQSVRQSTR
ADD TPN      IYAHT080.VM03NCMRQSDRQSTR
```

```
==
== VSE Partner LU
==
```

```
ADD PTNR-LU  IYAHT080.IYZMZSI2, SNANAME GBIBMIYA.IYZMZSI2, &
              PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES
```

```
ADD PTNR-MODE IYAHT080.IYZMZSI2.LU62PS, MODENAME LU62PS, &
              DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
              DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
              DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
              SENDWINDOW 4
```

```
==
== Start the LUs
==
```

```
START LU IYAHT080, SUB ALL
START TPN *
```

MQSeries applications require the Maxapplio value, which controls the maximum size of interprocess data transfers between MQSeries and the communications server process, to be set to 32000, which is larger than the default.

Channel definitions

Here are some example MQSeries channel definitions that support the SNAX configuration:

- A sender channel to MQSeries on MVS/ESA (non-CICS mover):

```
DEFINE CHANNEL(MT01.VM03.SDR0.0002) CHLTYPE(SDR) +
              TRPTYPE(LU62) +
              SEQWRAP(9999999) MAXMSGL(2048) +
              XMITQ('VM03NCM.TQ.SDR0.0001') +
              CONNAME('$BP01.IYAHT080.IYCNVM03') +
              MODENAME('LU62PS') TPNAME(DUMMY)
```

- A receiver channel from MQSeries on MVS/ESA:

```
DEFINE CHANNEL(VM03.MT01.SDR0.0002) CHLTYPE(RCVR) +
              TRPTYPE(LU62) REPLACE DESCR('Receiver channel from VM03NCM') +
              SEQWRAP(9999999) +
              MAXMSGL(2048) AUTOSTART(ENABLED) +
              CONNAME('$BP01.IYAHT080') TPNAME(VM03NCMSDRRCVR)
```

Setting up communications

- A server channel to MQSeries on MVS/ESA which is capable of initiating a conversation, or being initiated by a remote requester channel:

```
DEFINE CHANNEL(MT01.VM03.RQSV.0002) CHLTYPE(SVR) +
    TRPTYPE(LU62) +
    SEQWRAP(9999999) MAXMSGL(2048) +
    XMITQ('VM03NCM.TQ.RQSV.0001') +
    CONNAME('$BP01.IYAHT080.IYCNVM03') +
    MODENAME('LU62PS') TPNAME(VM03NCMRQSVSVR.DUMMY) +
    AUTOSTART(ENABLED)
```

where DUMMY is the TPNAME the MVS queue manager is listening on.

ICE communications example

There are two stages in configuring ICE for MQSeries:

1. The ICE process itself must be configured.
2. Line (\$ICE01, in the following example) and SNA information must be input to the ICE process.

Configuring the ICE process

Here is an example ICE process configuration. This configuration is located by default in a file called GOICE:

```
?tac1 macro
clear all
param backupcpu 1
param cinittimer 120
param collector $0
param config icect1
param idblk 05d
param idnum 312FF
param cpname IYAHR00C
param datapages 64
param dynamicrlu yes
param genesis $gen
param maxrcv 32000
param loglevel info
param netname GBIBMIYA
param password xxxxxxxxxxxxxxxxxxxxxx
param retrys1 5
param secuserid super.super
param startup %1%
param timer1 20
param timer2 300
param usstable default
run $system.ice.ice/name $ICE,nowait,cpu 0,pri 180,highpin off/
```

Notes:

1. The password param has been replaced by xxxxxxxxxxxxxxxxxxxxxx.
2. MQSeries applications require the maxrcv PARAM, which controls the maximum size of interprocess data transfers between MQSeries and the communications server process, to be set to 32000, which is larger than the default.

Defining the line and APC information

Once the ICE process has been started with this configuration, the following information is input to the ICE process using the Node Operator Facility (NOF**). This example defines a line called \$ICE01 running on the token ring port \$CHAMB.#ICE:

```

==
== ICE definitions for PU IYHR00C.
== Local LU for this PU is IYAHT0C0.
==

ALLOW ERRORS

OPEN $ICE

ABORT LINE $ICE01, SUB ALL

DELETE LINE $ICE01, SUB ALL

==
== ADD TOKEN RING LINE
==

ADD LINE $ICE01, TNDM $CHAMB.#ICE, &
      IDBLK %H05D, &
      PROTOCOL TOKENRING, WRITEBUFFERSIZE 8192

==
== ADD PU OBJECT
==

ADD PU IYHR00C, LINE $ICE01, MULTIRoute YES, &
      DMAC 400045121088, DSAP %H04, &
      NETNAME GBIBMIYA, IDNUM %H312FF, IDBLK %H05D, &
      RCPNAME GBIBMIYA.IYAQCDRM, SSAP %H08

==
== Add Local APPL Object
==

DELETE APPL IYAHT0C0
ADD APPL IYAHT0C0, ALIAS IYAHT0C0, LLU IYAHT0C0, PROTOCOL CPIC, &
      OPENNAME #IYAHT0C

==
== Add Mode LU62PS
==

DELETE MODE LU62PS
ADD MODE LU62PS, MAXSESS 8, MINCONWIN 4, MINCONLOS 3

==
== Add Partner LU Objects
==

== spinach (HP)

ABORT RLU IYABT0F0

```

Setting up communications

```
DELETE RLU IYABT0F0
ADD RLU IYABT0F0, MODE LU62PS, PARSESS YES
```

== stingray (AIX)

```
ABORT RLU IYA3T995
DELETE RLU IYA3T995
ADD RLU IYA3T995, MODE LU62PS, PARSESS YES
```

== coop007 (OS/2)

```
ABORT RLU IYAFT170
DELETE RLU IYAFT170
ADD RLU IYAFT170, MODE LU62PS, PARSESS YES
```

== MVS CICS

```
ABORT RLU IYCMVM03
DELETE RLU IYCMVM03
ADD RLU IYCMVM03, MODE LU62PS, PARSESS YES
```

== MVS Non-CICS

```
ABORT RLU IYCNVM03
DELETE RLU IYCNVM03
ADD RLU IYCNVM03, MODE LU62PS, PARSESS YES
```

== finnr100 (NT)

```
ABORT RLU IYAFT080
DELETE RLU IYAFT080
ADD RLU IYAFT080, MODE LU62PS, PARSESS YES
```

== winas18 (AS400)

```
ABORT RLU IYAFT110
DELETE RLU IYAFT110
ADD RLU IYAFT110, MODE LU62PS, PARSESS YES
```

```
ABORT RLU IYAHT080
DELETE RLU IYAHT080
ADD RLU IYAHT080, MODE LU62PS, PARSESS YES
```

==

```
== START UP ICE LINE $ICE01 AND SUB DEVICE
```

==

```
START LINE $ICE01, SUB ALL
```

Note: In order for this configuration to work, the port #ICE must have been defined to the token ring line. For example, these commands could be entered into SCF:

```
add port $chamb.#ice, type tr8025, address %H08
start port $chamb.#ice
```

where \$chamb is a token-ring controller, and the SAP of the port is %08.

Channel definitions for ICE

Here are some MQSeries channel definitions that would support this ICE configuration:

- A sender channel to MQSeries on MVS/ESA (non-CICS mover):

```
DEFINE CHANNEL(MT01.VM03.SDRC.ICE) CHLTYPE(SDR) +
  TRPTYPE(LU62) +
  SEQWRAP(9999999) MAXMSGL(2048) +
  XMITQ('VM03NCM.TQ.SDRC.ICE') +
  CONNAME('$ICE.#IYAHT0C.IYAHT0C0.IYCNVM03') +
  MODENAME('LU62PS') TPNAME(DUMMY)
```

- A receiver channel from MQSeries on MVS/ESA:

```
DEFINE CHANNEL(VM03.MT01.SDRC.ICE) CHLTYPE(RCVR) +
  TRPTYPE(LU62) REPLACE DESCR('Receiver channel from VM03NCM') +
  SEQWRAP(9999999) +
  MAXMSGL(2048) AUTOSTART(ENABLED) +
  CONNAME('$ICE.#IYAHT0C.IYAHT0C0') TPNAME(VM03NCMSDRCRCVR)
```

- A server channel to MQSeries on MVS/ESA that is capable of initiating a conversation, or being initiated by a remote requester channel:

```
DEFINE CHANNEL(MT01.VM03.RQSV.ICE) CHLTYPE(SVR) +
  TRPTYPE(LU62) +
  SEQWRAP(9999999) MAXMSGL(2048) +
  XMITQ('VM03NCM.TQ.RQSV.ICE') +
  CONNAME('$ICE.#IYAHT0C.IYAHT0C0.IYCNVM03') +
  MODENAME('LU62PS') TPNAME(VM03NCMRQSVSVR.DUMMY) +
  AUTOSTART(ENABLED)
```

where DUMMY is the TPNAME the MVS queue manager is listening on.

TCP/IP communications example

This example shows how to establish communications with a remote MQSeries system over TCP/IP.

TCPConfig stanza in QMINI

The QMINI file must contain an appropriate TCPConfig stanza. For example:

```
TCPConfig:
  TCPPort=1414
  TCPNumListenerPorts=1
  TCPListenerPort=1996
  TCPKeepAlive=1
```

The TCPPort value is the default outbound port for channels without a port value in the CONNAME field. TCPListenerPort identifies the port on which the TCP/IP listener will listen.

Defining a TCP/IP sender channel

A TCP/IP sender channel must be defined. In this example, the queue manager is MH01 on a host called SPINACH:

```
DEFINE CHANNEL(MT01_MH01_SDRC_0001) CHLTYPE(SDR) +
  TRPTYPE(TCP) +
  SEQWRAP(9999999) MAXMSGL(4194304) +
  XMITQ('MH01_TQ_SDRC_0001') +
  CONNAME('SPINACH.HURSLEY.IBM.COM(2000)')
```

Setting up communications

This channel would try to attach to a TCP/IP port number 2000 on the host SPINACH.

The following example shows a TCP/IP sender channel definition for a queue manager MH01 on the host SPINACH using the *default* outbound TCP/IP port:

```
DEFINE CHANNEL(MT01_MH01_SDR0001) CHLTYPE(SDR) +
    TRPTYPE(TCP) +
    SEQWRAP(9999999) MAXMSGL(4194304) +
    XMITQ('MH01_TQ_SDR0001') +
    CONNAME('SPINACH.HURSLEY.IBM.COM')
```

No port number is specified in the CONNAME. Therefore, the value specified on the TCPPort entry in the QMINI file (1414) is used.

Defining a TCP/IP receiver channel

An example TCP/IP receiver channel:

```
DEFINE CHANNEL(MH01_MT01_SDR0001) CHLTYPE(RCVR) +
    TRPTYPE(TCP)
```

A TCP/IP receiver channel requires no CONNAME value, but a TCP/IP listener must be running. There are two ways of starting a TCP/IP listener. Either:

1. Go into the queue manager's pathway using pathcom, and enter:

```
start server mqs-tcplis00
```

or

2. From the TACL prompt, enter

```
runmqtsr -m QMgrName
```

A TCP/IP listener, which will listen on the port defined in the QMINI file (in this example, 1996), is started.

Note: This port number can be overridden by the *-p Port* flag on **runmqtsr**.

Defining a TCP/IP sender channel on the remote system

The sender channel definition on the remote system to connect to this receiver channel could look like:

```
DEFINE CHANNEL(MH01_MT01_SDR0001) CHLTYPE(SDR) +
    TRPTYPE(TCP) +
    XMITQ('MT01_TQ_SDR0001') +
    CONNAME('TANDEM.ISC.UK.IBM.COM(1996)')
```


Configuring QMINI to support multiple TCP/IP listeners

To enable a queue manager to support multiple TCP/IP listeners, you must create a new PATHWAY server class for each additional listener, based on MQS-TCPLIS00.

In addition, each TCP/IP listener must have its own listener port entry in the TCPConfig stanza of the QMINI file.

For example:

```
TCPCfg:
  TCPPort=1414
  TCPNumListenerPorts=3
  TCPListenerPort=1996
  TCPListenerPort=1997
  TCPListenerPort=1998
  TCPKeepAlive=1
```

TCPNumListenerPorts must match the number of TCPListenerPort entries (3 in this example). This QMINI file is capable of supporting three TCP/IP listeners listening on ports 1996, 1997 and 1998. Typically, the server classes to support these three ports would be named MQS-TCPLIS00, MQS-TCPLIS01, and MQS-TCPLIS02.

For more information about adding server classes, see “TS/MP (Pathway) administration” on page 48.

Setting up communications

Appendix K. MQSeries clients

An MQSeries client is an MQSeries system that does not include a queue manager. The MQSeries client code directs MQI calls from applications running on the client system to a queue manager on an MQSeries server system to which it is connected.

This appendix provides information about MQSeries clients that is specific to MQSeries for Tandem NSK V2.2.0.1. It should be used in conjunction with the *MQSeries Clients* book.

Client support

MQSeries for Tandem NSK V2.2.0.1 can function as an MQSeries server system to all MQSeries clients that can connect to the server using TCP/IP or SNA LU 6.2 protocols. However, there is no MQSeries for Tandem NSK V2.2.0.1 client.

When an MQSeries client connects to a queue manager on MQSeries for Tandem NSK V2.2.0.1:

- Any MQGET, MQPUT, or MQPUT1 with an MQ*_SYNCPOINT option initiates a Tandem transaction, if one has not already been associated with the connection handle.
- Any MQGET, MQPUT, or MQPUT1 with neither an MQ*_SYNCPOINT nor an MQ*_NO_SYNCPOINT option initiates a Tandem transaction, if one has not already been associated with the connection handle.
- The MQCMIT call commits a Tandem transaction, if one is associated with the connection handle. The MQBACK call cancels the Tandem transaction, if one is associated with the connection handle.

In all cases, if the Tandem BEGINTRANSACTION fails, a *CompCode* of MQCC_FAILED, and a *Reason* of MQRC_SYNCPOINT_NOT_AVAILABLE are returned to the caller.

Security considerations

MQSeries for Tandem NSK V2.2.0.1 supports the use of channel security exits for the validation of clients, as follows:

- After a connection is established between the MQSeries client and the server, the client invokes the security exit on the server prior to returning from the MQCONN call.
- The server security exit can return information to the client security exit.

This dialog allows, for example, the communication of confidential data between the server and client. If the client has not defined a security exit, the values of the local environment variables MQ_USER_ID and MQ_PASSWORD are passed to the server via channel attributes. These attributes are available to the server security exit for validation.

Appendix L. Programmable System Management

MQSeries for Tandem NSK supports these system-management functions of MQSeries:

- Instrumentation events
- Programmable Command Formats (PCFs)
- Installable services

This appendix provides a summary of these functions in MQSeries for Tandem NSK. For detailed descriptions, see the *MQSeries Programmable System Management* book.

Instrumentation events

MQSeries for Tandem NSK supports the standard MQSeries instrumentation events, which result in the generation of an event message on an event queue.

You enable and disable events by specifying appropriate values for queue and queue manager attributes using:

- MQSC, as described in the *MQSeries Command Reference* book
- PCF commands, as described in the *MQSeries Programmable System Management* book
- Message Queue Management (MQM), as described in Chapter 5, "Managing queue managers" on page 59

Event types supported by MQSeries for Tandem NSK

MQSeries for Tandem NSK supports the following event types:

Table 15 (Page 1 of 2). Event types supported by MQSeries for Tandem NSK

Event type	Event name
Authority events	Not Authorized (type 1)
Channel events	Channel Activated Channel Conversion Error Channel Not Activated Channel Started Channel Stopped
Inhibit events	Get Inhibited Put Inhibited
Local events	Alias Base Queue Type Error Queue Type Error Unknown Alias Base Queue Unknown Object Name
Performance events	Queue Depth High Queue Depth Low Queue Full Queue Service Interval High Queue Service Interval OK

Table 15 (Page 2 of 2). Event types supported by MQSeries for Tandem NSK

Event type	Event name
Remote events	Default Transmission Queue Type Error Default Transmission Queue Usage Error Queue Type Error Remote Queue Name Error Transmission Queue Usage Error Unknown Default Transmission Queue Unknown Remote Queue Manager Unknown Transmission Queue
Start and stop events	Queue Manager Active Queue Manager Not Active

Event-message format

MQSeries for Tandem NSK supports the standard MQSeries event-message format. That is, the event message has two parts, the *message descriptor* (MQMD) and the *message data*. The message data comprises an event header and some data that is specific to the type of event.

The MQMD structure of an event message is summarized in Table 16. The event header structure (MQCFH) is summarized in Table 17 on page 353.

Table 16 (Page 1 of 2). MQMD structure of an event message

Parameter	Type	Values
<i>StrucId</i>	MQCHAR4	MQMD_STRUC_ID
<i>Version</i>	MQLONG	MQMD_VERSION_1
<i>Report</i>	MQLONG	MQRO_NONE
<i>MsgType</i>	MQLONG	MQMT_DATAGRAM
<i>Expiry</i>	MQLONG	MQEI_UNLIMITED
<i>Feedback</i>	MQLONG	MQFB_NONE
<i>Encoding</i>	MQLONG	Encoding of the queue manager generating the event.
<i>CodedCharSetId</i>	MQLONG	Coded character set ID (CCSID) of the queue manager generating the event.
<i>Format</i>	MQCHAR8	MQFMT_EVENT
<i>Priority</i>	MQLONG	Default priority of the event queue, if it is a local queue, or its local definition at the queue manager generating the event.
<i>Persistence</i>	MQLONG	Default persistence of the event queue, if it is a local queue, or its local definition at the queue manager generating the event.
<i>MsgId</i>	MQBYTE24	The value is uniquely generated by the queue manager.
<i>CorrelId</i>	MQBYTE24	MQCI_NONE
<i>BackoutCount</i>	MQLONG	The value is always 0.
<i>ReplyToQ</i>	MQCHAR48	Always blank.
<i>ReplyToQMgr</i>	MQCHAR48	The queue manager name at the originating system.

Table 16 (Page 2 of 2). MQMD structure of an event message

Parameter	Type	Values
<i>UserIdentifier</i>	MQCHAR12	Always blank.
<i>AccountingToken</i>	MQBYTE32	MQACT_NONE
<i>ApplIdentityData</i>	MQCHAR32	Always blank.
<i>PutApplType</i>	MQLONG	Type of application that put the message.
<i>PutApplName</i>	MQCHAR28	Name of the application that put the message.
<i>PutDate</i>	MQCHAR8	Date when the message was put, generated by the queue manager.
<i>PutTime</i>	MQCHAR8	Time when message was put, generated by the queue manager.
<i>ApplOriginData</i>	MQCHAR4	Always blank.

Table 17. Event header structure (MQCFH)

Parameter	Type	Values
<i>Type</i>	MQLONG	MQCFT_EVENT
<i>StrucLength</i>	MQLONG	MQCFH_STRUC_LENGTH
<i>Version</i>	MQLONG	MQCFH_VERSION_1
<i>Command</i>	MQLONG	MQCMD_Q_MGR_EVENT MQCMD_PERFM_EVENT MQCMD_CHANNEL_EVENT
<i>MsqSeqNumber</i>	MQLONG	Always 1.
<i>Control</i>	MQLONG	MQCFC_LAST
<i>CompCode</i>	MQLONG	MQCC_OK MQCC_WARNING
<i>Reason</i>	MQLONG	Reason code identifying event.
<i>ParameterCount</i>	MQLONG	The number of parameter structures that follow the MQCFH structure.

Queue statistics

Queue statistics and internal counts are updated when messages are put onto or removed from a queue. If a transaction is cancelled abnormally, these statistics can become inaccurate.

To maintain accurate values, the statistics and counts are updated whenever they are required for processing by a Programmable Command Format (PCF) message or by any MQINQ call.

Programmable command formats (PCFs)

MQSeries for Tandem NSK supports the standard Programmable Command Format (PCF) functions, as described in the *MQSeries Programmable System Management* book. PCF messages are made up of two parts, the *message descriptor* (MQMD) and the *message data*. The message data comprises a PCF header (MQCFH) and some PCF parameters defined by the structures MQCFIN, MQCFIL, MQCFST, and MQCFSL.

The PCF message descriptor (MQMD) is summarized in Table 18. The PCF header structure (MQCFH) is summarized in Table 19 on page 355. The PCF parameter structures are summarized in Table 20 on page 355 through Table 23 on page 356.

PCF message descriptor

For MQSeries for Tandem NSK, the standard PCF message descriptor applies. That is, the message descriptor contains these fields:

Table 18. PCF message descriptor

Field	Values
<i>Report</i>	Any valid value
<i>MsgType</i>	MQMT_REQUEST
<i>Expiry</i>	Any valid value
<i>Feedback</i>	MQFB_NONE
<i>Encoding</i>	Encoding used for the message data; conversion is performed if necessary.
<i>CodedCharSetId</i>	CCSID used for the message data; conversion is performed if necessary.
<i>Format</i>	MQFMT_ADMIN MQFMT_PCF (for user data)
<i>Priority</i>	Any valid value
<i>Persistence</i>	Any valid value
<i>MsgId</i>	Any valid value, including MQMI_NONE
<i>CorrelId</i>	Any valid value, including MQMI_NONE
<i>ReplyToQ</i>	Queue name
<i>ReplyToQMgr</i>	Queue manager name
Message context fields	Any valid value, including MQPMO_DEFAULT_CONTEXT

PCF header (MQCFH)

For MQSeries for Tandem NSK, the standard PCF header applies. That is, the PCF header structure contains these fields:

Table 19. PCF header

Field	Type	Values
<i>Type</i>	MQLONG	MQCFT_COMMAND MQCFT_RESPONSE MQCFT_EVENT
<i>StrucLength</i>	MQLONG	MQCFH_STRUC_LENGTH
<i>Version</i>	MQLONG	MQCFH_VERSION_1
<i>Command</i>	MQLONG	Valid command identifier.
<i>MsgSeqNumber</i>	MQLONG	Sequence number of the message.
<i>Control</i>	MQLONG	MQCFC_LAST MQCFC_NOT_LAST
<i>CompCode</i>	MQLONG	MQCC_OK MQCC_WARNING MQCC_FAILED MQCC_UNKNOWN
<i>Reason</i>	MQLONG	Reason code qualifying the completion code.
<i>ParameterCount</i>	MQLONG	Count of parameter structures.

PCF string parameter (MQCFST)

For MQSeries for Tandem NSK, the standard PCF string parameter structure (MQCFST) applies. That is, the PCF string parameter structure contains these fields:

Table 20. PCF string parameter

Field	Type	Value
<i>Type</i>	MQLONG	MQCFT_STRING
<i>StrucLength</i>	MQLONG	Length in bytes of the MQCFST structure.
<i>Parameter</i>	MQLONG	Parameter identifier.
<i>CodedCharSetId</i>	MQLONG	Coded character set identifier (CCSID).
<i>StringLength</i>	MQLONG	Length in bytes of the data in the <i>String</i> field.
<i>String</i>	MQCHAR × <i>StringLength</i>	String value.

PCF integer list parameter (MQCFIL)

For MQSeries for Tandem NSK, the standard PCF integer list parameter structure (MQCFIL) applies. That is, the PCF integer list parameter structure contains these fields:

Table 21. PCF integer list

Field	Type	Value
<i>Type</i>	MQLONG	MQCFT_INTEGER_LIST
<i>StrucLength</i>	MQLONG	Length in bytes of the MQCFIL structure.
<i>Parameter</i>	MQLONG	Parameter identifier.
<i>Count</i>	MQLONG	Number of elements in the <i>Values</i> array.
<i>Values</i>	MQLONG × <i>Count</i>	Parameter values.

PCF integer (MQCFIN)

For MQSeries for Tandem NSK, the standard PCF integer structure (MQCFIN) applies. That is, the PCF integer structure contains these fields:

Table 22. PCF integer

Field	Type	Value
<i>Type</i>	MQLONG	MQCFT_INTEGER
<i>StrucLength</i>	MQLONG	MQCFIN_STRUC_LENGTH
<i>Parameter</i>	MQLONG	Parameter identifier
<i>Value</i>	MQLONG	Parameter value

PCF string list (MQCFSL)

For MQSeries for Tandem NSK, the standard PCF string list structure (MQCFSL) applies. That is, the PCF string list structure contains these fields:

Table 23. PCF string list

Field	Type	Value
<i>Type</i>	MQLONG	MQCFT_STRING_LIST
<i>StrucLength</i>	MQLONG	Length in bytes of the MQCFSL structure
<i>Parameter</i>	MQLONG	Parameter identifier
<i>CodedCharSetId</i>	MQLONG	CCSID of the data in the <i>Strings</i> field.
<i>Count</i>	MQLONG	Number of strings in the <i>Strings</i> field.
<i>StringLength</i>	MQLONG	Length in bytes of each string in the <i>Strings</i> field.
<i>Strings</i>	MQCHAR × <i>StringLength</i> × <i>Count</i>	Set of string values for the parameter identified by the <i>Parameter</i> field.

PCF commands supported by MQSeries for Tandem NSK

The following MQSeries PCF commands are supported by MQSeries for Tandem NSK. For a complete description of these commands, see the *MQSeries Programmable System Management* book.

Table 24. PCF commands supported by MQSeries for Tandem NSK

Command	Command identifier
Change Channel	MQCMD_CHANGE_CHANNEL
Change Process	MQCMD_CHANGE_PROCESS
Change Queue	MQCMD_CHANGE_Q
Change Queue Manager	MQCMD_CHANGE_Q_MGR
Clear Queue	MQCMD_CLEAR_Q
Copy Channel	MQCMD_COPY_CHANNEL
Copy Queue	MQCMD_COPY_Q
Create Channel	MQCMD_CREATE_CHANNEL
Create Process	MQCMD_CREATE_PROCESS
Create Queue	MQCMD_CREATE_Q
Delete Channel	MQCMD_DELETE_CHANNEL
Delete Process	MQCMD_DELETE_PROCESS
Delete Queue	MQCMD_DELETE_Q
Escape	MQCMD_ESCAPE
Inquire Channel	MQCMD_INQUIRE_CHANNEL
Inquire Channel Names	MQCMD_INQUIRE_CHANNEL_NAMES
Inquire Channel Status	MQCMD_INQUIRE_CHANNEL_STATUS
Inquire Process	MQCMD_INQUIRE_PROCESS
Inquire Process Names	MQCMD_INQUIRE_PROCESS_NAMES
Inquire Queue	MQCMD_INQUIRE_Q
Inquire Queue Manager	MQCMD_INQUIRE_Q_MGR
Inquire Queue Names	MQCMD_INQUIRE_Q_NAMES
Ping Channel	MQCMD_PING_CHANNEL
Ping Queue Manager	MQCMD_PING_Q_MGR
Reset Channel	MQCMD_RESET_CHANNEL
Reset Queue Statistics	MQCMD_RESET_Q_STATS
Resolve Channel	MQCMD_RESOLVE_CHANNEL
Start Channel	MQCMD_START_CHANNEL
Stop Channel	MQCMD_STOP_CHANNEL

PCF command responses

In MQSeries for Tandem NSK, the command server generates standard response messages to each PCF command. There are three types of response:

- OK response
- Error response
- Data response

For more information, see the *MQSeries Programmable System Management* book.

Installable services

MQSeries for Tandem NSK supports the authorization service and the name service.

Authorization service interface

The authorization service enables queue managers to invoke authorization facilities. For example, a queue manager can check that a particular user ID is authorized to open a queue using the authorization service.

An authorization service component is supplied with MQSeries for Tandem NSK. This component is called the Object Authority Manager (OAM). By default, the OAM is active and works with the control commands **dspmqaut** (display authority) and **setmqaut** (set authority).

You can augment or replace the OAM with your own authorization service component, as described in the *MQSeries Programmable System Management* book.

Name service interface

The name service provides support to the queue manager for resolving the name of the queue manager that owns a queue.

The standard name service interface, as described in the *MQSeries Programmable System Management* book, is supported by MQSeries for Tandem NSK.

Appendix M. EMS event template used by MQSeries for Tandem NSK

The EMS template file (SMQSTMPL) contains the source code for the definitions of MQSeries EMS events. These definitions control how the information in the EMS event messages is displayed, and also show the type and meaning of the data contained in each EMS Event message.

The following types of event are generated:

ZMQS-VAL-EVT-ERROR

An FFST (a system resource problem, a software problem, or a hardware problem).

ZMQS-VAL-EVT-ERR

An error with MQSeries, referencing an FFST event and logged data on disk.

ZMQS-VAL-EVT-MSG

An MQSeries message, such as the starting of a queue manager or channel. All of these events correspond to an MQSeries AMQxxxxx log message and contain the same information and text. The variable data in each message is contained in individual tokens within the event message. For more information about the AMQxxxxx messages, see Appendix N, "Messages" on page 363.

ZMQS-VAL-EVT-QMGR

A queue manager event for authority, inhibit, local, remote, start, and stop events. These EMS events have effectively the same information content as their corresponding PCF event messages, which are described in the *MQSeries Programmable System Management* book. Individual tokens in the event message contain the variable data in each event message.

ZMQS-VAL-EVT-PERF

A performance event, corresponding with the standard MQSeries performance events. These events report statistical data about queues within a queue manager. The variable data in performance events is contained in individual tokens within the event message.

ZMQS-VAL-EVT-CHNL

A channel event, corresponding with the standard MQSeries channel events. Channel events report changes in status of channels, or problems in communication between queue managers. As with the other event message types, the variable data in channel events is contained in individual tokens within the event message.

EMS event template

Here is an extract from the definitions of the EMS templates:

```
VERSION: "IBM.MQS - 10JAN97"
SSID: ZMQS-VAL-SSID
SSNAME: "MQSeries", "MQS"

==
== This is an EMS FFST message
==
MSG: ZEMS-TKN-EVENTNUMBER, ZMQS-VAL-EVT-ERROR
    OVERRIDE ZEMS-TKN-EMPHASIS ZSPI-VAL-TRUE
    "MQSeries FFST from component COMP_<1>"
    "<*CR> Error Code : <2>"
    "<*CR> Severity : <3>"
    "<*CR> Module Name : <4>"
    "<*CR> Probe ID : <5>"
    "<*CR> Error Text : "
    "<*CR> <6>"
    1: ZMQS-TKN-COMPONENT
    2: ZMQS-TKN-ERROR-CODE

    3: ZMQS-TKN-SEVERITY
    4: ZMQS-TKN-MODULE-NAME
    5: ZMQS-TKN-PROBE-ID
    6: ZMQS-TKN-ERROR-TEXT

==
== This is an EMS Display Message Event
==
MSG: ZEMS-TKN-EVENTNUMBER, ZMQS-VAL-EVT-MSG
    "MQSeries message: <1>"
    "
    "<*CR> EXPLANATION :
        "<*CR> <2>"
    "
    "<*CR> ACTION :
        "<*CR> <3>"
    1: ZMQS-TKN-ERROR-TEXT
    2: ZMQS-TKN-ERROR-TEXT-2
    3: ZMQS-TKN-ERROR-TEXT-3

==
== This is an EMS Report Error Event
==
MSG: ZEMS-TKN-EVENTNUMBER, ZMQS-VAL-EVT-ERR
    OVERRIDE ZEMS-TKN-EMPHASIS ZSPI-VAL-TRUE
    "MQSeries Error"
    "<*CR> Error Code : <1>"
    "<*CR> Function : <2>"
    "<*CR> Probe ID : <3>"
    "<*CR> FFST File : <4>"
    1: ZMQS-TKN-ERROR-CODE
    2: ZMQS-TKN-MODULE-NAME
    3: ZMQS-TKN-PROBE-ID
    4: ZMQS-TKN-FILE-NAME
```

```

==
== This is an EMS copy of PCF Queue Manager event message
== for authority, inhibit, local, remote, start_and_stop events
==
MSG: ZEMS-TKN-EVENTNUMBER, ZMQS-VAL-EVT-QMGR
"MQSeries QMgr Event from <1>                                "
"<*CR> Reason : <2>                                         "
"<*IF 3><*CR> Reason Qualifier : <4>                         <*ENDIF>"
"<*IF 5><*CR> User ID : <6>                                   <*ENDIF>"
"<*IF 13><*CR> Object QMgr : <14>                             <*ENDIF>"
"<*IF 9><*CR> Options : <10>                                  <*ENDIF>"
"<*IF 11><*CR> Command : <12>                                 <*ENDIF>"
"<*IF 15><*CR> Queue Name : <16>                             <*ENDIF>"
"<*IF 17><*CR> Queue Type : <18>                             <*ENDIF>"
"<*IF 19><*CR> Base Queue Name : <20>                       <*ENDIF>"
"<*IF 21><*CR> XMit Queue Name : <22>                       <*ENDIF>"
"<*IF 30><*CR> Application Type : <31>                      <*ENDIF>"
"<*IF 32><*CR> Application Name : <33>                      <*ENDIF>"
1: ZMQS-TKN-QMGR
2: ZMQS-TKN-REASON
3: TOKENPRESENT(ZMQS-TKN-REASON-QUALIFIER)
4: ZMQS-TKN-REASON-QUALIFIER
5: TOKENPRESENT(ZMQS-TKN-USER-ID)
6: ZMQS-TKN-USER-ID
9: TOKENPRESENT(ZMQS-TKN-OPTIONS)
10: ZMQS-TKN-OPTIONS
11: TOKENPRESENT(ZMQS-TKN-COMMAND)
12: ZMQS-TKN-COMMAND
13: TOKENPRESENT(ZMQS-TKN-OBJ-QMGR)
14: ZMQS-TKN-OBJ-QMGR
15: TOKENPRESENT(ZMQS-TKN-Q-NAME)
16: ZMQS-TKN-Q-NAME
17: TOKENPRESENT(ZMQS-TKN-Q-TYPE)
18: ZMQS-TKN-Q-TYPE
19: TOKENPRESENT(ZMQS-TKN-BASE-Q-NAME)
20: ZMQS-TKN-BASE-Q-NAME
21: TOKENPRESENT(ZMQS-TKN-XMIT-Q-NAME)
22: ZMQS-TKN-XMIT-Q-NAME
30: TOKENPRESENT(ZMQS-TKN-APPL-TYPE)
31: ZMQS-TKN-APPL-TYPE
32: TOKENPRESENT(ZMQS-TKN-APPL-NAME)
33: ZMQS-TKN-APPL-NAME

```

```

==
== This is an EMS copy of PCF Performance event message
==
MSG: ZEMS-TKN-EVENTNUMBER, ZMQS-VAL-EVT-PERF
"MQSeries Performance Event from <1>                                "
"<*CR> Reason : <2>                                         "
"<*CR> Queue Name : <3>                                     "
"<*CR> Time Since Last Reset : <4>                         "
"<*CR> Highest Queue Depth : <5>                           "
"<*CR> # Of Messages Enqueued : <6>                       "
"<*CR> # Of Messages Dequeued : <7>                       "
1: ZMQS-TKN-QMGR
2: ZMQS-TKN-REASON

```

EMS event template

3: ZMQS-TKN-Q-NAME
4: ZMQS-TKN-TIME-SINCE-RESET
5: ZMQS-TKN-HIGH-Q-DEPTH
6: ZMQS-TKN-MSG-ENQ-COUNT
7: ZMQS-TKN-MSG-DEQ-COUNT

==

== This is an EMS copy of PCF Channel event message

==

MSG: ZEMS-TKN-EVENTNUMBER, ZMQS-VAL-EVT-CHNL

```
"MQSeries Channel Event from <1>          "
"<*CR> Reason : <2>                        "
"<*CR> Channel Name : <3>                   "
"<*CR> Xmit Queue Name : <5>                "
"<*CR> Connection Name : <7>                "
"<*CR> Reason Qualifier : <9>               "
"<*CR> Format : <11>                         "
"<*CR> Return Code : <13>                   "
"<*CR> Auxiliary rc 1 : <15>                 "
"<*CR> Auxiliary rc 2 : <17>                 "
"<*CR> CCSID 1 : <19>                       "
"<*CR> Auxiliary string 1 : <21>            "
"<*CR> CCSID 2 : <23>                       "
"<*CR> Auxiliary string 2 : <25>            "
"<*CR> CCSID 3 : <27>                       "
"<*CR> Auxiliary string 3 : <29>            "
```

1: ZMQS-TKN-QMGR
2: ZMQS-TKN-REASON
3: ZMQS-TKN-CHANNEL-NAME
5: ZMQS-TKN-XMIT-Q-NAME
7: ZMQS-TKN-CONN-NAME
9: ZMQS-TKN-REASON-QUALIFIER
11: ZMQS-TKN-FORMAT
13: ZMQS-TKN-RETURN-CODE
15: ZMQS-TKN-RETURN-CODE-2
17: ZMQS-TKN-RETURN-CODE-3
19: ZMQS-TKN-CCSID
21: ZMQS-TKN-ERROR-TEXT
23: ZMQS-TKN-CCSID-2
25: ZMQS-TKN-ERROR-TEXT-2
27: ZMQS-TKN-CCSID-3
29: ZMQS-TKN-ERROR-TEXT-3

Appendix N. Messages

This appendix describes the content and format of the messages issued by MQSeries for Tandem NSK.

Message format

The format of the MQSeries messages is as follows:

- The message identifier, where the identifier has two components:
 1. The characters "AMQ," which identify the message as originating from MQSeries
 2. A four-digit decimal code
- Text of the message

Structure of messages

This section describes the structure of MQSeries messages.

Message variables

Some messages display text or numbers that vary according to the circumstances giving rise to the message; these are known as *message variables*.

In this book, the message variables are shown as an '&' symbol, followed by a number.

Where there is more than one variable in a message, a different number is added to each '&' symbol.

Note: You should always look at the extended help for a message before carrying out any other action, because, in certain cases, the variables are displayed in the extended help only.

Message information

Where applicable, this information is also provided:

Explanation: Why the message was issued.

User action: Instructions to the user.

Note: The message file may contain the explanation of the message, in addition to the message itself.

MQSeries messages

MQSeries messages are numbered 5000 through 9999, and they are listed in this book in numeric order. However, not all numbers have been used, and therefore, the list is not continuous.

Message groups

MQSeries messages are grouped according to the part of MQSeries from which they originate:

5000 through 5999	Installable services - see page "Installable services messages" on page 365.
6000 through 6999	Common services - see page "Common services messages" on page 371.
7000 through 7999	The MQSeries product - see page "MQSeries product messages" on page 374.
8000 through 8999	Administering MQSeries - see page "Administration messages" on page 384.
9000 through 9999	Remote - see page "Remote messages" on page 405.

Installable services messages

AMQ5006 Unexpected error: rc = &1

Explanation:

An unexpected error occurred in an internal function of the product.

User action:

Save the generated output files and contact your IBM support center.

AMQ5501 There was not enough storage to satisfy the request

Explanation:

An internal function of the product attempted to obtain storage, but there was none available.

User action:

Stop the product and restart it. If this does not resolve the problem, save the generated output files and contact your IBM support center.

AMQ5511 Installable service component '&3' returned '&4'.

Explanation:

The internal function, that adds a component to a service, called the component initialization process. This process returned an error.

User action:

Check the component was installed correctly. If it was, and the component was supplied by IBM, then save the generated output files and contact your IBM support center. If the component was not supplied by IBM, save the generated output files and follow the support procedure for that component.

AMQ5512 Installable service component '&3' returned '&4' for queue manager name = '&5'.

Explanation:

An installable service component returned an unexpected return code.

User action:

Check the component was installed correctly. If it was, and the component was supplied by IBM, then save the generated output files and contact your IBM support center. If the component was not supplied by IBM, save the generated output files and follow the support procedure for that component.

AMQ5513 '&3' returned &1.

Explanation:

An unexpected error occurred.

User action:

Save the generated output files and contact your IBM support center.

AMQ5600 Usage: crtmqm [-z] [-q] [-c Text] [-d DefXmitQ] [-h MaxHandles]

User action:

None.

AMQ5601 [-t TrigInt] [-u DeadQ][-x MaxUMsgs] [-m Mlni] [-l CCSID]

User action:

None.

AMQ5602

[-e NumECs] [-p QMVol] -n PMonProc -o HomeTerm

User action:

None.

AMQ5603 Usage: dlrmqm [-z] QMgrName

Explanation:

See Explanation of message AMQ5700.

User action:

None

AMQ5604 Usage: dspmqaut [-m QMgrName] [-n ObjName] -t ObjType [-g Group] [-p Principal] [-s ServiceName]

User action:

None.

AMQ5605 Usage: endmqm [-z] [-c | -i | -p] QMgrName

Explanation:

See Explanation of message AMQ5700.

User action:

None.

AMQ5606 Usage: setmqaut -m QMgrName [-n ObjName] -t ObjType [-g Group] [-p Principal] [-s ServiceName] Authorizations

User action:

None.

AMQ5607 Usage: strmqm [-z] [QMgrName]

Explanation:

See Explanation of message AMQ5700.

User action:

None.

AMQ5608 Usage: dspmqtrn QMgrName

Explanation:

See Explanation of message AMQ5700.

User action:

None.

AMQ5609 Usage: rsvmqtrn -m QMgrName (-c | -b) Transaction,Number

Explanation:

See Explanation of message AMQ5700.

User action:

None.

AMQ5610 Usage: strmqtrc [-m QMgrName] [-t TraceType]

Explanation:

See Explanation of message AMQ5700.

User action:

None.

AMQ5611 Usage: endmqtrc [-m QMgrName] [-a]

Explanation:

See Explanation of message AMQ5700.

User action:

None.

AMQ5612 Usage: dspmqtrc [-t TemplateFile] InputFileName

Explanation:

See Explanation of message AMQ5700.

User action:

None.

AMQ5614 Usage: cleanrdf -b \BkpSysName [-m QMgrName]

User action:

None.

AMQ5615 -s MQSSProc QMgrName

User action:

None.

AMQ5616 Usage: altmqusr [-m QMgrName] -p Principal [-u UserName] [-r]

User action:

None.

AMQ5617 Principal LogonId LogonName
alias GroupName GroupType

User action:

None.

AMQ5618 The Principal name was specified incorrectly.
Explanation:

The specified Principal name does not conform to the rules required by MQSeries.

User action:

Correct the name and submit the command again.

AMQ5619 Error modifying an entry in the Principal database.

Explanation:

MQSeries was unable to update or delete the specified entry in the Principal database.

User action:

Make sure that the entry for this Principal exists and submit the command again.

AMQ5620 Usage: dspmqusr [-m QMgrName] [-p Principal]

User action:

None.

AMQ5621 The Tandem User name was specified incorrectly.

Explanation:

The specified Tandem User name does not conform to the rules required by MQSeries.

User action:

Correct the name and submit the command again.

AMQ5700 Queue manager name '&3', work queue name '&4'.

Explanation:

These are the values of the parameters with which the add-in task was started.

User action:

None.

Programmer response:

None.

AMQ5701 Checking mail-in database &3

Explanation:

The add-in task is performing a periodic check for mail memos that have arrived in the mail-in database called &3.

User action:

None.

Programmer response:

None.

AMQ5702 Checking for replies.

Explanation:

The add-in task is checking the reply queues for responses from MQSeries applications.

User action:

Programmer response:

None.

AMQ5703 MQSeries add-in task ended.

Explanation:

Termination of the MQSeries add-in task has completed.

User action:

None.

Programmer response:

None.

AMQ5704 Terminating.**Explanation:**

The add-in task is terminating, either due to a user request or an error.

User action:

None.

Programmer response:

None.

AMQ5705 Initializing.**Explanation:**

The add-in task is initializing. It processes the link database and connects to the queue manager in preparation to receive requests.

User action:

None.

Programmer response:

None.

AMQ5706 Mail-in database '&3', link database '&4', wait time &1 seconds.**Explanation:**

These are the values of the parameters with which the add-in task was started.

User action:

None.

Programmer response:

None.

AMQ5707 Add-in task initialization complete.**Explanation:**

The add-in task has finished reading the link database and is now ready to process requests.

User action:

None.

Programmer response:

None.

AMQ5708 Only two-byte integer values are supported for S390 format.**Explanation:**

The add-in task supports conversion of two-byte integers from S390 systems.

User action:

Ensure that the entry in the link database uses fields of only two bytes in length if they are in the S390 format.

Programmer response:

None.

AMQ5710 Text of user document causing previous message: '&3'.**Explanation:**

The add-in task generated the previous message in response to an error. This message contains the text of the user note associated with the error.

User action:

None.

Programmer response:

None.

AMQ5711 An error occurred in reading the link database.**Explanation:**

The add-in task detected an error while reading the link database.

User action:

Use the information in previous error messages to diagnose the error. Then, correct the contents of the link database and restart the add-in task.

Programmer response:

None.

AMQ5712 An error occurred while setting field '&5' in user document, return code &3**Explanation:**

The add-in task was trying to update a document in response to a reply from an MQSeries application. An error was encountered during the update of the field '&5'. The link database entry '&4' was being used to perform the update.

User action:

Make sure that the entry in the link database matches the description of the form being used for the update.

Programmer response:

None.

AMQ5714 Field '&4' not found in link database entry.**Explanation:**

The add-in task could not find a field called '&4' during processing of the link database. This field is a required field.

User action:

Examine the definition of the link database being used to ensure that all of the required fields are supplied. Refer to the IBM-supplied sample link database for an example of a valid link database.

Programmer response:

None.

AMQ5715 Data type '&4' not supported.**Explanation:**

The add-in task does not support the data type '&4'.

User action:

Consult the MQSeries documentation for a description of the list of supported data types. Update the entry in the link database using the unsupported data type. Then, stop and restart the add-in task.

Programmer response:

None.

AMQ5716 An error occurred connecting to MQSeries queue manager '&4', reason code &3**Explanation:**

The add-in task could not connect to MQSeries queue manager '&4'. The reason code from MQCONN was &3.

User action:

Look up the reason code in the MQSeries documentation to establish the cause of the error. Ensure that the queue manager exists and is running. If the add-in task is running as an MQSeries client, ensure that it can communicate with the server queue manager.

Programmer response:

None.

AMQ5717 An error occurred disconnecting from MQSeries queue manager '&4', return code '&3'.**Explanation:**

The add-in task encountered an error disconnecting from the MQSeries queue manager '&4'. The reason code from MQDISC was &3.

User action:

Look up the reason code in the MQSeries documentation to establish the cause of the error.

Programmer response:

None.

AMQ5718 An error occurred during processing of a request in the mail-in database.**Explanation:**

The add-in task encountered an error during processing of a request in the mail-in database. The processing involves transforming the contents of the mail memo into a message which is placed on an MQSeries queue. If the message has a reply, an additional message is formatted and placed on the internal work queue.

User action:

Use the information in previous error messages to diagnose the error.

Programmer response:

None.

AMQ5720 Errors detected in response message from MQSeries application.**Explanation:**

The response from an MQSeries application to a message sent by the add-in task satisfied the error conditions specified in the corresponding link database entry. The error data is '&4'.

User action:

Examine the error conditions in the link database entry to establish why the error conditions were satisfied. If an invalid request message was sent to the MQSeries application, correct the request messages being sent. If the problem was due to an error encountered by the MQSeries application, correct the cause of the error and retry the request.

Programmer response:

None.

AMQ5721 An error occurred opening internal work file '&4'.**Explanation:**

The add-in task could not open the internal work file used to hold the contents of a mail memo during processing. Possible causes include more than one program trying to use the same file.

User action:

Ensure that there is only one copy of the MQSeries add-in task running.

Programmer response:

None.

AMQ5723 Memory allocation failed.**Explanation:**

The add-in task was unable to allocate storage.

User action:

Try to free up some system memory and retry the operation.

Programmer response:

None.

AMQ5725 Empty mail memo received from mail-in database.**Explanation:**

The add-in task found a mail memo with an empty body in the mail-in database. Mail memos in the mail-in database must contain the information required to generate a message to place on an MQSeries queue.

User action:

Ensure that all entries placed in the mail-in database have the expected contents. None.

Programmer response:

None.

AMQ5727 Link database entry '&4' cannot be found.**Explanation:**

The add-in task received a request without a corresponding entry in the link database. The name of the required entry is '&4'.

User action:

Either add an entry of the correct name to the link database or change the request being generated to use an existing entry in the link database. If you add an entry to the link database, you will have to stop and restart the add-in task before the change takes effect.

Programmer response:

None.

AMQ5729 An error was encountered by the add-in task. Check the mail for details.**Explanation:**

This message is inserted into the error_field_msg field of a user document if an error is encountered by the add-in task during the processing of the document's associated mail memo.

User action:

None.

Programmer response:

None.

AMQ5730 Error encountered by MQSeries add-in task

Explanation:

This is the subject line of mail memos sent by the add-in task.

User action:

None.

Programmer response:

None.

AMQ5731 Idle.

Explanation:

The add-in task is waiting for the configured time interval to elapse before checking the mail-in database for new requests and checking the reply queues for new replies.

User action:

None.

Programmer response:

None.

AMQ5732 LOAD MQLINK -t -q WorkQName -w WaitTime -d MailInDB -l LinkDB QMgrName

Explanation:

This is a summary of the correct syntax for invoking the MQSeries add-in task in Lotus Notes. If you specify a queue manager name, it must be the last parameter. The order of the other parameters is not significant.

User action:

None.

Programmer response:

None.

AMQ5733 MQSeries add-in task loading.

Explanation:

The add-in task has been started and is accessing the link database in preparation to receive requests.

User action:

None.

Programmer response:

None.

AMQ5734 An error occurred opening the database '&4'. The error code was &3.

Explanation:

The add-in task could not open the named database. This could be because the database does not exist.

User action:

Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response:

None.

AMQ5735 An error occurred opening the mail file '&4'. The error code was &3.

Explanation:

The add-in task could not open the named mail file.

User action:

Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response:

None.

AMQ5736 An error occurred searching the database '&4'. The error code was &3.

Explanation:

The add-in task could not search the named database.

User action:

Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response:

None.

AMQ5737 An error occurred deleting an entry from the database '&4'. The error code was &3.

Explanation:

The add-in task could not delete an entry from the named database.

User action:

Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response:

None.

AMQ5738 An error occurred extracting the contents of a mail memo in the mail-in database '&5' to the file called '&4'. The error code was &3.

Explanation:

The add-in task could not extract the body of a mail memo into the named file. Possible causes include being unable to create the file or another program already using the file.

User action:

Ensure that there is only one copy of the MQSeries add-in task running. If the problem was due to the configuration in which you are operating Lotus Notes, refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response:

None.

AMQ5739 An error occurred opening a mail memo in the mail-in database '&4'. The error code was &3.

Explanation:

The add-in task could not open a mail memo in the named mail-in database.

User action:

Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response:

None.

AMQ5740 An error occurred opening an entry in the link database '&4'. The error code was &3.

Explanation:

The add-in task could not open an entry in the link database.

User action:

Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response:

None.

AMQ5741 An error occurred creating a mail memo. The error code was &3.

Explanation:

The add-in task could not create a mail memo. This is probably due to a shortage of resources.

User action:

Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response:

None.

AMQ5742 Could not send a mail memo to user '&4'. The error code was &3.

Explanation:

The add-in task could not send a mail memo to the named user to report an error condition.

User action:

Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response:

None.

AMQ5743 Could not find entry with ID '&5' in database '&4'. The error code was &3.

Explanation:

The add-in task could not find an entry in the database '&4' which it was to update in response to a reply from an MQSeries application. This may indicate that the entry has been manually deleted or that another application has already updated the entry.

User action:

Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response:

None.

AMQ5744 Could not update an entry in database '&4'. The error code was &3.

Explanation:

The add-in task could not update an entry in the database '&4' in response to a reply from an MQSeries application.

User action:

Refer to the Lotus Notes documentation for information to resolve the problem.

Programmer response:

None.

AMQ5745 An error occurred opening MQSeries queue '&4', reason code &3.

Explanation:

The add-in task could not open MQSeries queue '&4'. MQOPEN was called with open options &5. The reason code from MQOPEN was &3.

User action:

Look up the reason code in the MQSeries documentation to establish the cause of the error.

Programmer response:

None.

AMQ5746 An error occurred putting a message on MQSeries queue '&4', reason code &3.

Explanation:

The add-in task could not put a message on MQSeries queue '&4'. The reason code from MQPUT was &3.

User action:

Look up the reason code in the MQSeries documentation to establish the cause of the error.

Programmer response:

None.

AMQ5747 An error occurred getting a message from MQSeries queue '&4', reason code &3

Explanation:

The add-in task could not get a message from MQSeries queue '&4'. The reason code from MQGET was &3.

User action:

Look up the reason code in the MQSeries documentation to establish the cause of the error.

Programmer response:

None.

Common services messages

AMQ6004 An error occurred during MQSeries initialization or ending.

Explanation:

An error was detected during initialization or ending of MQSeries. The MQSeries error recording routine has been called.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6025 Program not found.

Explanation:

MQSeries is unable to start program &3 because it was not found.

User action:

Check the program name is correctly specified and rerun the program.

AMQ6026 A resource shortage prevented the creation of an MQSeries process.

Explanation:

An attempt to create an MQSeries process was rejected by the operating system due to a process limit (either the number of processes for each user or the total number of processes running system wide), or because the system does not have the resources necessary to create another process.

User action:

Investigate if a process limit is preventing the creation of the process and if so why the system is constrained in this way. Consider raising this limit or reducing the workload on the system.

AMQ6035 MQSeries failed, no storage available.

Explanation:

An internal function of the product attempted to obtain storage, but there was none available.

User action:

Stop the product and restart it. If this does not resolve the problem, save the generated output files and contact your IBM support center.

AMQ6037 MQSeries was unable to obtain enough storage.

Explanation:

The product is unable to obtain enough storage. The product's error recording routine may have been called.

User action:

Stop the product and restart it. If this does not resolve the problem see if a problem has been recorded. If a problem has been recorded, use the standard facilities supplied with your system to record the problem identifier, and to save the

generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6047 Conversion not supported.

Explanation:

MQSeries is unable to convert string data tagged in CCSID &1 to data in CCSID &2.

User action:

Check the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6048 DBCS error

Explanation:

MQSeries is unable to convert string data due to a DBCS error. Conversion is from CCSID &1 to CCSID &2.

User action:

Check the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6049 DBCS only string not valid.

Explanation:

MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2. Message descriptor data must be in single byte form. CCSID &2 is a DBCS only CCSID.

User action:

Check the CCSID of your job or system and change it to one supporting SBCS or mixed character sets. Refer to the appropriate National Language Support publications for character sets and CCSIDs supported.

AMQ6050 CCSID error.

Explanation:

MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2.

User action:

Check the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6051 Conversion length error.

Explanation:

MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2, due to an input length error.

User action:

None.

AMQ6052 Conversion length error.

Explanation:

MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2.

User action:

None.

AMQ6053 CCSID error

Explanation:

MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2.

User action:

One of the CCSIDs is not supported by the system. Check the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6064 An internal MQSeries error has occurred.

Explanation:

An error has been detected, and the MQSeries error recording routine has been called.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6090 MQSeries was unable to display an error message.

Explanation:

MQSeries has attempted to display the message associated with return code &6. The return code indicates that there is no message text associated with the message. Associated with the request are inserts &1 : &2 : &3 : &4 : &5.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6091 An internal MQSeries error has occurred.

Explanation:

Private memory has detected an error, and is abending due to &3. The error data is &1.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6100 An internal MQSeries error has occurred.

Explanation:

MQSeries has detected an error, and is abending due to &3. The error data is &1.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6107 CCSID not supported.

Explanation:

MQSeries is unable to convert string data in CCSID &1 to data in CCSID &2, because one of the CCSIDs is not recognized.

User action:

Check the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6115 An internal MQSeries error has occurred.

Explanation:

An error has been detected, and the MQSeries error recording routine has been called.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6118 An internal MQSeries error has occurred.

Explanation:

An error has been detected, and the MQSeries error recording routine has been called.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6119 An internal MQSeries error has occurred.

Explanation:

MQSeries detected an unexpected error when calling the operating system. The MQSeries error recording routine has been called.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6120 An internal MQSeries error has occurred.

Explanation:

An error has been detected, and the MQSeries error recording routine has been called.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6121 An internal MQSeries error has occurred.

Explanation:

An error has been detected, and the MQSeries error recording routine has been called.

User action:

MQSeries has detected a parameter count of &1 that is not valid. Use the standard facilities supplied with your system to record the problem identifier, and to save the generated

output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6122 An internal MQSeries error has occurred.

Explanation:

An error has been detected, and the MQSeries error recording routine has been called.

User action:

MQSeries has detected parameter &1 that is not valid, having value &2&3. Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6125 An internal MQSeries error has occurred.

Explanation:

An internal error has occurred with identifier &1. This message is issued in association with other messages.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6148 An internal MQSeries error has occurred.

Explanation:

MQSeries has detected an error, and is abending due to &3. The error data is &1.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6172 No codeset found for current locale.

Explanation:

No codeset could be determined for the current locale. Check that the locale in use is supported.

User action:

None.

AMQ6173 No CCSID found for codeset &3.

Explanation:

Codeset &3. has no supported CCSID. Check that the locale in use is supported. CCSIDs can be added by updating the file `/var/mqm/conv/table/ccsid.tbl`.

User action:

None.

AMQ6708 A disk full condition was encountered when formatting a new log file in location &3.

Explanation:

The queue manager attempted to format a new log file in directory &3. The drive or file system containing this directory did not have sufficient free space to contain the new log file.

User action:

Increase the amount of space available for log files and retry the request.

AMQ6710 Queue manager unable to access directory &3.

Explanation:

The queue manager was unable to access directory &3 for the log. This could be because the directory does not exist, or because the queue manager does not have sufficient authority.

User action:

Ensure that the directory exists and that the queue manager has authority to read and write to it. Ensure that the LogPath attribute in the queue manager's configuration file matches the intended log path.

AMQ6767 Log file &3 could not be opened for use.

Explanation:

Log file &3 could not be opened for use. Possible reasons include the file being missing, the queue manager being denied permission to open the file or the contents of the file being incorrect.

User action:

If the log file was required to start the queue manager, ensure that the log file exists and that the queue manager is able to read from and write to it. If the log file was required to recreate an object from its media image and you do not have a copy of the required log file, delete the object instead of recreating it.

MQSeries product messages

AMQ7001 The location specified for creation of the queue manager is not valid.

Explanation:

The directory under which queue managers are to be created is not valid. It may not exist, or there may be a problem with authorization.

User action:

The location is specified in the machine-wide ini file. Correct the file and submit the request again.

AMQ7002 An error occurred manipulating a file.

Explanation:

An internal error occurred while trying to create or delete a queue manager file. It is likely that the error was caused by there being insufficient space on a disk, or by problems with authorization to the underlying filesystem.

User action:

Identify the file that caused the error, using problem determination techniques. Correct the error in the filesystem and submit the request again.

AMQ7005 The queue manager is running.

Explanation:

You tried to perform an action that requires the queue manager stopped, however, it is currently running. You probably tried to delete or start a queue manager that is currently running.

User action:

If the queue manager should be stopped, stop the queue manager and submit the failed command again.

AMQ7006 Missing attribute &5 on stanza starting on line &1 of ini file &3.

Explanation:

The &4 stanza starting on line &1 of configuration file &3 is missing the required &5 attribute.

User action:

Check the contents of the file and retry the operation.

AMQ7008 The queue manager already exists.

Explanation:

You tried to create a queue manager that already exists.

User action:

If you specified the wrong queue manager name, correct the name and submit the request again.

AMQ7010 The queue manager does not exist.

Explanation:

You tried to perform an action against a queue manager that does not exist. You may have specified the wrong queue manager name.

User action:

If you specified the wrong name, correct it and submit the command again. If the queue manager should exist, create it, and then submit the command again.

AMQ7012 The specified trigger interval is not valid.

Explanation:

You specified a value for the trigger interval that is not valid. The value must be not less than zero and not greater than 999 999 999.

User action:

Correct the value and resubmit the request.

AMQ7013 There is an error in the name of the specified dead letter queue.

Explanation:

You specified a name for the dead letter queue that is not valid.

User action:

Correct the name and resubmit the request.

AMQ7014 There is an error in the name of the specified default transmission queue.

Explanation:

You specified a name for the default transmission queue that is not valid.

User action:

Correct the name and submit the command again.

AMQ7015 There is an error in the maximum number of open object handles specified.

Explanation:

You specified a value for the maximum number of open object handles to be allowed that is not valid. The value must be not less than zero and not greater than 999 999 999.

User action:

Correct the value and submit the command again.

AMQ7016 There is an error in the maximum number of uncommitted messages specified.

Explanation:

You specified a value for the maximum number of uncommitted messages to be allowed that is not valid. The value must be not less than 1 and not greater than 999 999 999.

User action:

Correct the value and submit the command again.

AMQ7017 Log not available.

Explanation:

The queue manager was unable to use the log. This could be due to a log file being missing or damaged, or the log path to the queue manager being inaccessible.

User action:

Ensure that the LogPath attribute in the queue manager configuration file is correct. If a log file is missing or otherwise unusable, restore a backup copy of the file, or the entire queue manager.

AMQ7018 The queue manager has stopped**Explanation:**

See Explanation of message AMQ7019

User action:

See User action for message AMQ7019

AMQ7019 An error occurred while creating the directory structure for the new queue manager.**Explanation:**

During creation of the queue manager an error occurred while trying to create a file or directory.

User action:

Identify why the queue manager files cannot be created. It is probable that there is insufficient space on the specified disk, or that there is a problem with access control. Correct the problem and submit the command again.

AMQ7021 An error occurred while deleting the directory structure for the queue manager.**Explanation:**

While deleting the queue manager, an error occurred deleting a file or directory. The queue manager may not have been completely deleted.

User action:

Follow problem determination procedures to identify the file or directory and to complete deletion of the queue manager.

AMQ7024 Arguments supplied to a command are not valid.**Explanation:**

You supplied arguments to a command that it could not interpret. It is probable that you specified a flag not accepted by the command, or that you included extra flags.

User action:

Correct the command and submit it again.

AMQ7025 Error in the supplied command description.**Explanation:**

The descriptive text you supplied on the command was in error.

User action:

Correct the descriptive text and submit the command again.

AMQ7026 A principal or group name was invalid.**Explanation:**

You specified the name of a principal or group which does not exist.

User action:

Correct the name and resubmit the request.

AMQ7028 The queue manager is not available for use.**Explanation:**

You have requested an action that requires the queue manager running, however, the queue manager is not currently running.

User action:

Start the required queue manager and submit the command again.

AMQ7030 Request to quiesce the queue manager accepted. The queue manager will stop when there is no further work for it to perform.**Explanation:**

You have requested that the queue manager end when there is no more work for it. In the meantime, it will refuse new applications that attempt to start, although it allows those already running to complete their work.

User action:

None.

AMQ7031 The queue manager is stopping.**Explanation:**

You issued a command that requires the queue manager running, however, it is currently in the process of stopping. The command cannot be run.

User action:

None

AMQ7041 Object already exists.**Explanation:**

A Define Object operation was performed, but the name selected for the object is already in use by an object that is unknown to MQSeries. The object name selected by MQSeries was &3, in directory &4, of object type &5.

User action:

Remove the conflicting object from the MQSeries system, then try the operation again.

AMQ7042 Media image not available for object &3 of type &4.**Explanation:**

The media image for object &3, type &4, is not available for media recovery. A log file containing part of the media image cannot be accessed.

User action:

A previous message indicates which log file could not be accessed. Restore a copy of the log file and all subsequent log files from backup. If this is not possible, you must delete the object instead.

AMQ7044 Media recovery not allowed.**Explanation:**

Media recovery is not possible on a queue manager using a circular log. Damaged objects must be deleted on such a queue manager.

User action:

None.

AMQ7047 An unexpected error was encountered by a command.**Explanation:**

An internal error occurred during the processing of a command.

User action:

Follow problem determination procedures to identify the cause of the error.

AMQ7048 The queue manager name is either not valid or not known

Explanation:

Either the specified queue manager name does not conform to the rules required by MQSeries or the queue manager does not exist. The rules for naming MQSeries objects are detailed in the MQSeries Command Reference.

User action:

Correct the name and submit the command again.

AMQ7053 The transaction has been committed.

Explanation:

The prepared transaction has been committed.

User action:

None.

AMQ7054 The transaction has been backed out.

Explanation:

The prepared transaction has been backed out.

User action:

None.

AMQ7055 The transaction number is not recognized.

Explanation:

The number of the transaction you supplied was not recognized as belonging to an in-doubt transaction.

User action:

Ensure that you entered a valid transaction number. It is possible that the transaction number you entered corresponds to a transaction which was committed or backed out before you issued the command to resolve it.

AMQ7056 Transaction number &1,&2.

Explanation:

This message is used to report the number of an in-doubt transaction.

User action:

None.

AMQ7064 Log path not valid or inaccessible.

Explanation:

The supplied log path could not be used by the queue manager. Possible reasons for this include the path not existing, the queue manager not being able to write to the path, or the path residing on a remote device.

User action:

Ensure that the log path exists and that the queue manager has authority to read and write to it. If the queue manager already exists, ensure that the LogPath attribute in the queue manager's configuration file matches the intended log path.

AMQ7065 Insufficient space on disk.

Explanation:

The operation cannot be completed due to shortage of disk space.

User action:

Either make more disk space available, or reduce the disk requirements of the command you issued.

AMQ7066 There are no prepared transactions.

Explanation:

There are no prepared transactions to be resolved.

User action:

None.

AMQ7068 Authority file contains an authority stanza that is not valid.

Explanation:

A syntax error has been found in one of the files containing authorization information for the queue manager.

User action:

Correct the contents of the incorrect authorization file by editing it.

AMQ7069 The queue manager was created successfully, but cannot be made the default.

Explanation:

The queue manager was defined to be the default queue manager for the machine when it was created. However, although the queue manager has been created, an error occurred trying to make it the default. There may not be a default queue manager defined for the machine at present.

User action:

There is probably a problem with the machine-wide ini file. Verify the existence of the file, its access permissions, and its contents. If its backup file exists, reconcile the contents of the two files and then delete the backup. Finally, either update the machine-wide ini file by hand to specify the desired default queue manager, or delete and recreate the queue manager.

AMQ7073 Log size not valid.

Explanation:

Either the number of log files or the size of the log files was outside the accepted values.

User action:

Make sure that the log parameters you enter lie within the valid range.

AMQ7074 Unknown stanza key &4 on line &1 of ini file &3.

Explanation:

Line &1 of the configuration file &3 contained a stanza called &4. This stanza is not recognized.

User action:

Check the contents of the file and retry the operation.

AMQ7075 Unknown attribute &4 on line &1 of ini file &3.**Explanation:**

Line &1 of the configuration file &3 contained an attribute called &4 that is not valid. This attribute is not recognized in this context.

User action:

Check the contents of the file and retry the operation.

AMQ7076 Value &5 not valid for attribute &4 on line &1 of ini file &3**Explanation:**

Line &1 of the configuration file &3 contained value &5 that is not valid for the attribute &4.

User action:

Check the contents of the file and retry the operation.

AMQ7077 You are not authorized to perform the requested operation.**Explanation:**

You tried to issue a command for the queue manager. You are not authorized to perform the command.

User action:

Contact your system administrator to perform the command for you. Alternatively, request authority to perform the command from your system administrator.

AMQ7080 No objects processed.**Explanation:**

No objects were processed, either because no objects matched the criteria given, or because the objects found did not require processing.

User action:

None.

AMQ7081 Object &3, type &4 recreated.**Explanation:**

The object &3, type &4 was recreated from its media image.

User action:

None.

AMQ7082 Object &3, type &4 is not damaged.**Explanation:**

Object &3, type &4 cannot be recreated since it is not damaged.

User action:

None

AMQ7083 A resource problem was encountered by a command.**Explanation:**

The command failed due to a resource problem. Possible causes include the log being full or the command running out of memory.

User action:

Look at the previous messages to diagnose the problem. Rectify the problem and retry the operation.

AMQ7084 Object &3, type &4 damaged.**Explanation:**

The object &3, type &4 was damaged. The object must be deleted or, if the queue manager supports media recovery, recreated from its media image.

User action:

Delete the object or recreate it from its media image.

AMQ7085 Object &3, type &4 not found.**Explanation:**

Object &3, type &4 cannot be found.

User action:

None.

AMQ7086 Media image for object &3, type &4 recorded.**Explanation:**

The media image for object &3, type &4 has been recorded.

User action:

None.

AMQ7087 Object &3, type &4 is a temporary object**Explanation:**

Object &3, type &4 is a temporary object. Media recovery operations are not permitted on temporary objects.

User action:

None.

AMQ7088 Object &3, type &4 in use.**Explanation:**

Object &3, type &4 is in use. Either an application has it open or, if it is a local queue, there are uncommitted messages on it.

User action:

Ensure that the object is not opened by any applications, and that there are no uncommitted messages on the object, if it is a local queue. Then, retry the operation.

AMQ7089 Media recovery already in progress.**Explanation:**

Another media recovery operation is already in progress. Only one media recovery operation is permitted at a time.

User action:

Wait for the existing media recovery operation to complete and retry the operation.

AMQ7090 The queue manager CCSID is not valid.**Explanation:**

The CCSID to be used by the QMGR is not valid, because:

1. It is a DBCS CCSID.
2. The CCSID encoding is not ASCII or ASCII related. EBCDIC or UCS2 encodings are not valid on this machine.
3. The CCSID encoding is unknown.

User action:

Check the CCSID is valid for the machine on which you are working.

Programmer response:

None.

AMQ7091 You are performing authorization for the queue manager, but you specified an object name.

Explanation:

Modification of authorizations for a queue manager can be performed only from that queue manager. You must not specify an object name.

User action:

Correct the command and submit it again.

AMQ7092 An object name is required but you did not specify one.

Explanation:

The command needs the name of an object, but you did not specify one.

User action:

Correct the command and submit it again.

AMQ7093 An object type is required but you did not specify one.

Explanation:

The command needs the type of the object, but you did not specify one.

User action:

Correct the command and submit it again.

AMQ7094 You specified an object type that is not valid, or more than one object type.

Explanation:

Either the type of object you specified was not valid, or you specified multiple object types on a command which supports only one.

User action:

Correct the command and submit it again.

AMQ7095 An entity name is required but you did not specify one.

Explanation:

The command needs one or more entity names, but you did not specify any. Entities can be principals or groups.

User action:

Correct the command and submit it again.

AMQ7096 An authorization specification is required but you did not provide one.

Explanation:

The command sets the authorizations on MQSeries objects. However you did not specify which authorizations are to be set.

User action:

Correct the command and submit it again.

AMQ7097 You gave an authorization specification that is not valid.

Explanation:

The authorization specification you provided to the command contained one or more items that could not be interpreted.

User action:

Correct the command and submit it again.

AMQ7098 The command accepts only one entity name. You specified more than one.

Explanation:

The command can accept only one principal or group name. You specified more than one.

User action:

Correct the command and submit it again.

AMQ7099 Entity &3 has the following authorizations for object &4:

Explanation:

Informational message. The list of authorizations follows.

User action:

None.

AMQ7305 Trigger message could not be put on an initiation queue.

Explanation:

The attempt to put a trigger message on queue &4 on queue manager &5 failed with reason code &1. The message will be put on the dead-letter queue.

User action:

Ensure that the initiation queue is available, and operational.

AMQ7306 The dead-letter queue must be a local queue.

Explanation:

An undelivered message has not been put on the dead-letter queue &4 on queue manager &5, because the queue is not a local queue. The message will be discarded.

User action:

Inform your system administrator.

AMQ7307 A message could not be put on the dead-letter queue.

Explanation:

The attempt to put a message on the undelivered-message queue &4 on queue manager &5 failed with reason code &1. The message will be discarded.

User action:

Ensure that the undelivered-message queue is available, and operational.

AMQ7308 Trigger condition &1 was not satisfied.**Explanation:**

At least one of the conditions required for generating a trigger message was not satisfied, so a trigger message was not generated. If you were expecting a trigger message, consult the MQSeries Application Programming Guide for a list of the conditions required. (Note that arranging for condition &1 to be satisfied might not be sufficient because the conditions are checked in an arbitrary order, and checking stops when the first unsatisfied condition is discovered.)

User action:

If a trigger message is required, ensure that all the conditions for generating one are satisfied.

AMQ7310 Report message could not be put on a reply-to queue.**Explanation:**

The attempt to put a report message on queue &4 on queue manager &5 failed with reason code &1. The message will be put on the undelivered-message queue.

User action:

Ensure that the reply-to queue is available, and operational.

AMQ7463 The log for queue manager &3 is full.**Explanation:**

This message is issued when an attempt to write a log record is rejected because the log is full. The queue manager will attempt to resolve the problem.

User action:

This situation may be encountered during a period of unusually high message traffic. However, if you persistently fill the log, you may have to consider enlarging the size of the log. You can either increase the number of log files by changing the values in the queue manager configuration file. You will then have to stop and restart the queue manager. Alternatively, if you need to make the log files themselves bigger, you will have to delete and recreate the queue manager.

AMQ7464 The log for queue manager &3 is no longer full.**Explanation:**

This message is issued when a log was previously full, but an attempt to write a log record has now been accepted. The log full situation has been resolved.

User action:

None

AMQ7465 The log for queue manager &3 is full. This is due to the presence of a long-running transaction.**Explanation:**

This message is issued when an attempt made to resolve a log full situation fails, because the space is occupied by a long-running transaction.

User action:

Try to ensure that the duration of your transactions is not excessive. Commit or roll back any old transactions to release log space for further log records.

AMQ7466 The log for queue manager &3 is too small to support the current data rate.**Explanation:**

This message is issued when the monitoring tasks maintaining the log cannot keep up with the current rate of data being written.

User action:

The number of primary log files configured should be increased to prevent possible log full situations.

AMQ7467 The oldest log file required to start queue manager &3 is &4.**Explanation:**

The log file &4 contains the oldest log record required to restart the queue manager. Log records older than this may be required for media recovery.

User action:

You can move log files older than &4 to an archive medium to release space in the log directory. If you move any of the log files required to recreate objects from their media images, you will have to restore them to recreate the objects.

AMQ7468 The oldest log file required to perform media recovery of queue manager &3 is &4.**Explanation:**

The log file &4 contains the oldest log record required to recreate any of the objects from their media images. Any log files prior to this will not be accessed by media recovery operations.

User action:

You can move log files older than &4 to an archive medium to release space in the log directory.

AMQ7469 Transactions rolled back to release log space.**Explanation:**

The log space for the queue manager is becoming full. One or more long-running transactions have been rolled back to release log space so that the queue manager can continue to process requests.

User action:

Try to ensure that the duration of your transactions is not excessive. You may consider increasing the size of the log to allow transactions to last longer before the log starts to become full.

AMQ7472 Object &3, type &4 damaged.**Explanation:**

Object &3, type &4 has been marked as damaged. This indicates that the queue manager was either unable to access the object in the file system, or that some kind of inconsistency with the data in the object was detected.

User action:

If a damaged object is detected, the action performed depends on whether the queue manager supports media recovery and when the damage was detected. If the queue manager does not support media recovery, you must delete the object as no recovery is possible. If the queue manager does support media recovery and the damage is detected during the processing performed when the queue manager is being started, the queue manager will automatically initiate media recovery of the object. If the queue manager supports

media recovery and the damage is detected once the queue manager has started, it may be recovered from a media image using the rcrmqobj command or it may be deleted.

AMQ7901 The data-conversion exit &3 has not loaded.

Explanation:

The data-conversion exit program, &3, failed to load. The internal function gave exception &4.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7902 The data conversion exit &3 was not loaded. The operating system call &4 returned &1.

Explanation:

User action:

Specify REPLACE to over-write the existing file, or choose a different output file name.

AMQ7903 The data-conversion exit &3 cannot be found.

Explanation:

Message data conversion has been requested for an MQSeries message with a user-defined format, but the necessary data-conversion exit program, &3, cannot be found. The internal function gave exception &4.

User action:

Check that the necessary data-conversion exit &3 exists.

AMQ7904 The data conversion exit &3 cannot be found, or loaded.

Explanation:

Message data conversion was requested for an MQSeries message with a user-defined format, but the necessary data conversion exit program, &3, was not found, or loaded. The &4 function call gave a return code of &1.

User action:

Check that the necessary data conversion exit routine exists one of the standard directories for dynamically loaded modules. If necessary, inspect the generated output to examine the message descriptor (MQMD structure) of the MQSeries message for which conversion was requested. This may help you to determine where the message originated.

AMQ7905 Unexpected exception &4 in data-conversion exit.

Explanation:

The data-conversion exit program, &3, ended with an unexpected exception &4. The message has not been converted.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7907 Unexpected exception in data-conversion exit.

Explanation:

The data-conversion exit routine, &3, ended with an unexpected exception. The message has not been converted.

User action:

Correct the error in the data-conversion exit routine.

AMQ7921 An internal MQSeries error occurred.

Explanation:

The MQDXP structure passed to the Internal Formats Conversion routine contains an incorrect eyecatcher field.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7922 A PCF message is incomplete.

Explanation:

Message data conversion cannot convert a message in Programmable Command Format (PCF) because the message is only &1 bytes long and does not contain a PCF header. The message has either been truncated, or it contains data that is not valid.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7923 A message had an unrecognized integer encoding.

Explanation:

Message data conversion cannot convert a message because the integer encoding value of the message, &1, was not recognized.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7924 Bad length in the PCF header (length = &1).

Explanation:

Message data conversion cannot convert a message in Programmable Command Format (PCF) because the PCF header structure contains an incorrect length field. Either the message has been truncated, or it contains data that is not valid.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message

Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7925 Message version &1 is not supported.

Explanation:

Message data conversion cannot convert a message because the Version field of the message contains an incorrect value.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7926 A PCF message has an incorrect parameter count value &1.

Explanation:

Message data conversion cannot convert a message in Programmable Command Format (PCF) because the parameter count field of the PCF header is incorrect.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7927 Bad type in PCF structure number &1 (type = &2).

Explanation:

A Programmable Command Format (PCF) structure passed to the Internal Formats Converter contained an incorrect type field.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7928 Bad length in PCF structure number &1 (length = &2).

Explanation:

A Programmable Command Format (PCF) structure passed to the Internal Formats Converter contained an incorrect length field.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7929 A PCF structure is incomplete.

Explanation:

Message data conversion cannot convert a message in Programmable Command Format (PCF) because structure number &1, of Type value &2, within the message is incomplete. The message has either been truncated, or it contains data that is not valid.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7930 Bad CCSID in PCF structure number &1 (CCSID = &2).

Explanation:

A Programmable Command Format (PCF) structure passed to the Internal Formats Converter contains an incorrect CCSID.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7931 Bad length in PCF structure number &1 (length = &2).

Explanation:

Message data conversion cannot convert a message in Programmable Command Format (PCF) because one of the structures of the message contains an incorrect length field.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7932 Bad count in PCF structure number &1 (count = &2).

Explanation:

Message data conversion cannot convert a message in Programmable Command Format (PCF) because a StringList structure of the message contains an incorrect count field.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor, the headers of the message, and the incorrect structure to determine the source of the message, and to see how data that is not valid became included in the message.

AMQ7933 Bad string length in PCF structure.**Explanation:**

Message data conversion cannot convert a message in Programmable Command Format (PCF) because structure number &1 of the message contains an incorrect string length value &2.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor, the headers of the message, and the incorrect structure to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7934 Wrong combination of MQCCSI_DEFAULT with MQCCSI_EMBEDDED.**Explanation:**

Message data conversion could not convert a message in Programmable Command Format (PCF) because structure &1 of the message contained a CodedCharSetId field of MQCCSI_DEFAULT while the message itself had a CodedCharSetId of MQCCSI_EMBEDDED. This is an incorrect combination.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor, the headers of the message and the incorrect structure to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7935 Bad CCSID in message header (CCSID = &1).**Explanation:**

Message data conversion could not convert a message because the Message Descriptor of the message contained an incorrect CodedCharSetId field.

User action:

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7936 The file &3 already exists.**Explanation:**

The output file already exists, but REPLACE has not been specified.

User action:

Specify REPLACE to over-write the existing file, or select a different output file name.

AMQ7943 Usage: crtmqcvx SourceFile TargetFile**Explanation:**

See Explanation of message AMQ7953.

User action:

See User action for message AMQ7953.

AMQ7953 One structure has been parsed.**Explanation:**

The crtmqcvx command has parsed one structure.

User action:

None.

AMQ7954 &1 structures have been parsed.**Explanation:**

The crtmqcvx command has parsed %1 structures.

User action:

None.

AMQ7955 Unexpected field: &1.**Explanation:**

The field within the structure is of a type that is not recognized.

User action:

Correct the field and retry the command.

AMQ7956 Bad array dimension.**Explanation:**

An array field of the structure has an incorrect dimension value.

User action:

Correct the field and retry the command.

AMQ7957 Warning at line &1.**Explanation:**

The structure contains another field after a variable length field.

User action:

Correct the structure and retry the command.

AMQ7958 Error at line &1 in field &3.**Explanation:**

Field name '&3' is a field of type 'float'. Fields of type float are not supported by this command.

User action:

Either correct the structure to eliminate fields of type float, or write your own routine to support conversion of these fields.

AMQ7959 Error at line &1 in field &3.**Explanation:**

Field name '&3' is a field of type 'double'. Fields of type double are not supported by this command.

User action:

Either correct the structure to eliminate fields of type double, or write your own routine to support conversion of these fields.

AMQ7960 Error at line &1 in field &3.**Explanation:**

Field name '&3' is a 'pointer' field. Fields of type pointer are not supported by this command.

User action:

Either correct the structure to eliminate fields of type pointer, or write your own routine to support conversion of these fields.

AMQ7961 Error at line &1 in field &3.**Explanation:**

Field name '&3' is a 'bit' field. Bit fields are not supported by this command.

User action:

Either correct the structure to eliminate bit fields, or write your own routine to support conversion of these fields.

AMQ7962 No input file specified.**Explanation:**

This command requires that an input file is specified.

User action:

Specify the name of the input file and retry the command.

AMQ7963 No output file specified.**Explanation:**

This command requires that an output file name is specified.

User action:

Specify the name of the output file and retry the command.

AMQ7964 Unexpected option &3.**Explanation:**

The option specified is not valid for this command.

User action:

Retry the command with a valid option.

AMQ7965 Incorrect number of arguments.**Explanation:**

The command was passed an incorrect number of arguments.

User action:

Retry the command, passing it the correct number of arguments.

AMQ7968 Cannot open file '&3'.**Explanation:**

You cannot open the file &3.

User action:

Check that you have the correct authorization to the file and retry the command.

AMQ7969 Syntax error.**Explanation:**

This line of the input file contains a language syntax error.

User action:

Correct the syntax error and retry the command.

AMQ7970 Syntax error on line &1.**Explanation:**

This message identifies where, in the input file, a previously reported error was detected.

User action:

Correct the error and retry the command.

Administration messages

AMQ8001 MQSeries queue manager created.
Explanation:

MQSeries queue manager &5 created.

User action:

None.

AMQ8002 MQSeries queue manager deleted.
Explanation:

MQSeries queue manager &5 deleted.

User action:

None.

AMQ8003 MQSeries queue manager started.
Explanation:

MQSeries queue manager &5 started.

User action:

None.

AMQ8004 MQSeries queue manager ended.
Explanation:

MQSeries queue manager &5 ended.

User action:

None.

AMQ8005 MQSeries queue manager changed.
Explanation:

MQSeries queue manager &5 changed.

User action:

None.

AMQ8006 MQSeries queue created.
Explanation:

MQSeries queue &5 created.

User action:

None.

AMQ8007 MQSeries queue deleted.
Explanation:

MQSeries queue &5 deleted.

User action:

None.

AMQ8008 MQSeries queue changed.
Explanation:

MQSeries queue &5 changed.

User action:

None.

AMQ8010 MQSeries process created.
Explanation:

MQSeries process &5 created.

User action:

None.

AMQ8011 MQSeries process deleted.
Explanation:

MQSeries process &5 deleted.

User action:

None.

AMQ8012 MQSeries process changed.
Explanation:

MQSeries process &5 changed.

User action:

None.

AMQ8013 MQM process copied.
Explanation:

MQM process &5 created in library &3 by copying.

User action:

None.

AMQ8014 MQSeries channel created.
Explanation:

MQSeries channel &5 created.

User action:

None.

AMQ8015 MQSeries channel deleted.
Explanation:

MQSeries channel &5 deleted.

User action:

None.

AMQ8016 MQSeries channel changed.
Explanation:

MQSeries channel &5 changed.

User action:

None.

AMQ8018 Start MQSeries channel accepted.
Explanation:

MQSeries channel &5 is being started. The start channel function has been initiated. This involves a series of operations across the network before the channel is actually started. The channel status displays "BINDING" for a short period while communication protocols are negotiated with the channel with whom communication is being initiated.

User action:

None.

AMQ8019 Stop MQSeries channel accepted.**Explanation:**

MQSeries channel &5 has been requested to stop.

User action:

None.

AMQ8020 Ping MQSeries channel complete.**Explanation:**

Ping MQSeries channel &5 complete.

User action:

None.

AMQ8021 MQSeries Listener program started.**Explanation:**

The MQSeries channel listener program has been started.

User action:

None.

AMQ8022 MQSeries queue cleared.**Explanation:**

All messages on MQSeries queue &5 have been deleted.

User action:

None.

AMQ8023 MQSeries channel reset.**Explanation:**

MQSeries channel &5 has been reset.

User action:

None.

AMQ8024 MQSeries channel initiator started.**Explanation:**

The channel initiator for MQSeries queue &5 has been started.

User action:

None.

AMQ8025 MQSeries channel resolved.**Explanation:**

In doubt messages for MQSeries channel &5 have been resolved.

User action:

None.

AMQ8026 End MQSeries queue manager accepted.**Explanation:**

A controlled stop request has been initiated for MQSeries queue manager &5.

User action:

None.

AMQ8027 MQSeries command server started.**Explanation:**

The MQSeries command server has been started.

User action:

None.

AMQ8028 MQSeries command server ended.**Explanation:**

The MQSeries command server has been stopped.

User action:

None.

AMQ8029 MQSeries authority granted.**Explanation:**

Authority for MQSeries object &5 granted.

User action:

None.

AMQ8030 MQSeries authority revoked.**Explanation:**

Authority for MQSeries object &5 revoked.

User action:

None.

AMQ8033 MQSeries object recreated.**Explanation:**

MQSeries object &5 has been recreated from image.

User action:

None.

AMQ8034 MQSeries object image recorded.**Explanation:**

Image of MQSeries object &5 has been recorded.

User action:

None.

**AMQ8035 MQSeries Command Server Status . . . :
Running****Explanation:**

See Explanation of message AMQ8041.

User action:

See User action for message AMQ8041.

**AMQ8036 MQSeries command server status . . . :
Stopping****Explanation:**

See Explanation of message AMQ8041.

User action:

See User action for message AMQ8041.

AMQ8037 MQSeries command server status . . . : Starting

Explanation:

See Explanation of message AMQ8041.

User action:

See User action for message AMQ8041.

AMQ8038 MQSeries command server status . . . : Running with queue disabled

Explanation:

See Explanation of message AMQ8041.

User action:

See User action for message AMQ8041.

AMQ8039 MQSeries command server status . . . : Stopped

Explanation:

See Explanation of message AMQ8041.

User action:

See User action for message AMQ8041.

AMQ8040 MQSeries command server ending.

Explanation:

See Explanation of message AMQ8041.

User action:

See User action for message AMQ8041.

AMQ8041 The queue manager cannot be restarted because processes, that were previously connected, are still running.

Explanation:

Processes, that were connected to the queue manager the last time it was running, are still active. The queue manager cannot be restarted.

User action:

Stop the processes and try to start the queue manager.

AMQ8042 Process &1 is still running.

Explanation:

See Explanation of message AMQ8043.

User action:

See User action for message AMQ8043.

AMQ8043 Non runtime application attempted to connect to runtime only queue manager.

Explanation:

A non runtime application attempted to connect to a queue manager on a node where support for non runtime applications has not been installed. The connect attempt will be rejected with a reason of MQRC_ENVIRONMENT_ERROR.

User action:

If the node is intended to support only runtime applications then investigate why a non runtime application has attempted to connect to the queue manager. If the node is intended to

support non runtime only applications then investigate if the base option has been installed. The base option must be installed if non runtime applications are to run on this node.

AMQ8046 Starting LU 6.2 responder for MQSeries channel

Explanation: None

User action: None

AMQ8101 Unexpected error (&1).

Explanation:

An unexpected reason code with hexadecimal value &4 was received from the MQSeries queue manager during command processing. (Note that hexadecimal values in the range X'07D1'-X'0BB7' correspond to MQI reason codes 2001-2999.) More information might be available in the log. If the reason code value indicates that the error was associated with a particular parameter, the parameter concerned is &2.

User action:

Correct the error and then try the command again.

AMQ8102 MQSeries object name specified in &2 not valid.

Explanation:

MQSeries object name &5 specified in &2 is not valid. The length of the name must not exceed 48 characters, or 20 characters if it is a channel name. The name should contain the following characters only: lowercase a-z, uppercase A-Z, numeric 0-9, period (.), forward slash (/), underscore (_) and percent sign (%).

User action:

Change the length of the parameter value or change the parameter value to contain a valid combination of characters, then try the command again.

AMQ8103 Insufficient storage available.

Explanation:

There was insufficient storage available to perform the requested operation.

User action:

Free some storage and then try the command again.

AMQ8104 MQSeries directory &3 not found.

Explanation:

Directory &3 was not found. This directory is created when MQSeries is installed successfully. Refer to the log for more information.

User action:

Verify that installation of MQSeries was successful. Correct the error and then try the command again.

AMQ8105 Object error.

Explanation:

An object error occurred. Refer to the log for more information.

User action:

Correct the error and then try the command again.

AMQ8106 MQSeries queue manager being created.**Explanation:**

The MQSeries queue manager is being created.

User action:

Wait for the creation process to complete and then try the command again.

AMQ8107 MQSeries queue manager running.**Explanation:**

The MQSeries queue manager is running.

User action:

None.

AMQ8108 MQSeries queue manager ending.**Explanation:**

The MQSeries queue manager is ending.

User action:

Wait for the MQSeries queue manager to end and then try the command again.

AMQ8109 MQSeries queue manager being deleted.**Explanation:**

The MQSeries queue manager is being deleted.

User action:

Wait for the deletion process to complete.

AMQ8110 MQSeries queue manager already exists.**Explanation:**

MQSeries queue manager &5 already exists.

User action:

None.

AMQ8117 MQSeries queue manager deletion incomplete.**Explanation:**

Deletion of MQSeries queue manager &5 was only partially successful. An object was not found, or could not be deleted. Refer to the log for more information.

User action:

Delete any remaining MQSeries queue manager objects.

AMQ8118 MQSeries queue manager does not exist.**Explanation:**

MQSeries queue manager &5 does not exist.

User action:

Create the message queue manager (crtmqm command) and then try the command again.

AMQ8135 Not authorized.**Explanation:**

You are not authorized to perform the requested operation for the MQSeries object &5 specified in &2. Either you are not authorized to perform the requested operation, or you are not authorized to the specified MQSeries object. For a copy command, you may not be authorized to the specified source MQSeries object, or, for a create command, you may not be authorized to the system default MQSeries object of the specified type.

User action:

Obtain the necessary authority from your security officer or MQSeries administrator. Then try the command again.

AMQ8137 MQSeries queue manager already starting.**Explanation:**

The strmqm command was unsuccessful because MQSeries queue manager &5 is already starting.

User action:

Wait for the strmqm command to complete.

AMQ8138 The MQSeries queue has an incorrect type.**Explanation:**

The operation is not valid with MQSeries queue &5 because it is not a local queue.

User action:

Change the QNAME parameter to specify an MQSeries queue of the correct type.

AMQ8139 Already connected.**Explanation:**

A connection to the MQSeries queue manager already exists.

User action:

None.

AMQ8140 Resource timeout error.**Explanation:**

A timeout occurred in the communication between internal MQSeries queue manager components. This is most likely to occur when the system is heavily loaded.

User action:

Wait until the system is less heavily loaded, then try the command again.

AMQ8141 MQSeries queue manager starting.**Explanation:**

MQSeries queue manager &5 is starting.

User action:

Wait for the MQSeries queue manager startup process to complete and then try the command again.

AMQ8142 MQSeries queue manager stopped.**Explanation:**

MQSeries queue manager &5 is stopped.

User action:

Use the strmqm command to start the MQSeries queue manager, and then try the command again.

AMQ8143 MQSeries queue not empty.**Explanation:**

MQSeries queue &5 specified in &2 is not empty or contains uncommitted updates.

User action:

Commit or rollback any uncommitted updates. If the command is DELETE QLOCAL, use the CLEAR QLOCAL command to clear the messages from the MQSeries queue. Then try the command again.

AMQ8144 Log not available.**Explanation:**

The MQSeries logging resource is not available.

User action:

Use the dlrmqm command to delete the MQSeries queue manager and then the crtmqm command to create the MQSeries queue manager. Then try the command again.

AMQ8145 Connection broken.**Explanation:**

The connection to the MQSeries queue manager failed during command processing. This may be caused by an endmqm -i command being issued by another user, or by an MQSeries queue manager error.

User action:

Use the strmqm command to start the message queue manager, wait until the message queue manager has started, and try the command again.

AMQ8146 MQSeries queue manager not available.**Explanation:**

The MQSeries queue manager is not available because it has been stopped or has not been created.

User action:

Use the crtmqm command to create the message queue manager, or the strmqm command to start the message queue manager as necessary. Then try the command again.

AMQ8147 MQSeries object not found.**Explanation:**

If the command entered was Change, the MQSeries object &5 specified in &2 does not exist. If the command entered was Copy, the source MQSeries object does not exist. If the command entered was Create, the system default MQSeries object of the specified type does not exist.

User action:

Correct the MQSeries object name and then try the command again or, if you are creating a new MQSeries queue or process object, either specify all parameters explicitly or ensure that the system default object of the required type exists. The system default queue names are SYSTEM.DEFAULT.LOCAL.QUEUE, SYSTEM.DEFAULT.ALIAS.QUEUE and SYSTEM.DEFAULT.REMOTE.QUEUE. The system default process name is SYSTEM.DEFAULT.PROCESS.

AMQ8148 MQSeries object in use.**Explanation:**

MQSeries object &5 specified in &2 is in use by an MQSeries application program.

User action:

Wait until the MQSeries object is no longer in use and then try the command again, or specify FORCE to force the processing of the MQSeries ALTER command regardless of any application program affected by the change. If the object is the dead-letter queue and the open input count is nonzero, it may be in use by an MQSeries channel. If the object is another MQSeries queue object with a nonzero open output

count, it may be in use by an MQSeries channel (of type RCVR or RQSTR). In either case, use the STOP CHANNEL and START CHANNEL commands to stop and restart the channel in order to solve the problem.

AMQ8149 MQSeries object damaged.**Explanation:**

The MQSeries object &5 specified in &2 is damaged.

User action:

The MQSeries object contents are not valid. Issue the DISPLAY CHANNEL, DISPLAY QUEUE, or DISPLAY PROCESS command, as required, to determine the name of the damaged object. Issue the DEFINE command, for the appropriate object type, to replace the damaged object, then try the command again.

AMQ8150 MQSeries object already exists.**Explanation:**

MQSeries object &5 specified for &2 could not be created because it already exists.

User action:

Check that the name is correct and try the command again specifying REPLACE, or delete the MQSeries object. Then try the command again.

AMQ8151 MQSeries object has different type.**Explanation:**

The type specified for MQSeries object &5 is different from the type of the object being altered or defined.

User action:

Use the correct MQSeries command for the object type, and then try the command again.

AMQ8152 Source MQSeries object has different type.**Explanation:**

The type of the source MQSeries object is different from that specified.

User action:

Correct the name of the command, or source MQSeries object name, and then try the command again, or try the command using the REPLACE option.

AMQ8153 Insufficient disk space for the specified queue.**Explanation:**

The command failed because there was insufficient disk space available for the specified queue.

User action:

Release some disk space and then try the command again.

AMQ8155 Connection limit exceeded.**Explanation:**

The queue manager connection limit has been exceeded.

User action:

The maximum limit on the number of MQSeries application programs that may be connected to the MQSeries queue manager has been exceeded. Try the command later.

AMQ8156 MQSeries queue manager quiescing.**Explanation:**

The MQSeries queue manager is quiescing.

User action:

The queue manager was stopping with -c specified for endmqm. Wait until the queue manager has been restarted and then try the command again.

AMQ8157 Security error.**Explanation:**

An error was reported by the security manager program.

User action:

Inform your systems administrator, wait until the problem has been corrected, and then try the command again.

AMQ8159 MAXDEPTH not allowed with queue type *ALS or *RMT.**Explanation:**

The MAXDEPTH parameter may not be specified for an MQM queue of type *ALS or *RMT.

User action:

Remove the MAXDEPTH parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8160 DFTSHARE not allowed with queue type *ALS or *RMT.**Explanation:**

The DFTSHARE parameter may not be specified for an MQM queue of type *ALS or *RMT.

User action:

Remove the DFTSHARE parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8172 Already disconnected.**Explanation:**

The MQI reason code of 2018 was returned from the MQSeries queue manager in response to an MQDISC request issued during command processing.

User action:

None.

AMQ8173 No processes to display.**Explanation:**

There are no matching processes defined on this system.

User action:

Using the DEFINE PROCESS command to create a process.

AMQ8174 No queues to display.**Explanation:**

There are no matching queues defined on this system.

User action:

Using the appropriate command to define a queue of the type that you require, that is, DEFINE QALIAS, DEFINE QLOCAL, DEFINE QMODEL, or DEFINE QREMOTE.

AMQ8185 Operating system object already exists.**Explanation:**

The MQSeries object cannot be created because an object that is not known to MQSeries already exists in the MQSeries directory with the name that should be used for the new object. Refer to the log for previous messages.

User action:

Remove the non-MQSeries object from the MQSeries library, and try the command again.

AMQ8186 Image not available for MQSeries object &5.**Explanation:**

MQSeries object &5 type &3 cannot be recreated because the image is not fully available in the logs that are currently online. Refer to earlier messages in the error log for information about the error logs that need to be brought online for this object to be recreated.

User action:

Bring the relevant error logs online, and try the command again.

AMQ8187 MQSeries object &5 is currently open.**Explanation:**

MQSeries object &5, type &3, is currently in use, so the &1 command cannot be issued against it. If a generic list was presented to the command, the command is still issued against the other objects in the list.

User action:

Wait until the object is no longer in use, and try the command again.

AMQ8188 Insufficient authorization to MQSeries object &5.**Explanation:**

You are not authorized to issue the &1 command against MQSeries object &5 type &3. If a generic list was presented to the command, the command is still issued against the other objects in the list.

User action:

Obtain sufficient authorization for the object, and retry the command.

AMQ8189 MQSeries object &5 is damaged.**Explanation:**

MQSeries object &5 type &3 is damaged and the &1 command cannot be issued against it. If a generic list was presented to the command then the command is still issued against the other objects in the list.

User action:

Issue the appropriate DEFINE command for the object, specifying REPLACE, and then try the command again.

AMQ8190 &1 succeeded on &2 objects and failed on &3 objects.

Explanation:

An operation performed on a generic list of objects was not completely successful.

User action:

Examine the log for details of the errors encountered, and take appropriate action.

AMQ8191 MQSeries command server is starting.

Explanation:

The MQSeries command server is starting.

User action:

Wait for the strmqcsv command to complete and then try the operation again.

AMQ8192 MQSeries command server already starting.

Explanation:

The request to start the MQSeries command server was unsuccessful because the MQSeries command server is already starting.

User action:

Wait for the strmqcsv command to complete.

AMQ8193 MQSeries command server is ending.

Explanation:

The MQSeries command server is ending.

User action:

Wait for the endmqcsv command to complete and then try the command again.

AMQ8194 MQSeries command server already ending.

Explanation:

The end MQSeries command server request was unsuccessful because the MQSeries command server is already ending.

User action:

Wait for the endmqcsv command to complete.

AMQ8195 MQSeries command server already running.

Explanation:

The strmqcsv command was unsuccessful because the MQSeries command server is already running.

User action:

None.

AMQ8196 MQSeries command server already stopped.

Explanation:

The request to end the MQSeries command server was unsuccessful because the MQSeries command server is already stopped.

User action:

None.

AMQ8197 Deleted MQSeries queue damaged.

Explanation:

The deleted MQSeries queue &5 was damaged, and any messages it contained have been lost.

User action:

None.

AMQ8226 MQSeries channel already exists.

Explanation:

MQSeries channel &3 cannot be created because it already exists.

User action:

Check that the name is correct and try the command again specifying REPLACE, or delete the MQSeries channel and then try the command again.

AMQ8227 Channel &3 not found.

Explanation:

ALTER CHANNEL has been issued for a nonexistent channel.

User action:

Correct the MQSeries channel name and then try the command again.

AMQ8296 &4 MQSC commands completed successfully.

Explanation:

The &1 command has completed successfully. The &4 MQSeries commands from &5 have been processed without error and a report written to the printer spool file.

User action:

None.

AMQ8297 &4 MQSC commands verified successfully.

Explanation:

The &1 command completed successfully. The &4 MQSeries commands from &5 have been verified and a report written to the printer spool file.

User action:

None.

AMQ8298 Error report generated for MQSC command process.

Explanation:

The &1 command attempted to process the sequence of MQSeries commands from &5 and encountered some errors, however, the operation may have partially completed. A report has been written to the printer spool file.

User action:

Examine the spooled printer file for details of the errors encountered, correct the MQSC source file, and retry the operation.

AMQ8299 Cannot open &5 for MQSC process.**Explanation:**

The &1 command failed to open &5 for MQSeries command processing.

User action:

Check that the intended file exists, and has been specified correctly. Correct the specification or create the object, and try the operation again.

AMQ8302 Internal failure initializing MQSeries services.**Explanation:**

An error occurred while attempting to initialize MQSeries services.

User action:

None.

AMQ8303 Insufficient storage available to process request.**Explanation:****User action:**

See User action for message AMQ8304.

AMQ8304 Tracing cannot be started. Too many traces are already running.**Explanation:****User action:**

Stop one or more of the other traces and try the command again.

AMQ8401 &1 MQSC commands read.**Explanation:**

The MQSC script contains &1 commands.

User action:

None.

AMQ8402 &1 commands have a syntax error.**Explanation:**

The MQSC script contains &1 commands having a syntax error.

User action:

None.

AMQ8403 &1 commands cannot be processed.**Explanation:**

The MQSC script contains &1 commands that failed to process.

User action:

None.

AMQ8404 Command failed.**Explanation:**

An MQSC command has been recognized, but cannot be processed.

User action:

None.

AMQ8405 Syntax error detected at or near end of command segment below:-**Explanation:**

The MQSC script contains &1 commands having a syntax error.

User action:

None.

AMQ8406 Unexpected 'end of input' in MQSC.**Explanation:**

An MQSC command contains a continuation character, but the 'end of input' has been reached without completing the command.

User action:

None.

AMQ8407 Display Process details.**Explanation:**

The MQSC DISPLAY PROCESS command completed successfully, and details follow this message.

User action:

None.

AMQ8408 Display Queue Manager details.**Explanation:**

The MQSC DISPLAY QMGR command completed successfully, and details follow this message.

User action:

None.

AMQ8409 Display Queue details.**Explanation:**

The MQSC DISPLAY QUEUE command completed successfully, and details follow this message.

User action:

None.

AMQ8410 Parser error.**Explanation:**

The MQSC Parser has an internal error.

User action:

None.

AMQ8411 Duplicate Keyword error.**Explanation:**

A command in the MQSC script contains duplicate keywords.

User action:

None.

AMQ8412 Numeric Range error.**Explanation:**

The value assigned to an MQSC command keyword is out of the permitted range.

User action:

None.

AMQ8413 String Length Error.

Explanation:

A string assigned to an MQSC keyword is either NULL, or longer than the maximum permitted for that keyword.

User action:

None.

AMQ8414 Display Channel details.

Explanation:

The MQSC DISPLAY CHL command completed successfully, and details follow this message.

User action:

None.

AMQ8415 MQSeries commands are active.

Explanation:

The MQSC DISPLAY QMGR command completed successfully, and details follow this message.

User action:

None.

AMQ8416 MQSC timed out waiting for a response from the command server.

Explanation:

MQSC did not receive a response message from the remote command server in the time specified.

User action:

None.

AMQ8417 Display Channel Status details.

Explanation:

The MQSC DISPLAY CHANNEL STATUS command completed successfully, and details follow this message.

User action:

None.

AMQ8418 &1 command responses received.

Explanation:

Running in queued mode, &1 command responses were received from the remote command server.

User action:

None.

AMQ8419 The Queue is already in the DCE cell.

Explanation:

The Queue is already in the cell, that is, its SCOPE attribute is already CELL.

User action:

None.

AMQ8420 Channel Status not found.

Explanation:

No status was found for the specified channel(s).

User action:

None.

AMQ8421 A required keyword was not specified.

Explanation:

A keyword required in this command was not specified.

User action:

None.

AMQ8424 Error detected in a name keyword.

Explanation:

A keyword in an MQSC command contained a name string which was not valid. This may be because it contained characters which are not accepted in MQ names. Typical keywords which can produce this error are QLOCAL (and the other q types), CHANNEL, XMITQ, INITQ, MCANAME etc.

User action:

None.

AMQ8498 Starting MQSeries Commands.

Explanation:

The MQSC script contains &1 commands.

User action:

None.

AMQ8499 Usage: runmqsc -e -v -w WaitTime -x -i -o QMgrName

Explanation:

None.

User action:

None.

AMQ8500 MQSeries Display MQ Files

Explanation:

Title for the dspmqfls command.

User action:

None.

AMQ8501 Common services initialization failed with return code &1.

Explanation:

A request by the command server to initialize common services failed with return code &1.

User action:

None.

AMQ8502 Connect shared memory failed with return code &1.

Explanation:

A request by the command server to connect shared memory failed with return code &1.

User action:

None.

AMQ8503 Post event semaphore failed with return code &1.**Explanation:**

A request by the command server to post an event semaphore failed with return code &1.

User action:

None.

AMQ8504 Command server MQINQ failed with reason code &1.**Explanation:**

An MQINQ request by the command server, for the MQSeries queue &3, failed with reason code &1.

User action:

None.

AMQ8505 Reallocate memory failed with return code &1.**Explanation:**

A request by the command server to reallocate memory failed with return code &1.

User action:

None.

AMQ8506 Command server MQGET failed with reason code &1.**Explanation:**

An MQGET request by the command server, for the MQSeries queue &3, failed with reason code &1.

User action:

None.

AMQ8507 Command server MQPUT1 request for an undelivered message failed with reason code &1.**Explanation:**

An attempt by the command server to put a message to the dead-letter queue, using MQPUT1, failed with reason code &1. The MQDLH reason code was &2.

User action:

None.

AMQ8508 Queue Manager Delete Object List failed with return code &1.**Explanation:**

A request by the command server to delete a queue manager object list failed with return code &1.

User action:

None.

AMQ8509 Command server MQCLOSE reply-to queue failed with reason code &1.**Explanation:**

An MQCLOSE request by the command server for the reply-to queue failed with reason code &1.

User action:

None.

AMQ8511 Usage: strmqcsv QMgrName**Explanation:**

See Explanation of message AMQ8514.

User action:

None.

AMQ8512 Usage: endmqcsv [-c | -i] QMgrName**Explanation:**

See Explanation of message AMQ8514.

User action:

None.

AMQ8513 Usage: dspmqcsv QMgrName**Explanation:**

See Explanation of message AMQ8514.

User action:

None.

AMQ8514 No response received after &1 seconds.**Explanation:**

The command server has not reported the status of running, to the start request, before the timeout of &1 seconds was reached.

User action:

None.

AMQ8515 MQSeries Alter MQ Files**Explanation:**

Title for the altmqfls command.

User action:

None.

AMQ8516 MQSeries Clean Queue Manager**Explanation:**

Title for the cleanqm command.

User action:

None.

AMQ8517 The messages files are partitioned and cannot be moved.**Explanation:**

Partition error from the altmqfls command.

User action:

None.

AMQ8601 MQSeries trigger monitor started.**Explanation:**

The MQSeries trigger monitor has been started.

User action:

None.

AMQ8602 MQSeries trigger monitor ended.

Explanation:

The MQSeries trigger monitor has ended.

User action:

None.

AMQ8603 Usage: runmqtrm [-m QMgrName] [-q InitQ]

Explanation:

See Explanation of message AMQ8604.

User action:

See User action for message AMQ8604.

AMQ8604 Use of MQSeries trigger monitor not authorized.

Explanation:

The MQSeries trigger monitor cannot be run due to lack of authority to the requested queue manager or initiation queue.

User action:

Obtain the necessary authority from your security officer or MQSeries administrator. Then try the command again.

AMQ8605 Queue manager not available to the MQSeries trigger monitor

Explanation:

The queue manager specified for the trigger monitor does not exist, or is not active.

User action:

Check that you named the correct queue manager. Ask your systems administrator to start it, if it is not active. Then try the command again.

AMQ8606 Insufficient storage available for the MQSeries trigger monitor.

Explanation:

There was insufficient storage available for the MQSeries trigger monitor to run.

User action:

Free some storage and then try the command again.

AMQ8607 MQSeries trigger monitor connection failed.

Explanation:

The trigger monitor's connection to the requested queue manager failed because of MQI reason code &1 from MQCONN.

User action:

Consult your systems administrator about the state of the queue manager.

AMQ8608 MQSeries trigger monitor connection broken.

Explanation:

The connection to the queue manager failed while the trigger monitor was running. This may be caused by an endmqm command being issued by another user, or by an MQSeries queue manager error.

User action:

Consult your systems administrator about the state of the queue manager.

AMQ8609 Initiation queue missing or wrong type

Explanation:

The named initiation queue could not be found; or the queue type is not correct for an initiation queue.

User action:

Check that the named queue exists, and is a local queue, or that the named queue is an alias for a local queue which exists.

AMQ8610 Initiation queue in use

Explanation:

The MQSeries trigger monitor could not open the initiation queue because the queue is open for exclusive use by another application.

User action:

Wait until the queue is no longer in use, and try the command again.

AMQ8611 Initiation queue could not be opened.

Explanation:

The MQSeries trigger monitor could not open the initiation queue; reason code &1 was returned from MQOPEN.

User action:

Consult your systems administrator.

AMQ8612 Waiting for a trigger message

Explanation:

The MQSeries trigger monitor is waiting for a message to arrive on the initiation queue.

User action:

None.

AMQ8613 Initiation queue changed or deleted

Explanation:

The MQSeries trigger monitor is unable to continue because the initiation queue has been deleted or changed since it was opened.

User action:

Retry the command.

AMQ8614 Initiation queue not enabled for input.

Explanation:

The MQSeries trigger monitor cannot read from the initiation queue because input is not enabled.

User action:

Ask your systems administrator to enable the queue for input.

AMQ8615 MQSeries trigger monitor failed to get message.

Explanation:

The MQSeries trigger monitor failed because of MQI reason code &1 from MQGET.

User action:

Consult your systems administrator.

AMQ8616 End of application trigger.**Explanation:**

The action to trigger an application has been completed.

User action:

None.

AMQ8617 Not a valid trigger message.**Explanation:**

The MQSeries trigger monitor received a message that is not recognized.

User action:

Consult your systems administrator.

AMQ8618 Error starting triggered application.**Explanation:**

An error was detected when trying to start the application identified in a trigger message.

User action:

Check that the application the trigger monitor was trying to start is available.

AMQ8619 Application type &1 not supported.**Explanation:**

A trigger message was received which specifies application type &1; the trigger monitor does not support this type.

User action:

Use an alternative trigger monitor for this initiation queue.

AMQ8620 Trigger message with warning &1**Explanation:**

The trigger monitor received a message with a warning. For example, it may have been truncated or it could not be converted to the trigger monitor's data representation. The reason code for the warning is &1.

User action:

None.

AMQ8621 Usage: runmqtmc [-m QMgrName] [-q InitQ]**Explanation:**

See Explanation of message AMQ8708.

User action:

None.

AMQ8622 Usage: CICS-Transaction-Name [MQTMC2 structure]**Explanation:**

See Explanation of message AMQ8708.

User action:

None.

AMQ8701 Usage: rcdmqimg [-z] [-m QMgrName] -t ObjType [GenericObjName]**Explanation:**

See Explanation of message AMQ8708.

User action:

None.

AMQ8702 Usage: rcrmobj [-z] [-m QMgrName] -t ObjType [GenericObjName]**Explanation:**

See Explanation of message AMQ8708.

User action:

None.

AMQ8703 Usage: dspmqfls [-m QMgrName] [-t ObjType] GenericObjName**Explanation:**

See Explanation of message AMQ8708.

User action:

None.

AMQ8704 Usage: altmqfls [-m QMgrName] -t ObjType [-v NewVolume] [-s MQSSName] [-u UpdateOpt] [-c RecalcOpt] [-g GetOpt] ObjectName**User action:**

None.

AMQ8708 Dead letter queue handler started to process INPUTQ(&3).**Explanation:**

The dead letter queue handler (runmqdlq) has been started and has parsed the input file without detecting any errors and is about to start processing the queue identified in the message.

User action:

None.

AMQ8709 Dead letter queue handler ending.**Explanation:**

The dead letter queue handler (runmqdlq) is ending because the WAIT interval has expired and there are no messages on the dead letter queue, or because the queue manager is shutting down, or because the dead letter queue handler has detected an error. If the dead letter queue handler has detected an error, an earlier message will have identified the error.

User action:

None.

AMQ8710 Cannot move queue file to 'x' (where 'x' is a volume)**Explanation:** None**User action:** None**AMQ8711 Queue files moved to 'x'****Explanation:** None**User action:** None

AMQ8721 Dead letter queue message not prefixed by a valid MQDLH.**Explanation:**

The dead letter queue handler (runmqdlq) retrieved a message from the nominated dead letter queue, but the message was not prefixed by a recognizable MQDLH. This typically occurs because an application is writing directly to the dead letter queue but is not prefixing messages with a valid MQDLH. The message is left on the dead letter queue and the dead letter queue handler continues to process the dead letter queue. Each time the dead letter queue handler repositions itself to a position before this message to process messages that could not be processed on a previous scan it will reprocess the failing message and will consequently reissue this message.

User action:

Remove the invalid message from the dead letter queue. Do not write messages to the dead letter queue unless they have been prefixed by a valid MQDLH. If you require a dead letter queue handler that can process messages not prefixed by a valid MQDLH, you must change the sample program called amqsdq to cater for your needs.

AMQ8722 Dead letter queue handler unable to put message: Rule &1 Reason &2.**Explanation:**

This message is produced by the dead letter queue handler when it is requested to redirect a message to another queue but is unable to do so. If the reason that the redirect fails is the same as the reason the message was put to the dead letter queue then it is assumed that no new error has occurred and no message is produced. The retry count for the message will be incremented and the dead letter queue handler will continue.

User action:

Investigate why the dead letter queue handler was unable to put the message to the dead letter queue. The line number of the rule used to determine the action for the message should be used to help identify to which queue the dead letter queue handler attempted to PUT the message.

AMQ8741 Unable to connect to queue manager(&3) : CompCode = &1 Reason = &2.**Explanation:**

The dead letter queue handler (runmqdlq) could not connect to the requested queue manager. This message is typically issued when the requested queue manager has not been started or is quiescing, or if the process does not have sufficient authority. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action:

Take appropriate action based upon the completion code and reason.

AMQ8742 Unable to open queue manager: CompCode = &1 Reason = &2.**Explanation:**

The dead letter queue handler (runmqdlq) could not open the queue manager object. This message is typically issued because of a resource shortage or because the process does not have sufficient authority. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action:

Take appropriate action based upon the completion code and reason.

AMQ8743 Unable to inquire on queue manager: CompCode = &1 Reason = &2.**Explanation:**

The dead letter queue handler (runmqdlq) could not inquire on the queue manager. This message is typically issued because of a resource shortage or because the queue manager is ending. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action:

Take appropriate action based upon the completion code and reason.

AMQ8744 Unable to close queue manager: CompCode = &1 Reason = &2.**Explanation:**

The dead letter queue handler (runmqdlq) could not close the queue manager. This message is typically issued because of a resource shortage or because the queue manager is ending. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action:

Take appropriate action based upon the completion code and reason.

AMQ8745 Unable to open dead letter queue(&3) for browse: CompCode = &1 Reason = &2.**Explanation:**

The dead letter queue handler (runmqdlq) could not open the dead letter queue for browsing. This message is typically issued because another process has opened the dead letter queue for exclusive access, or because an invalid dead letter queue name was specified. Other possible reasons include resource shortages or insufficient authority. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action:

Take appropriate action based upon the completion code and reason.

AMQ8746 Unable to close dead letter queue: CompCode = &1 Reason = &2.**Explanation:**

The dead letter queue handler (runmqdlq) could not close the dead letter queue. This message is typically issued because of a resource shortage or because the queue manager is ending. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action:

Take appropriate action based upon the completion code and reason.

AMQ8747 Integer parameter(&2) outside permissible range for &3 on line &1.**Explanation:**

An integer supplied as input to the dead letter handler was outside of the valid range of values for a particular keyword.

User action:

Correct the input data and restart the dead letter queue handler.

AMQ8748 Unable to get message from dead letter queue: CompCode = &1 Reason = &2.**Explanation:**

The dead letter queue handler (runmqdlq) could not get the next message from the dead letter queue. This message is typically issued because of the queue manager ending, a resource problem, or another process having deleted the dead letter queue. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action:

Take appropriate action based upon the completion code and reason.

AMQ8749 Unable to commit/backout action on dead letter queue: CompCode = &1 Reason**Explanation:**

The dead letter queue handler (runmqdlq) was unable to commit or backout an update to the dead letter queue. This message is typically issued because of the queue manager ending, or because of a resource shortage. If the queue manager has ended, the update to the dead letter queue (and any associated updates) will be backed out when the queue manager restarts. If the problem was due to a resource problem then the updates will be backed out when the dead letter queue handler terminates. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action:

Take appropriate action based upon the completion code and reason.

AMQ8750 No valid input provided to runmqdlq.**Explanation:**

Either no input was provided to runmqdlq, or the input to runmqdlq contained no valid message templates. If input was provided to runmqdlq but was found to be invalid, earlier messages will have been produced explaining the cause of the error. The dead letter queue handler will end.

User action:

Correct the input data and restart the dead letter queue handler.

AMQ8751 Unable to obtain private storage.**Explanation:**

The dead letter queue handler (runmqdlq) was unable to obtain private storage. This problem would typically arise as a result of some more global problem. For example if there is a persistent problem that is causing messages to be written to the DLQ and the same problem (for example queue full) is preventing the dead letter queue handler from taking the requested action with the message, it is necessary for the dead letter queue handler to maintain a large amount of state data to remember the retry counts associated with each message, or if the dead letter queue contains a large number of messages and the rules table has directed the dead letter queue handler to ignore the messages.

User action:

Investigate if some more global problem exists, and if the dead letter queue contains a large number of messages. If the problem persists contact your support center.

AMQ8752 Parameter(&3) exceeds maximum length on line &1.**Explanation:**

A parameter supplied as input to the dead letter handler exceeded the maximum length for parameters of that type.

User action:

Correct the input data and restart the dead letter queue handler.

AMQ8753 Duplicate parameter(&3) found on line &1.**Explanation:**

Two or more parameters of the same type were supplied on a single input line to the dead letter queue handler.

User action:

Correct the input and restart the dead letter queue handler.

AMQ8756 Error detected releasing private storage.**Explanation:**

The dead letter queue handler (runmqdlq) was informed of an error while attempting to release an area of private storage. The dead letter queue handler ends.

User action:

This message should be preceded by a message or FFST information from the internal routine that detected the error. Take the action associated with the earlier error information.

AMQ8757 Integer parameter(&3) outside permissible range on line &1.**Explanation:**

An integer supplied as input to the dead letter handler was outside of the valid range of integers supported by the dead letter queue handler.

User action:

Correct the input data and restart the dead letter queue handler.

AMQ8758 &1 errors detected in input to runmqdlq.**Explanation:**

One or more errors have been detected in the input to the dead letter queue handler(runmqdlq). Error messages will have been generated for each of these errors. The dead letter queue handler ends.

User action:

Correct the input data and restart the dead letter queue handler.

AMQ8759 Invalid combination of parameters to dead letter queue handler on line &1.**Explanation:**

An invalid combination of input parameters has been supplied to the dead letter queue handler. Possible causes are:
no ACTION specified,
ACTION(FWD) but no FWDQ specified,
HEADER(YES|NO) specified without ACTION(FWD).

User action:

Correct the input data and restart the dead letter queue handler.

AMQ8760 Unexpected failure while initializing process: Reason = &1.**Explanation:**

The dead letter queue handler (runmqdlq) could not perform basic initialization required to use MQ services because of an unforeseen error. The dead letter queue handler ends.

User action:

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8761 Unexpected failure while connecting to queue manager: CompCode = &1 Reason**Explanation:**

The dead letter queue handler (runmqdlq) could not connect to the requested queue manager because of an unforeseen error. The dead letter queue handler ends.

User action:

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8762 Unexpected error while attempting to open queue manager: CompCode = &1 Reason = &2.**Explanation:**

The dead letter queue handler (runmqdlq) could not open the queue manager because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action:

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8763 Unexpected error while inquiring on queue manager: CompCode = &1 Reason = &**Explanation:**

The dead letter queue handler (runmqdlq) could not inquire on the queue manager because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action:

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8764 Unexpected error while attempting to close queue manager: CompCode = &1 Reason = &2.**Explanation:**

The dead letter queue handler (runmqdlq) could not close the queue manager because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action:

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8765 Unexpected failure while opening dead letter queue for browse: CompCode = &1 Reason = &2.**Explanation:**

The dead letter queue handler (runmqdlq) could not open the dead letter queue for browsing because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action:

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8766 Unexpected error while closing dead letter queue: CompCode = &1 Reason = &2**Explanation:**

The dead letter queue handler (runmqdlq) could not close the dead letter queue because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action:

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8767 Unexpected error while getting message from dead letter queue: CompCode = &1 Reason = &2.

Explanation:

The dead letter queue handler (runmqdlq) could not get the next message from the dead letter queue because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action:

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8768 Unexpected error committing/backing out action on dead letter queue: CompCode = &1 Reason = &2.

Explanation:

The dead letter queue handler (runmqdlq) was unable to either commit or backout an update to the dead letter queue because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action:

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8769 Unable to disconnect from queue manager: CompCode = &1 Reason = &2.

Explanation:

The dead letter queue handler (runmqdlq) was unable to disconnect from the queue manager because of an unexpected error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

User action:

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Contact your support center. Do not discard these files until the problem has been resolved.

AMQ8801 EC Boss &3 for Queue Manager &4 is Initializing

Explanation:

The EC Boss for Queue Manager &4 is beginning the start up sequence. The process name of the EC Boss is &3

User action:

None

AMQ8802 EC Boss &3 for Queue Manager &4 initialization complete.

Explanation:

The EC Boss for Queue Manager &4 has completed process start up actions. The process name of the EC Boss is &3

User action:

None

AMQ8803 EC Boss &3 for Queue Manager &4 controlled shutdown initiated.

Explanation:

The EC Boss for Queue Manager &4 has entered the controlled shutdown state. The Queue Manager will not accept new work, and once operations in progress have completed, connections will be terminated. When there are no more connections, the Queue Manager will end. The process name of the EC Boss is &3.

User action:

None

AMQ8804 EC Boss &3 for Queue Manager &4 quiesce shutdown initiated.

Explanation:

The EC Boss for Queue Manager &4 has entered the quiesce shutdown state.

The Queue Manager will not accept new work, but will allow existing connections to complete before ending. The process name of the EC Boss is &3.

User action:

None

AMQ8805 EC Boss &3 for Queue Manager &4 immediate shutdown initiated.

Explanation:

The EC Boss for Queue Manager &4 has entered the immediate shutdown state. Any current connections are terminated and the Queue Manager will end immediately. The process name of the EC Boss is &3.

User action:

None

AMQ8806 EC / EC Boss &3 for Queue Manager &4 cannot access file &5.

Explanation:

An EC, or the EC Boss (process name &3) for Queue Manager &4 has not been able to access the file named &5. This file is critical to the operation of the Queue Manager, and the Queue Manager will not start properly until the problem is corrected.

User action:

End the Queue Manager and check the existence or file attributes of the file named &5. Verify that the file exists, and has the appropriate file security and type attributes, correct the problem and restart the Queue Manager.

AMQ8807 EC / EC Boss &3 for Queue Manager &4 obtained file error &1 on file &5.

Explanation:

An EC, or the EC Boss (process name &3) for Queue Manager &4 obtained Tandem file error &1 while attempting an IO operation to file &5.

The successful completion of the IO operation may be critical to the correct operation of the Queue Manager, and the Queue Manager may not operate properly until the problem is corrected.

User action:

End the Queue Manager and check the file attributes of the file named &5.

Verify that the file exists, and has the appropriate file security and type attributes, correct the problem and restart the Queue Manager.

AMQ8808 Incorrect Queue Manager name &4 supplied to process &3

Explanation:

A Queue Manager process (process name &3) was supplied with an invalid or nonexistent Queue Manager name, &4. The initialization of the process failed as a result.

User action:

End the Queue Manager and check queue manager name that is being used in the configuration databases. After correcting the problem restart the Queue Manager.

AMQ8809 Queue Manager &4 Started.

Explanation:

The EC Boss has reported that the Queue Manager named &4 has entered the "started" state.

User action:

None.

AMQ8810 EC number &1, process name &3, for Queue Manager &4 is initializing.

Explanation:

An EC in the Queue Manager named &4 has started and is performing process initialization.

User action:

None.

AMQ8811 EC number &1, process name &3, for Queue Manager &4 has completed initialization.

Explanation:

An EC in the Queue Manager named &4 has completed process initialization.

User action:

None.

AMQ8812 EC number &1, process name &3, for Queue Manager &4 has started controlled shutdown.

Explanation:

An EC in the Queue Manager named &4 has reported that a controlled shutdown has started. The EC will wait for all currently running agents to end before performing the final shutdown actions.

User action:

None.

AMQ8813 EC number &1, process name &3, for Queue Manager &4 has started quiesce shutdown.

Explanation:

An EC in the Queue Manager named &4 has reported that a quiesce shutdown has started. The EC will wait for all currently running agents to end before performing the final shutdown actions.

User action:

None.

AMQ8814 EC number &1, process name &3, for Queue Manager &4 has started immediate shutdown.

Explanation:

An EC in the Queue Manager named &4 has reported that an immediate shutdown has started. The EC will terminate immediately, without waiting for currently running agents to end.

User action:

None.

AMQ8815 EC number &1, process name &3, for Queue Manager &4 has shutdown.

Explanation:

An EC in the Queue Manager named &4 has reported that it has completed shutdown actions. When all ECs in the Queue Manager have completed shutdown actions, the Queue Manager will end.

User action:

None.

AMQ8816 Queue Manager &4 has started, though only &1 of &2 ECs have registered.

Explanation:

The Queue Manager named &4 has entered the started state, and will now accept connections. However, only &1 of the expected &2 ECs have registered with the EC Boss. The Queue Manager's load balancing and overall performance will be adversely affected, however it will still be able to service connections.

User action:

Examine the logs to determine the cause of the failure to start the missing ECs. End the Queue Manager, and rectify the problem if possible. Restart the Queue Manager and ensure that the Queue Manager starts correctly.

AMQ8817 Process &3 in Queue Manager &4 cannot process a request to a resource problem.

Explanation:

The Queue Manager named &4 has entered the started state, and will now accept connections. However, only &1 of the expected &2 ECs have registered with the EC Boss. The Queue Manager's load balancing and overall performance will be adversely affected, however it will still be able to service connections.

User action:

The process named &3 has failed to process a request from another process due to failure to allocate a resource, such as memory, or disk space. Depending on the criticality of the resource itself, this may cause further errors, or the failure of certain Queue Manager components. User Action: Examine the logs to determine the cause of the failure. If there are resource problems that can be corrected, correct them and attempt the operation again.

AMQ8818 EC Boss in Queue Manager &4 rejected a registration from process &3.**Explanation:**

The process named &3 attempted to register with the EC Boss. The EC Boss detected a problem with the registration information and rejected the attempt.

User action:

Examine the logs to determine further information about the problem. Determine the identity of the process, and verify that the process is an EC. If the process is not an EC, or cannot be identified, then a security threat may be present.

AMQ8819 EC Number &1 registered with the EC Boss in Queue Manager &4.**Explanation:**

EC number &1 has registered with the EC Boss. When all the expected ECs have registered, the Queue Manager enters the started state.

User action:

None.

AMQ8820 An unknown message received by process &3 in Queue Manager &4 from process &5 has been rejected.**Explanation:**

The process &3 has received and rejected a message that is either not of the correct format, or from an unknown source.

User action:

Examine the log to determine if further information is available. Try to identify the process to ensure that a security threat is not present.

AMQ8821 The EC Boss in Queue Manager &4 detected the failure of EC number &1**Explanation:**

The EC Boss has detected that EC number &1 has terminated unexpectedly.

If the maximum number of restarts performed on this EC has not already been exceeded, PATHWAY will attempt to restart the EC.

User action:

Examine the log to determine if further information is available.

AMQ8822 The EC Boss in Queue Manager &4 failed to find an EC to service a request.**Explanation:**

The EC Boss failed to find an active EC to service a request that was made, either by an application (in order to start a connection), or by an administration command (for example, to start or stop a channel). It is possible that all the ECs in the Queue Manager have failed repeatedly, exceeding the maximum number of restarts allowed by PATHWAY.

User action:

Examine the log to see if further information is available on the state of the Queue Manager. The Queue Manager will need to be ended and restarted.

AMQ8823 Process &3 in Queue Manager &4 received and rejected a message from an unknown source, &5.**Explanation:**

A process in Queue Manager &4 received and rejected a message from a source that is not authorized or not registered to communicate with the Queue Manager. The process is identified by &5. The process that received the message is identified by &3.

User action:

Examine the log to determine if further information is available on the identity of the source of the message. Try to determine the identity of the sender to ensure that a security threat is not present.

AMQ8824 The EC Boss in Queue Manager &4 detected an inconsistency in the context data for agent process &3.**Explanation:**

The EC Boss found that the information it had previously held about the agent &3 is not consistent with new information.

User action:

Examine the log to see if further information is available relating to process &3.

AMQ8825 EC number &1 in Queue Manager &4 detected the failure of the EC Boss.**Explanation:**

An EC detected that the EC Boss process for the Queue Manager has failed. If the maximum number of restarts for the EC Boss has not been exceeded, PATHWAY will attempt to restart the EC Boss.

User action:

Examine the log to see if further information is available relating to the failure of the EC Boss. If the problem persists, end the Queue Manager correct the problem and restart. If the problem cannot be identified as a configuration problem, use the standard facilities supplied with your system to record the problem identifier, and save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8826 EC number &1 in Queue Manager &4 detected the failure of an &5 agent with process name &3.**Explanation:**

An EC detected that an &5 agent process &3 has failed. If the maximum number of restarts of agent processes has not already been exceeded, the EC will attempt to restart the agent process when it is required.

User action:

Examine the log to see if further information is available relating to the failure of the agent process. If the problem persists, end the Queue Manager correct the problem and restart. If the problem cannot be identified as a configuration problem, use the standard facilities supplied with your system to record the problem identifier, and save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8827 EC number &1 in Queue Manager &4 failed to communicate with the EC Boss.

Explanation:

An EC attempted to communicate with the EC Boss, but the attempt failed. The failure to communicate is interpreted by the EC as EC Boss failure.

User action:

Examine the log to see if further information is available relating to the failure to communicate with the EC Boss. If the problem persists, end the Queue Manager correct the problem and restart. If the problem cannot be identified as a configuration problem, use the standard facilities supplied with your system to record the problem identifier, and save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8828 EC number &1 in Queue Manager &4 failed to communicate with &5 agent process &3.

Explanation:

An EC attempted to communicate with an agent process, but the attempt failed. The failure to communicate is interpreted by the EC as agent process failure. Depending on various factors, the EC may attempt to restart the agent.

User action:

Examine the log to see if further information is available relating to the failure to communicate with the agent. If the problem persists, end the Queue Manager correct the problem and restart. If the problem cannot be identified as a configuration problem, use the standard facilities supplied with your system to record the problem identifier, and save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8829 EC number &1 in Queue Manager &4 failed to start an &5 agent.

Explanation:

An EC attempted to create an agent process, but the attempt failed. If the maximum number of agent restarts has not already been exceeded, the EC will attempt to restart the agent process.

User action:

Examine the log to see if further information is available relating to the failure to start the agent. If the problem persists, end the Queue Manager correct the problem and restart. If the problem cannot be identified as a configuration problem, use the standard facilities supplied with your system to record the problem identifier, and save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8830 EC number &1 in Queue Manager &4 failed to service a stop channel agent request for channel &5.

Explanation:

An EC attempted to process a stop channel request, but the attempt failed.

The failure will be relayed back to the original requestor via the EC Boss.

User action:

Examine the log to see if further information is available relating to the failure to service the stop channel request. The originator of the stop channel request will be informed of the failure, together with the reason for the failure.

AMQ8831 EC number &1 in Queue Manager &4 failed to service an agent "done" request from agent process &3.

Explanation:

An EC attempted to process an agent "done" request, but the attempt failed. An agent "done" request indicates that agent process &3 has completed its work and is asking the EC whether to terminate, or to go idle. For some reason, the EC failed to process the request. The EC will terminate the agent process.

User action:

Examine the log to see if further information is available relating to the failure to service the agent "done" request.

AMQ8832 EC number &1 in Queue Manager &4 created an idle &5 agent process.

Explanation:

An EC successfully created an idle agent.

User action:

None.

AMQ8833 EC number &1 in Queue Manager &4 failed to deactivate &5 agent process &3.

Explanation:

An EC failed to activate an idle agent in order to service a connection or start channel request. The request could not be satisfied by the EC. The EC returns a failure completion and reason code to the originator of the request.

User action:

Examine the log to see if further information is available relating to the failure to deactivate the agent.

AMQ8834 EC number &1 in Queue Manager &4 failed to activate &5 agent process &3.

Explanation:

An EC failed to deactivate an active agent after the agent indicated that it had completed processing a connection or channel.

User action:

Examine the log to see if further information is available relating to the failure to deactivate the agent.

AMQ8835 EC number &1 in Queue Manager &4 destroyed idle &5 agent process &3.

Explanation:

An EC successfully destroyed an idle agent process. The EC normally performs this operation as a result of managing the pool of idle agents. Agents that have been used more than a certain (configurable) number of times are destroyed and a fresh agent created in their place.

User action:

None.

AMQ8836 EC number &1 in Queue Manager &4 failed to destroy an idle &5 agent process &3.**Explanation:**

An EC failed to destroy an idle agent process. The EC normally performs this operation as a result of managing the pool of idle agents. Agents that have been used more than a certain (configurable) number of times are destroyed and a fresh agent created in their place.

User action:

Examine the log to see if further information is available relating to the failure to destroy the agent.

AMQ8837 EC number &1 in Queue Manager &4 failed to create an idle &5 agent.**Explanation:**

An EC failed to create an idle &5 agent process. The EC normally performs this operation as a result of managing the pool of idle agents. Agents that have been used more than a certain (configurable) number of times are destroyed and a fresh agent created in their place.

User action:

Examine the log to see if further information is available relating to the failure to create the agent.

AMQ8838 EC number &1 in Queue Manager &4 initiated creation of an idle &5 agent.**Explanation:**

An EC successfully initiated the creation of an idle &5 agent process. The EC normally performs this operation as a result of managing the pool of idle agents. Agents that have been used more than a certain (configurable) number of times are destroyed and a fresh agent created in their place.

User action:

None.

AMQ8839 EC number &1 in Queue Manager &4 failed to complete a &3 request for channel &5.**Explanation:**

An EC failed to complete the processing of an &3 request. The originator of the request is passed the completion status and reason code.

User action:

Examine the log to see if further information is available relating to the failure to complete the processing of the request.

AMQ8840 EC number &1 in Queue Manager &4 failed to complete an agent status request for agent process &3.**Explanation:**

An EC failed to complete the processing of an agent status request. The EC Boss or EC has detected an inconsistency in the context information about the agent.

User action:

Examine the log to see if further information is available relating to the failure to complete the processing of the request.

AMQ8841 EC process &3 in Queue Manager &4 is waiting for the EC Boss to initialize.**Explanation:**

An EC is waiting for the EC Boss to initialize and create its entry in the RUNTIME file for the Queue Manager.

User action:

None.

AMQ8842 EC number &1 in Queue Manager &4 failed to auto-start an agent for agent channel &5.**Explanation:**

An EC Failed to auto-start an LU 6.2 Responder MCA agent to service channel &5. The channel is marked to be auto-started in the channel definitions, but the EC failed to start the agent process during initialization.

User action:

Examine the log to see if further information is available relating to the failure to auto-start the agent.

AMQ8843 LU6.2 responder process started for channel 'y'**Explanation:** None**User action:** None**AMQ8844 LU6.2 responder process for channel 'y' ended****Explanation:** None**User action:** None**AMQ8845 MQSeries MQSS Server has restarted its backup process**

Explanation: The MQSeries Status Server &3 detected the failure of its backup process and has restarted a new backup in CPU &1.

User action: Use the standard operating system facilities to diagnose the cause of the backup MQSS Server failure and attempt to correct it. MQSeries will continue without interruption.

Programmer response:

None.

AMQ8846 MQSeries MQSS Server takeover initiated

Explanation: The MQSeries Status Server backup &3 detected the failure of its primary process and is in the process of taking over and starting a new backup. The new MQSS Server primary process is now running in CPU &1.

User action:

Use the standard operating system facilities to diagnose the cause of the primary MQSS Server failure and attempt to correct it. MQSeries will continue without interruption.

Programmer response:

None.

AMQ8850 **WARNING - EXCEPTION DETECTED**
Installation configuration exceeds the recorded license use entitlement. Please see your Program Provider to obtain the required additional use-authorization.

Explanation:

None.

User action:

None.

AMQ8851 **MQSeries cleanrdf utility has detected an error**

Explanation:

cleanrdf (queue manager &5) encountered a(n) &4 error on the rdfpurge file '&3'. The file system returned error code &1.

User action:

Use the standard operating system facilities to verify the state of this file and reinvoke the utility if the error is deemed transient.

Programmer response:

None.

AMQ8852 **MQSeries cleanrdf utility has detected an error**

Explanation: cleanrdf (queue manager &5) has detected that the backup system &4 is inaccessible. The file system returned error code &1.

User action:

Contact your systems administrator and reinvoke the utility if the error is deemed transient.

Programmer response:

None.

AMQ8853 **MQSeries cleanrdf utility has detected an error**

Explanation: cleanrdf (queue manager &5) has encountered a TM/MP &4 error. The system returned error code &1.

User action:

Contact your systems administrator and reinvoke the utility if the error is deemed transient.

Programmer response:

None.

AMQ8854 **MQSeries cleanrdf utility has detected an error**

Explanation:

cleanrdf (queue manager &5) encountered a(n) &4 error on file &3. The system returned error code &1.

User action:

Ensure that a file with this name exists on the same volume and subvolume (create if necessary - format is irrelevant) on both the primary system and backup systems before reinvoking the utility.

Programmer response:

None.

AMQ8855 **MQSeries cleanrdf utility has detected an error**

Explanation: cleanrdf (queue manager &5) encountered a(n) &4 error for the FUP process &3. The system returned error code &1.

User action: Use the standard operating system facilities to verify the MQRDFUPPROGNAME and MQRDFUPPROCESSNAME environment parameters. Reinvoke the utility if the error is deemed transient.

Programmer response:

None.

AMQ8856 **MQSeries cleanrdf utility has detected an error**

Explanation:

cleanrdf (queue manager &5) encountered an error when attempting to duplicate file &3 to backup system &4. The system returned error code &1.

User action:

Use the standard operating system facilities to verify the state of this file on both primary and backup systems. Reinvoke the utility if the error is deemed transient.

Programmer response:

None.

AMQ8857 **MQSeries CleanRDF utility STATISTICS Message**

Explanation:

CleanRDF of queue manager &5 has completed operation. &1 files were Deleted. &2 files were Skipped. &3 static files were duplicated to backup system &4.

User action:

None

Programmer response:

None.

Remote messages

AMQ9001 Channel program ended normally.

Explanation:

Channel program '&3' ended normally.

User action:

None.

AMQ9002 Channel program started.

Explanation:

Channel program '&3' started.

User action:

None.

AMQ9181 The response set by the exit is not valid.

Explanation:

The user exit '&3' returned a response code '&1' that is not valid in the ExitResponse field of the channel exit parameters (MQCXP). Message AMQ9190 is issued giving more details, and the channel stops.

User action:

Investigate why the user exit program set a response code that is not valid.

AMQ9182 The secondary response set by the exit is not valid.

Explanation:

The user exit '&3' returned a secondary response code '&1' in the ExitResponse2 field of the channel exit parameters (MQCXP) that is not valid. Message AMQ9190 is issued giving more details, and the channel stops.

User action:

Investigate why the user exit program set a secondary response code that is not valid.

AMQ9184 The exit buffer address set by the exit is not valid.

Explanation:

The user exit '&3' returned an address '&1' for the exit buffer that is not valid, when the secondary response code in the ExitResponse2 field of the channel exit parameters (MQCXP) is set to MQXR2_USE_EXIT_BUFFER. Message AMQ9190 is issued giving more details, and the channel stops.

User action:

Investigate why the user exit program set an exit buffer address that is not valid. The most likely cause is the failure to set a value, so that the value is 0.

AMQ9189 The data length set by the exit is not valid.

Explanation:

The user exit '&3' returned a data length value '&1' that was not greater than zero. Message AMQ9190 is issued giving more details, and the channel stops.

User action:

Investigate why the user exit program set a data length that is not valid.

AMQ9190 Channel stopping because of an error in the exit.

Explanation:

The user exit '&3', invoked for channel '&4' with id '&1' and reason '&2', returned values that are not valid, as reported in the preceding messages. The channel stops.

User action:

Investigate why the user exit program set values that are not valid.

AMQ9196 Data length is larger than the agent buffer length.

Explanation:

The data length '&1' set by exit '&3' is larger than the agent buffer length. The user exit returned data in the supplied agent buffer, but the length specified is greater than the length of the buffer. Message AMQ9190 is issued giving more details, and the channel stops.

User action:

Investigate why the user exit program set a data length that is not valid..

AMQ9197 Data length is larger than the exit buffer length.

Explanation:

The data length '&1' set by exit '&3' is larger than the exit buffer length. The user exit returned data in the supplied exit buffer, but the length specified is greater than the length of the buffer. Message AMQ9190 is issued giving more details, and the channel stops.

User action:

Investigate why the user exit program set a data length that is not valid.

AMQ9201 Allocate failed to host '&3'.

Explanation:

The attempt to allocate a conversation using &4 to host '&3' was not successful.

User action:

The error may be due to an incorrect entry in the &4 parameters contained in the channel definition to host '&3'. Correct the error and try again. If the error persists, record the error values and contact your systems administrator. The return code from &4 was &1 (X'&2'). It may be possible that the listening program at host '&3' is not running. If this is the case, perform the relevant operations to start the listening program for protocol &4 and try again.

AMQ9202 Remote host '&3' not available, retry later.

Explanation:

The attempt to allocate a conversation using &4 to host '&3' was not successful. However the error may be a transitory one and it may be possible to successfully allocate a &4 conversation later.

User action:

Try the connection again later. If the failure persists, record the error values and contact your systems administrator. The return code from &4 is &1 (X'&2'). The reason for the failure may be that this host cannot reach the destination host. It may also be possible that the listening program at host '&3'

was not running. If this is the case, perform the relevant operations to start the &4 listening program, and try again.

AMQ9203 A configuration error for &4 occurred.**Explanation:**

Allocation of a &4 conversation to host '&3' was not possible.

User action:

The configuration error may be one of the following: 1. If the communications protocol is LU6.2, it may be that one of the transmission parameters (Mode, or TP Name) is incorrect. Correct the error and try again. The mode name should be the same as the mode defined on host &3. The TP name on &3 should be defined. 2. If the communications protocol is LU6.2, it may be that an LU6.2 session has not been established. Contact your systems administrator. 3. If the communications protocol is TCP/IP, it may be that the host name specified is incorrect. Correct the error and try again. 4. If the communications protocol is TCP/IP, it may be that the host name specified cannot be resolved to a network address. The host name may not be in the name server. The return code from &4 is &1 (X'&2'). Record the error values and tell the system administrator.

AMQ9204 Connection to host '&3' rejected.**Explanation:**

Connection to host '&3' over &4 was rejected.

User action:

The remote system might not be configured to allow connections from this host. Check the &4 listener program has been started on host '&3'. If the conversation uses LU6.2, it is possible that either the user ID or password supplied to the remote host is incorrect. If the conversation uses TCP/IP, it is possible that the remote host does not recognize the local host as a valid host. The return code from &4 is &1 X('&2'). Record the values and tell the systems administrator.

AMQ9205 The host name supplied is not valid.**Explanation:**

The supplied &4 host name '&3' could not be resolved into a network address. Either the name server does not contain the host, or the name server was not available.

User action:

Check the &4 configuration on your host.

AMQ9206 Error on send to host '&3'.**Explanation:**

An error occurred sending data over &4 to '&3'. This may be due to a communications failure.

User action:

Record the value &1 and the return code &4 and tell your systems administrator.

AMQ9207 The data received from host '&3' is not valid.**Explanation:**

Incorrect data format received from host '&3' over &4. It may be that an unknown host is attempting to send data. An FFST file has been generated containing the invalid data received.

User action:

Tell the systems administrator.

AMQ9208 Error on receive from host '&3'.**Explanation:**

An error occurred receiving data from '&3' over &4. This may be due to a communications failure.

User action:

Record the &4 return code &1 (X'&2') and tell the systems administrator.

AMQ9209 Connection to host '&3' closed.**Explanation:**

An error occurred receiving data from '&3' over &4. The connection to the remote host has unexpectedly terminated.

User action:

Tell the systems administrator.

AMQ9210 Remote attach failed.**Explanation:**

There was an incoming attach from a remote host but the local host could not complete the bind.

User action:

Record the &4 return code &1 (X'&2') and tell the systems administrator who should check the &4 configuration.

AMQ9211 Error allocating storage.**Explanation:**

The program was unable to obtain enough storage.

User action:

Stop some programs which are using storage and retry the operation. If the problem persists contact your Systems Administrator.

AMQ9212 A TCP/IP socket could not be allocated.**Explanation:**

A TCP/IP socket could not be created, possibly because of a storage problem.

User action:

Try the program again. If the failure persists record the value &1 and tell the systems administrator.

AMQ9213 A communications error for &4 occurred.**Explanation:**

An unexpected error occurred in communications.

User action:

The return code from the &4&3 call was &1 (X'&2'). Record these values and tell the systems administrator.

AMQ9214 Attempt to use an unsupported communications protocol.**Explanation:**

An attempt was made to use an unsupported communications protocol type &2.

User action:

Check the channel definition file. It may be that the communications protocol entered is not a currently supported one.

AMQ9215 Communications subsystem unavailable.**Explanation:**

An attempt was made to use the communications subsystem, but it has not been started.

User action:

Start the communications subsystem, and rerun the program.

AMQ9216 Usage: &3 [-m QMgrName] [-n TPName]**Explanation:**

Values passed to the responder channel program are not valid. The parameter string passed to this program is as follows :- [-m QMgrName] [-n TPName] Default values will be used for parameters not supplied.

User action:

Correct the parameters passed to the Channel program and retry the operation.

AMQ9217 The TCP/IP listener program could not be started.**Explanation:**

An attempt was made to start a new instance of the listener program, but the program was rejected.

User action:

The failure could be because either the subsystem has not been started (in this case you should start the subsystem), or there are too many programs waiting (in this case you should try to start the listener program later).

AMQ9218 The TCP/IP listener program could not bind to port number &1.**Explanation:**

An attempt to bind the TCP/IP socket to the listener port was unsuccessful.

User action:

The failure could be due to another program using the same port number. Record the return code &2 from the bind and tell the systems administrator.

AMQ9219 The TCP/IP listener program could not create a new connection for the incoming conversation.**Explanation:**

An attempt was made to create a new socket because an attach request was received, but an error occurred.

User action:

The failure may be transitory, try again later. If the problem persists, record the return code &1 and tell the systems administrator. It may be necessary to free some jobs, or restart the communications system.

AMQ9220 The &4 communications program could not be loaded.**Explanation:**

The attempt to load the &4 library or procedure '&3' failed with error code &1.

User action:

Either the library must be installed on the system or the environment changed to allow the program to locate it.

AMQ9221 Unrecognized protocol was specified.**Explanation:**

The specified value of '&3' was not recognized as one of the protocols supported.

User action:

Correct the parameter and retry the operation.

AMQ9222 Cannot find the configuration file.**Explanation:**

The configuration file '&3' cannot be found. This file contains default definitions for communication parameters. Default values will be used.

User action:

None.

AMQ9223 Enter a protocol type.**Explanation:**

The operation you are performing requires that you enter the type of protocol.

User action:

Add the protocol parameter and retry the operation.

AMQ9224 Unexpected token detected.**Explanation:**

On line &1 of the INI file keyword '&3' was read when a keyword was expected.

User action:

Correct the file and retry the operation.

AMQ9225 File syntax error.**Explanation:**

A syntax error was detected on line &1 while processing the INI file.

User action:

Correct the problem and retry the operation.

AMQ9226 Usage: &3 [-m QMgrName] -t (TCP | LU62 | NETBIOS) [ProtocolOptions]**Explanation:**

Values passed to the listener program were invalid. The parameter string passed to this program is as follows :- [-m QMgrName] (-t TCP [-p Port] | -t LU62 [-n TPName] | -t NETBIOS [-l LocalName] [-e Names] [-s Sessions] [-o Commands] [-a Adaptor]) Default values will be used for parameters not supplied.

User action:

Correct the parameters passed to the listener program and retry the operation.

AMQ9227 &3 local host name not provided.**Explanation:****User action:**

Add a local name to the configuration file and retry the operation.

AMQ9228 The &4 responder program could not be started.

Explanation:

An attempt was made to start an instance of the responder program, but the program was rejected.

User action:

The failure could be because either the subsystem has not been started (in this case you should start the subsystem), or there are too many programs waiting (in this case you should try to start the responder program later).

AMQ9229 The application has been ended.

Explanation:

You have issued a request to end the application.

User action:

None.

AMQ9230 An unexpected &4 event occurred.

Explanation:

During the processing of network events, an unexpected event &1 occurred.

User action:

None.

AMQ9231 The supplied parameter is not valid.

Explanation:

The value of the &4 &5 parameter has the value '&3'. This value has either not been specified or has been specified incorrectly.

User action:

Check value of the &5 parameter and correct it if necessary. If the fault persists, record the return code (&1,&2) and &4 and tell the systems administrator.

AMQ9232 No &3 specified

Explanation:

The operation requires the specification of the &3 field.

User action:

Specify the &3 and retry the operation.

AMQ9233 Error creating Listener thread for &3.

Explanation:

The process attempted to create a new thread for an incoming connection.

User action:

Contact the systems administrator.

AMQ9235 The supplied Local LU was invalid.

Explanation:

The &4 Local LU name '&3' was invalid.

User action:

Either the Local LU name was entered incorrectly or it was not in the &4 communications configuration. Correct the error and try again.

AMQ9236 The supplied Partner LU was invalid.

Explanation:

The &4 Partner LU name '&3' was invalid.

User action:

Either the Partner LU name was entered incorrectly or it was not in the &4 communications configuration. Correct the error and try again.

AMQ9240 A communications error for SNAX occurred.

Explanation:

An unexpected error occurred in communications. The reply code return code from the SNAX APC &3 request was &1 in the &4 header. The detail return code was &2.

User action:

Record the error values and tell the systems administrator.

AMQ9501 Usage: &3 [-m QMgrName] -c ChName.

Explanation:

Values passed to the channel program are not valid. The parameter string passed to this program is as follows :- [-m QMgrName] -c ChName Default values will be used for parameters not supplied.

User action:

Correct the parameters passed to the Channel program and retry the operation.

AMQ9502 Type of channel not suitable for action requested.

Explanation:

The operation requested cannot be performed on channel '&3'. Some operations are only valid for certain channel types. For example, you can only ping a channel from the end sending the message.

User action:

Check whether the channel name is specified correctly. If it is check that the channel has been defined correctly.

AMQ9503 Channel negotiation failed.

Explanation:

Channel '&3' between this machine and the remote machine could not be established due to a negotiation failure.

User action:

Tell the systems administrator who should look at the log on the remote system for messages explaining the cause of the negotiation failure.

AMQ9504 A protocol error was detected for channel '&3'.

Explanation:

During communications with the remote queue manager, the channel program detected a protocol error. The failure type was &1 with associated data of &2.

User action:

Contact the systems administrator who should examine the error logs to determine the cause of the failure.

AMQ9505 Channel sequence number wrap values are different.**Explanation:**

The sequence number for channel '&3' is &1, but the value specified at the remote location is &2. The two values must be the same before the channel can be started.

User action:

Change either the local or remote channel definitions so that the values specified for the message sequence number wrap values are the same.

AMQ9506 Message receipt confirmation failed.**Explanation:**

Channel '&3' has ended because the remote queue manager did not accept the last batch of messages.

User action:

The error log for the channel at the remote site will contain an explanation of the failure. Contact the remote Systems Administrator to resolve the problem.

AMQ9507 Channel '&3' is currently in-doubt.**Explanation:**

The requested operation cannot complete because the channel is in-doubt with host '&4'.

User action:

Examine the status of the channel, and either restart a channel to resolve the in-doubt state, or use the RESOLVE CHANNEL command to correct the problem manually.

AMQ9508 Program cannot connect to the queue manager.**Explanation:**

The connection attempt to queue manager '&4' failed with reason code &1.

User action:

Ensure that the queue manager is available and operational.

AMQ9509 Program cannot open queue manager object.**Explanation:**

The attempt to open either the queue or queue manager object '&4' on queue manager '&5' failed with reason code &1.

User action:

Ensure that the queue is available and retry the operation.

AMQ9510 Messages cannot be retrieved from a queue.**Explanation:**

The attempt to get messages from queue '&4' on queue manager '&5' failed with reason code &1.

User action:

Ensure that the required queue is available and operational.

AMQ9511 Messages cannot be put to a queue.**Explanation:**

The attempt to put messages to queue '&4' on queue manager '&5' failed with reason code &1.

User action:

Ensure that the required queue is available and operational.

AMQ9512 Ping operation is not valid for channel '&3'.**Explanation:**

Ping may only be issued for SENDER or SERVER channel types.

User action:

If the local channel is a receiver channel, you must issue the ping from the remote queue manager.

AMQ9513 Maximum number of channels reached.**Explanation:**

The maximum number of channels that can be in use simultaneously has been reached.

User action:

Either wait for some of the operating channels to close or use the stop channel command to close some channels. Retry the operation when some channels are available. The number of permitted channels is a configurable parameter in the queue manager configuration file.

AMQ9514 Channel '&3' is in use.**Explanation:**

The requested operation failed because channel '&3' is currently active.

User action:

Either end the channel manually, or wait for it to close, and retry the operation.

AMQ9515 Channel '&3' changed.**Explanation:**

The statistics shown are for the channel requested, but it is a new instance of the channel. The previous channel instance has ended.

User action:

None.

AMQ9516 File error occurred.**Explanation:**

The filesystem returned error code &1 for file '&3'.

User action:

Record the name of the file '&3' and tell the systems administrator, who should ensure that file '&3' is correct and available.

AMQ9517 File damaged.**Explanation:**

The program has detected damage to the contents of file '&3'.

User action:

Record the values and tell the systems administrator who must restore a saved version of file '&3'. The return code was '&1' and the record length returned was '&2'.

AMQ9518 File '&3' not found.

Explanation:

The program requires that the file '&3' is present and available.

User action:

Record the name of the file and tell the systems administrator who must ensure that file '&3' is available to the program.

AMQ9519 Channel '&3' not found.

Explanation:

The requested operation failed because the program could not find a definition of channel '&3'.

User action:

Check that the name is specified correctly and the channel definition is available.

AMQ9520 Channel not defined remotely.

Explanation:

There is no definition of channel '&3' at the remote location.

User action:

Add an appropriate definition to the remote hosts list of defined channels and retry the operation.

AMQ9521 Host is not supported by this channel.

Explanation:

The connection across channel '&5' was refused because the remote host '&4' did not match the host '&3' specified in the channel definition.

User action:

Update the channel definition, or remove the explicit mention of the remote machine connection name.

AMQ9522 Error accessing the status table.

Explanation:

The program could not access the channel status table.

User action:

None.

AMQ9523 Remote host detected a protocol error.

Explanation:

During communications through channel '&3', the remote queue manager channel program detected a protocol error. The failure type was &1 with associated data of &2.

User action:

Tell the systems administrator, who should examine the error files to determine the cause of the failure.

AMQ9524 Remote queue manager unavailable.

Explanation:

Channel '&3' cannot start because the remote queue manager is not currently available.

User action:

Either start the remote queue manager, or retry the operation later.

AMQ9525 Remote queue manager is ending.

Explanation:

Channel '&3' is closing because the remote queue manager is ending.

User action:

None.

AMQ9526 Message sequence number error for channel '&3'.

Explanation:

The local and remote queue managers do not agree on the next message sequence number. A message with sequence number &1 has been sent when sequence number &2 was expected.

User action:

Determine the cause of the inconsistency. It could be that the synchronization information has become damaged, or has been backed out to a previous version.

If the situation cannot be resolved, the sequence number can be manually reset at the sending end of the channel using the RESET CHANNEL command.

AMQ9527 Cannot send message through channel '&3'.

Explanation:

The channel has closed because the remote queue manager cannot receive a message.

User action:

Contact the systems administrator who should examine the error files of the remote queue manager, to determine why the message cannot be received, and then restart the channel.

AMQ9528 User requested closure of channel '&3'.

Explanation:

The channel is closing because of a request by the user.

User action:

None.

AMQ9529 Target queue unknown on remote host.

Explanation:

Communication using channel '&3' has ended because the target queue for a message is unknown at the remote host.

User action:

Ensure that the remote host contains a correctly defined target queue, and restart the channel.

AMQ9530 Program could not inquire queue attributes.

Explanation:

The attempt to inquire the attributes of queue '&4' on queue manager '&5' failed with reason code &1.

User action:

Ensure that the queue is available and retry the operation.

AMQ9531 Transmission queue specification error.**Explanation:**

Queue '&4' identified as a transmission queue in the channel definition '&3' is not a transmission queue.

User action:

Ensure that the queue name is specified correctly. If so, alter the queue usage parameter of the queue to that of a transmission queue.

AMQ9532 Program cannot set queue attributes.**Explanation:**

The attempt to set the attributes of queue '&4' on queue manager '&5' failed with reason code &1.

User action:

Ensure that the queue is available and retry the operation.

AMQ9533 Channel '&3' is not currently active.**Explanation:**

The channel was not stopped because it was not currently active.

User action:

None.

AMQ9534 Channel '&3' is currently not enabled.**Explanation:**

The channel program ended because the channel is currently not enabled.

User action:

Issue the START CHANNEL command to re-enable the channel.

AMQ9535 User exit not valid.**Explanation:**

Channel program '&3' ended because user exit '&4' is not valid.

User action:

Ensure that the user exit is specified correctly in the channel definition, and that the user exit program is correct and available.

AMQ9536 Channel ended by an exit.**Explanation:**

Channel program '&3' was ended by exit '&4'.

User action:

None.

AMQ9537 Usage: &3 [-m QMgrName] [-q InitQ]**Explanation:**

Values passed to the Channel initiator program are not valid. The parameter string passed to this program is as follows :- [-m QMgrName] [-q InitQ] Default values will be used for parameters not supplied.

User action:

Correct the parameters passed to the program and retry the operation.

AMQ9538 Commit control error.**Explanation:**

An error occurred when attempting to start commitment control. Either exception '&3' was received when querying commitment status, or commitment control could not be started.

User action:

Refer to the error log for other messages pertaining to this problem.

AMQ9539 No channels available.**Explanation:**

The channel initiator program received a trigger message to start an MCA program to process queue '&3'. The program could not find a defined, available channel to start.

User action:

Ensure that there is a defined channel, which is enabled, to process the transmission queue.

AMQ9540 Commit failed.**Explanation:**

The program ended because return code &1 was received when an attempt was made to commit change to the resource managers. The commit ID was '&3'.

User action:

Tell the systems administrator.

AMQ9541 CCSID supplied for data conversion not supported.**Explanation:**

The program ended because, either the source CCSID '&1' or the target CCSID '&2' is not valid, or is not currently supported.

User action:

Correct the CCSID that is not valid, or ensure that the requested CCSID can be supported.

AMQ9542 Queue manager is ending.**Explanation:**

The program will end because the queue manager is quiescing.

User action:

None.

AMQ9543 Status table damaged.**Explanation:**

The channel status table has been damaged.

User action:

End all running channels and issue a DISPLAY CHSTATUS command to see the status of the channels. Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9544 Messages written to the 'dead-letter queue'.**Explanation:**

During the processing of channel '&3' one or more messages have been put to a dead-letter queue. The location of the messages is &1, where 1 is the local dead-letter queue and 2 is the remote dead-letter queue.

User action:

Examine the contents of the dead-letter queue. Each message is contained in a structure that describes why the message was put to the queue, and to where it was originally addressed. The program identifier (PID) of the processing program was '&4'.

AMQ9545 Disconnect interval expired.**Explanation:**

Channel '&3' closed because no messages arrived on the transmission queue within the disconnect interval period.

User action:

None.

AMQ9546 Error return code received.**Explanation:**

The program has ended because return code &1 was returned from an internal function.

User action:

Correct the reason for the failure and retry the operation.

AMQ9547 Type of remote channel not suitable for action requested.**Explanation:**

The operation requested cannot be performed because channel '&3' on the remote machine is not of a suitable type. For example, if the local channel is defined as a sender the remote machine must define its channel as either a receiver or requester.

User action:

Check that the channel name is specified correctly. If it is, check that the remote channel has been defined correctly.

AMQ9548 Message put to the 'dead-letter queue'.**Explanation:**

During processing a message has been put to the dead-letter queue.

User action:

Examine the contents of the dead-letter queue. Each message is contained in a structure that describes why the message was put to the queue, and to where it was originally addressed.

AMQ9549 Transmission Queue '&3' inhibited for MQGET.**Explanation:**

An MQGET failed because the transmission queue had been previously inhibited for MQGET.

User action:

None.

AMQ9550 Channel program &3 cannot be stopped at this time.**Explanation:**

The channel program is currently busy and cannot be stopped at the moment.

User action:

Issue the STOP CHANNEL command again at a later time.

AMQ9551 Protocol not supported by remote host**Explanation:**

The operation you are performing over Channel '&3' to the host at '&4' is not supported by the target host.

User action:

Check that the connection name parameter is specified correctly and that the levels of the products in use are compatible.

AMQ9552 Security flow not received.**Explanation:**

During communications through channel '&3' the local security exit requested security data from the remote machine. The security data has not been received so the channel has been closed.

User action:

Tell the systems administrator who should ensure that the security exit on the remote machine is defined correctly.

AMQ9553 Not supported.**Explanation:**

The command or function attempted is not currently supported on this platform.

User action:

None.

AMQ9554 User not authorized.**Explanation:**

You are not authorized to perform the Channel operation.

User action:

Tell the systems administrator who should ensure that the correct access permissions are available to you, and then retry the operation.

AMQ9555 File format error.**Explanation:**

The file '&3' does not have the expected format.

User action:

Ensure that the file name is specified correctly.

AMQ9556 Channel synchronization file missing or damaged.**Explanation:**

The channel synchronization file '&3' is missing or does not correspond to the stored channel information for queue manager '&4'.

User action:

Rebuild the synchronization file using the rcrmqobj command
rcrmqobj -t syncfile (-m q-mgr-name)

AMQ9557 Queue Manager User ID initialization failed.**Explanation:**

The call to initialize the user ID failed with CompCode &1 and Reason &2.

User action:

Correct the error and try again.

AMQ9558 Remote Channel is not currently available.**Explanation:**

The channel program ended because the channel '&3' is not currently available on the remote system. This could be because the channel is disabled or that the remote system does not have sufficient resources to run a further channel.

User action:

Check the remote system to ensure that the channel is available to run and retry the operation.

AMQ9560 Rebuild Synchronization File - program started**Explanation:**

Rebuilding the Synchronization file for Queue Manager '&3'.

User action:

None.

AMQ9561 Rebuild Synchronization File - program completed normally**Explanation:**

Rebuild Synchronization File program completed normally.

User action:

None.

AMQ9562 Synchronization file in use.**Explanation:**

The Synchronization file '&3' is in use and cannot be recreated.

User action:

Stop any channel activity and retry the rcrmjob command.

AMQ9563 Synchronization file cannot be deleted**Explanation:**

The filesystem returned error code &1 for file '&3'.

User action:

Tell the systems administrator who should ensure that file '&3' is available and not in use.

AMQ9564 Synchronization File cannot be created**Explanation:**

The filesystem returned error code &1 for file '&3'.

User action:

Tell the systems administrator.

AMQ9565 No dead-letter queue defined.**Explanation:**

The queue manager '&4' does not have a defined dead-letter queue.

User action:

Either correct the problem that caused the program to try and write a message to the dead-letter queue or create a dead-letter queue for the queue manager.

AMQ9566 Invalid MQSERVER value**Explanation:**

The value of the MQSERVER environment variable was '&3'. The variable should be in the format 'ChannelName/Protocol/ConnectionName'.

User action:

Correct the MQSERVER value and retry the operation.

AMQ9572 Message header is not valid.**Explanation:**

Channel '&3' is stopping because a message header is not valid. During the processing of the channel, a message was found that has a header that is not valid. The dead-letter queue has been defined as a transmission queue, so a loop would be created if the message had been put there.

User action:

Correct the problem that caused the message to have a header that is not valid.

AMQ9573 Maximum number of active channels reached.**Explanation:**

There are too many channels active to start another. The current defined maximum number of active channels is &1.

User action:

Either wait for some of the operating channels to close or use the stop channel command to close some channels. Retry the operation when some channels are available. The maximum number of active channels is a configurable parameter in the queue manager configuration file.

AMQ9574 Channel &3 can now be started.**Explanation:**

Channel &3 has been waiting to start, but there were no channels available because the maximum number of active channels was running. One, or more, of the active channels has now closed so this channel can start.

User action:

None.

AMQ9600 The TCP Listener &3 in Queue Manager &4 cannot find an available port.**Explanation:**

The TCP listener has tried all the ports that are configured in the QMINI file for this Queue Manager, and none were available for listening on. The TCP Listener has now terminated. The TCP Listener is either not needed (because there are already TCP Listeners running on all the Queue Manager ports), or there is a configuration problem with the Queue Manager.

User action:

AMQ9712 • AMQ9999

Review the QMINI file TCP/IP Listener stanzas to determine if there is a configuration problem. The port numbers themselves may be incorrect, or overlap with ports being used by other Queue Managers on the same system, or with other services.

AMQ9712 MQSeries Clean Queue Manager started

Explanation: None

User action: None

AMQ9713 MQSeries Clean Queue Manager complete

Explanation: None

User action: None

AMQ9999 Channel program ended abnormally.

Explanation:

Channel program '&3' ended abnormally.

User action:

Look at previous error messages for channel program '&3' in the error files to determine the cause of the failure.

Appendix O. 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.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States, or other countries, or both:

AIX	AS/400
BookManager	CICS
FFST	First Failure Support Technology
IBM	IBMLink
MQ	MQSeries
MVS/ESA	OS/2
OS/390	VSE/ESA
VTAM	

Lotus and LotusScript 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 of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Part 4. Glossary and index

Glossary of terms and abbreviations

This glossary defines MQSeries terms and abbreviations used in this book. If you cannot find a particular term, 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 can 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

administration commands. MQSeries commands used to manage MQSeries objects, such as queues, processes, and namelists.

alert. A message sent to a management services focal point in a network to identify a problem or an impending problem.

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.

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.

APC. Advanced Program Communication.

application queue. A queue used by an application.

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 you attempt to open an MQSeries object.

authorization file. A file that provides security definitions for an object, a class of objects, or all classes of object.

authorized program analysis report (APAR). A report of a problem caused by a suspected defect in a current, unaltered release of a program.

B

back out. 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.

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.

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.

C

call back. 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). 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 • distributed application

channel event. An event that indicates that a channel instance has become available or unavailable. Channel events are generated on the queue managers at both ends of the channel.

checkpoint. (1) A time when significant information is written on the log. Contrast with *syncpoint*. (2) In MQSeries on UNIX systems, the point in time when a data record described in the log is the same as the data record in the queue. Checkpoints are generated automatically and are used during the system restart process.

CICS. Customer Information Control System.

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 *MQI 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 MQI client. See also *server connection channel type*.

coded character set identifier (CCSID). The name of a coded set of characters and their code point assignments.

command. In MQSeries, an instruction that can be carried out by the queue manager.

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. The act of completing a transaction so that changes to the database are recorded and stable. Protected resources are released after the transaction is committed.

Common Run-Time Environment (CRE). A set of services that enable system and application programmers to write mixed-language programs. These shared, run-time services can be used by C, COBOL85, FORTRAN, Pascal, and TAL programs.

completion code. A return code indicating how an MQI call has ended.

configuration file (also known as ini file). A file that contains configuration information related to logs, communications, or installable services. See also *stanza*.

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. A method of allowing security to be handled such that messages are obliged to carry details of their origins in the message descriptor.

controlled shutdown. See *quiesced shutdown*.

CRE. Common Run-Time Environment.

Customer Information Control System (CICS). An IBM transaction management system that provides concurrent online access to data files by means of user-written application programs. CICS also includes facilities for building, using, and maintaining databases.

D

data conversion interface (DCI). The MQSeries interface to which customer- or vendor-written programs that convert application data between different machine encodings and CCSIDs must conform. A part of the MQSeries Framework.

datagram. The simplest message that MQSeries supports. This type of message does not require a reply.

DCE. Distributed Computing Environment.

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.

dead-letter queue handler. An MQSeries-supplied utility that monitors a dead-letter queue (DLQ) and processes messages on the queue in accordance with a user-written rules table.

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.

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 Computing Environment (DCE).

Middleware that provides basic services, making the development of distributed applications easier. DCE is defined by the Open Software Foundation (OSF).

distributed queue management. In message queuing, the setup and control of message channels to queue managers on other systems.

DLQ (dead-letter queue). A queue to which a queue manager or application sends messages that it cannot deliver to their correct destination.

dynamic queue. A local queue that is created when a program opens a model queue object. See also *permanent dynamic queue* and *temporary dynamic queue*.

E

EC. EC is a subsidiary controlling process in the queue manager, responsible for a set of agents.

EC Boss. The Execution Controller Boss is the main controlling process in the queue manager.

EMS. Event Monitoring System.

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.

F

FFST. First Failure Support Technology.

FIFO (first-in-first-out). A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time.

First Failure Support Technology (FFST). Used by MQSeries on UNIX systems, MQSeries for OS/2, MQSeries for Windows NT, and MQSeries for AS/400 to detect and report software problems.

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.

Framework. In MQSeries, a collection of programming interfaces that allow customers or vendors to write programs that extend or replace certain functions provided in MQSeries products. The interfaces are:

- MQSeries data conversion interface (DCI)
- MQSeries message channel interface (MCI)
- MQSeries name service interface (NSI)
- MQSeries security enabling interface (SEI)
- MQSeries trigger monitor interface (TMI)

G

get. In message queuing, to use the MQGET call to remove a message from a queue. See also *browse*.

H

handle. See *connection handle* and *object handle*.

I

ICE. Intersystem Communications Environment is a family of Tandem-based software products that enables you to access a variety of applications on Tandem computers.

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*.

ini file. See *configuration file*.

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

input parameter • message queue management

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.

installable services. In MQSeries on UNIX systems, MQSeries for Tandem, MQSeries for OS/2, and MQSeries for Windows NT, additional functionality provided as independent components. The installation of each component is optional: in-house or third-party components can be used instead. See also *authorization service*, *name service*, and *user identifier service*.

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.

L

linear logging. In MQSeries on UNIX systems, MQSeries for OS/2, and MQSeries for Windows NT, the process of keeping restart data in a sequence of files. New files are added to the sequence as necessary. The space in which the data is written is not reused until the queue manager is restarted. Contrast with *circular logging*.

listener. In MQSeries distributed queuing, a program that monitors information about incoming network connections.

local definition. An MQSeries object that belongs to a local queue manager.

local definition of a remote queue. An MQSeries object that belongs 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, records the work done by queue managers while they receive, transmit, and deliver messages.

logical unit of work (LUW). See *unit of work*.

MCA (message channel agent). A program that transmits prepared messages from a transmission queue to a communication link, or from a communication link to a destination queue.

MCI (message channel interface). 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. In message queuing applications, a communication sent between programs. See also *persistent message* and *nonpersistent message*. 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 and a receiver) 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.

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 lets application programs access message queuing services.

message queue management. The Message Queue Management (MQM) facility in MQSeries for Tandem

NSK V2.2.0.1 uses PCF command formats and control commands. MQM runs as a PATHWAY SCOBOL requester under the Terminal Control Process (TCP) and uses an MQM SERVERCLASS server, which invokes the C language API to perform PCF commands. There is a separate instance of MQM for each queue manager configured on a system, since each queue manager is controlled under its own PATHWAY configuration. Consequently, an MQM is limited to the management of the queue manager to which it belongs.

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 queuing interface). The programming interface provided by the MQSeries queue managers. This programming interface lets application programs access message queuing services.

MQI channel. Connects an MQI client to a queue manager on a server system, and transfers only MQI calls and responses in a bidirectional manner. Contrast with *message channel*.

MQM. Message Queue Management.

MQI client. Part of an MQSeries product that can be installed on a system without installing the full queue manager. The MQI client accepts MQI calls from applications and communicates with a queue manager on a server system.

MQI server. An MQI server is a queue manager that provides queuing services to one or more clients. All the MQSeries objects, for example queues, exist only on the queue manager system, that is, on the MQI server machine. A server can support normal local MQI applications as well.

MQSC. MQSeries commands.

MQSeries. A family of IBM licensed programs that provides message queuing services.

MQSeries commands (MQSC). Human readable commands, uniform across all platforms, that are used to manipulate MQSeries objects. Contrast with *programmable command format (PCF)*.

N

name service interface (NSI). The MQSeries interface to which customer- or vendor-written programs that resolve queue-name ownership must conform. A part of the MQSeries Framework.

nonpersistent message. A message that does not survive a restart of the queue manager. Contrast with *persistent message*.

NSI. Name service interface.

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 namelist (MVS/ESA only), or a channel.

Object authority manager (OAM). In MQSeries on UNIX systems, MQSeries for Tandem, and &qmnt;, the default authorization service for command and object management. The OAM can be replaced by, or run in combination with, a customer-supplied security service.

object descriptor. A data structure that identifies a particular MQSeries object (MQOD). 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.

output parameter. A parameter of an MQI call in which the queue manager returns information when the call completes or fails.

P

PCF. Programmable command format.

PCF command. See *programmable command format*.

pending event. An unscheduled event that occurs as a result of a connect request from a CICS adapter.

performance event. A category of event that indicates a limit condition has occurred.

performance trace • remote queuing

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 is functioning.

platform. In MQSeries, the operating system under which a queue manager is running.

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.

programmable command format (PCF). A type of MQSeries message that is used by:

- User administration applications that put PCF commands onto the system command input queue of a specified queue manager.
- User administration applications, to get the results of a PCF command from a specified queue manager.
- A queue manager, as a notification that an event has occurred.

Contrast with *MQSC*.

program temporary fix (PTF). A solution or by-pass of a problem diagnosed by IBM field engineering as the result of a defect in a current, unaltered release of a program.

PTF. Program temporary fix.

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, an error condition caused by a queue being 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. In MQSeries, a shutdown of a queue manager that allows all connected applications to disconnect. Contrast with *immediate shutdown* and *preemptive 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.

remote queue. A queue that belongs 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 is remote if it is not the queue manager 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.

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.

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 for requesting a reply from another program.

resolution path. The set of queues that are opened when an application specifies an alias or a remote queue on input to the MQOPEN call.

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*.

rules table. A control file containing one or more rules that the dead-letter queue handler applies to messages on the DLQ.

S

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.

shutdown. See *immediate shutdown*, *preemptive shutdown*, and *quiesced shutdown*.

single-phase back out. A method in which an action that is 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.

SNA. Systems Network Architecture.

stanza. A group of lines in a configuration file that assigns a value to a parameter that modifies the

store and forward • utility

behavior of a queue manager, client, or channel. In MQSeries on systems, a configuration (ini) file can contain a number of stanzas.

store and forward. The temporary storing of packets, messages, or frames in a data network before they are retransmitted toward their destination.

symptom string. Diagnostic information displayed in a structured format designed for searching the IBM software support database.

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.

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.

T

TACL. Tandem Advanced Command Language.

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*.

thread. In MQSeries, the lowest level of parallel execution available on an operating system platform.

time-independent messaging. See *asynchronous messaging*.

TMF. Transaction Management Facility.

TMI. Trigger monitor interface.

TM/MP. NonStop Transaction Manager/MP.

tranid. See *transaction identifier*.

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 that lets a queue manager start an application automatically when predetermined conditions on a queue are satisfied.

trigger message. A message that contains information about the program that a trigger monitor is to start.

trigger monitor. A continuously-running application that serves 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

undelivered-message queue. See *dead-letter queue*.

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.

Index

Special Characters

\$SYSTEM, default installation volume 25

Numerics

2058 (return code) 26

A

access to MQSeries, enabling 36
 ACTION keyword, rules table 161
 action keywords, rules table 161
 ADD SERVER, PATHCOM command 48
 adding nondefault MQSS Server 68
 administration
 authorizations 151
 command sets 43
 control commands 43
 MQSeries commands (MQSC) 44
 programmable command format commands (PCF) 44
 local 99
 remote 126
 channels 127
 objects 125
 transmission queues 127
 agent processes 65
 alias queues
 authorizations to 145
 description 7
 aliases
 queue manager 135
 reply-to queues 135
 AllQueueManagers stanza, MQSINI file 184
 alter queue manager attributes 102
 ALTER SERVER, PATHCOM command 49
 alternate user authority 145
 altmqfls command 111, 112, 220
 related commands 222
 altmqusr command 223
 related commands 224
 amqscoma 74, 293
 AMQxxxx messages 363
 APC pathway definition, example 338
 application
 data 4
 design considerations 202
 MQI administration support 99
 programming errors, examples of 198
 time-independent 3
 applications, compiling and binding 318

APPLIDAT keyword, rules table 160
 APPLNAME keyword, rules table 160
 APPLTYPE keyword, rules table 160
 APPLTYPE(NSK), process attribute 298
 attributes of MQSeries objects
 ALL attribute 109
 changing, local queue 111
 default 109
 MQSC and PCFs compared 45
 queue manager
 altering 102
 displaying 101
 queues 7
 audit-trail size, TM/MP 177
 authority
 alternate user 145
 commands 143
 context 146
 events 351
 installable services 143
 set/reset command 265
 Authority stanza, QMINI file 186
 authorization
 administration 151
 dspmqaut command 144
 files 153
 lists 141
 MQI 148
 setmqaut command 144
 user groups 140
 authorization service 36
 configuring 36
 interface 358
 AUTOSTART, channel attribute 298

B

binding 14
 BookManager xvii
 browsing queues 114
 building applications 315

C

case-sensitive control commands 43
 CCSID parameter, crtmqm command 64
 CCSIDs, supported by MQSeries for Tandem
 NSK 281
 changing queue attributes 111
 channel
 attributes 330
 commands 147

Index

- channel (*continued*)
 - configuration 186
 - defining 128
 - description 10, 125
 - escape command authorizations 151
 - events 170, 351
 - exit programs 326, 329
 - remote administration 127
 - run command 258
 - run initiator command 257
 - security 147
 - security requirements 147
 - starting 129
- channel exits 12
- channel initiator disconnect interval 66
- channel synchronization subvolume 72
- ChannelInitiator stanza, QMINI file 186
- Channels menu, MQM 87
- Channels stanza, QMINI file 186
- channels, limiting number of 66
- cleanrdf command 225
- CLEAR QLOCAL MQSC 299
- clearing a local queue 113
- clients 11, 349
- cnv1520 command 226
 - related commands 227
- cnvclchl command 228
- cnvmsgs command 229
 - related commands 231
- command errors 199
- command files 103
- command queue 9
- command server
 - display command 244
 - displaying status 77
 - end command 251
 - remote administration 77
 - start command 271
 - starting 77
 - stopping 77
- command set
 - administration 43
 - comparison 45
- commands
 - comparison of sets 45
 - control 43
 - altnqfls 220
 - altnqusr 223
 - cleanrdf 225
 - cnv1520 226
 - cnvclchl 228
 - cnvmsgs 229
 - crmqcvx 232
 - crmqm 234
 - dltmqm 238
 - dspmqaout 240
 - dspmqcsv 244
 - commands (*continued*)
 - control (*continued*)
 - dspmqfls 245
 - dspmqtrc 248
 - dspmqusr 249
 - endmqcsv 251
 - endmqm 253
 - endmqtrc 255
 - instmqm 256
 - runmqchi 257
 - runmqchl 258
 - runmqdlq 259
 - runmqlsr 260
 - runmqsc 261
 - runmqtrm 264
 - setmqaut 265
 - strmqcsv 271
 - strmqm 272
 - strmqtrc 273
 - upgmqm 276
- MQSC
 - ALTER QLOCAL 111
 - ALTER QREMOTE 134
 - command files 103
 - DEFINE CHANNEL 128
 - DEFINE QALIAS 118
 - DEFINE QLOCAL 110
 - DEFINE QLOCAL LIKE 110
 - DEFINE QLOCAL REPLACE 111
 - DEFINE QMODEL 119
 - DEFINE QREMOTE 132
 - DELETE QLOCAL 114
 - DISPLAY QREMOTE 134
 - using 44
 - verifying 105
 - programmable command format (PCF) 44
 - run DLQ handler (runmqdlq) 157
 - runmqsc 100
 - security commands
 - dspmqaout 144
 - setmqaut 141
 - set/reset authority (setmqaut) 143
 - commit and backout 175
 - communications examples
 - ICE 342
 - SNAX 335
 - TCP/IP 345
 - communications setup 333
 - communications, customizing 37
 - compilation 14
 - compiling and binding applications 318
 - configuration
 - kernel 27
 - name service 39
 - configuration files
 - editing 193

- configuration files (*continued*)
 - MQSeries (MQSINI) 183
 - backing up 61
 - contents 184
 - overview 183
 - path 106
 - overview 183
 - queue manager (QMINI) 140
 - backing up 61
 - contents 185
 - disabling the object authority manager 140
 - stanzas 185
 - Configuration stanza, QMINI file 185
 - configuring MQSS Servers 40
 - contents of
 - MQSINI 184
 - QMINI 185
 - context authority 146
 - control commands 43
 - altnqfls 220
 - altnqusr 223
 - case sensitive 43
 - cleanrdf 225
 - cnv1520 226
 - cnvclchl 228
 - cnvmsgs 229
 - crtmqcvx 232
 - crtmqm 234
 - dltmqm 238
 - dspmqaout 240
 - dspmqcsv 244
 - dspmqfls 245
 - dspmqtrc 248
 - dspmqusr 249
 - endmqcsv 251
 - endmqm 253
 - endmqtrc 255
 - instmqm 256
 - runmqchi 257
 - runmqchl 258
 - runmqdlq 259
 - runmqlsr 260
 - runmqsc 100, 261
 - runmqtrm 264
 - setmqaut 265
 - strmqcsv 271
 - strmqm 272
 - strmqtrc 273
 - upgmqm 276
 - controlled shutdown 74
 - copying a channel 97
 - copying a queue 85
 - Correlld
 - disabling message retrieval by 40
 - performance considerations when using 202
 - creating
 - channel definition 90
 - default objects 74
 - groups 20
 - principals 73
 - process definitions 122
 - queue 83
 - queue manager 59, 72
 - system objects 74
 - transmission queue 134
 - users 20
 - crtmqcvx command 232
 - crtmqm command 234
 - related commands 237
 - current queue depth 110
 - customizing
 - options available 35
 - overview
 - communications 37
 - data conversion 37
 - defining objects 37
- ## D
- data conversion
 - crtmqcvx command 232
 - customizing 37
 - data files subvolume, queue manager 70
 - data-conversion exits 12
 - building 324
 - using 330
 - dead-letter header, MQDLH 157
 - dead-letter queue
 - description 9
 - handler 259
 - specifying 61
 - debugging
 - common programming errors 198
 - preliminary checks 195
 - secondary checks 199, 202
 - default
 - installation volume (\$SYSTEM) 25
 - MQSS Server, name of 15, 68
 - objects 10, 74
 - queue manager 60
 - accidental change 76
 - accidental deletion 235
 - changing 76, 102
 - commands processed 100
 - system objects 293
 - TCP/IP port 66
 - transmission queue 61, 134
 - DefaultPrefix parameter, crtmqm command 62
 - DefaultProcess stanza, QMINI file 185
 - DefaultQueueManager stanza, MQSINI file 184

Index

- defining queues 83
 - deleting
 - channel 93
 - local queue 86, 114
 - queue manager 76, 238
 - DESTQ keyword, rules table 160
 - DESTQM keyword, rules table 160
 - directories, queue manager 145
 - disabling the object authority manager (OAM) 140
 - disk requirements for installation 20
 - display
 - authority command 240
 - channel status 93
 - command server command 244
 - MQSeries files command 245
 - MQSeries formatted trace output command 248
 - process definitions 123
 - queue manager attributes 101
 - status of command server 77
 - distributed queuing
 - dead-letter queue 9
 - incorrect output 205
 - undelivered-message queue 9
 - DLQ handler
 - invoking 157
 - rules table 158
 - dltmqm command 238
 - related commands 239
 - dspmqaout command 240
 - related commands 243
 - using 141, 143
 - dspmqcsv command 244
 - related commands 244
 - dspmqls command 245
 - related commands 247
 - dspmqr command 248
 - related commands 248
 - dspmqr command 249
 - related commands 250
 - dynamic queues 5
 - authorizations to 145
- ## E
- EC Boss, role of 63
 - EC processes, number of 62
 - EC stanza, QMINI file 185
 - ECBoss stanza, QMINI file 185
 - elementary data types, TAL 301
 - EMS event template, MQSeries 359
 - EMS events 171, 209
 - alternative collector, specifying 174
 - default collector 174
 - setting the MQEMSEVENTS PARAM 173
 - writing programs to process 174
 - EMSCollectorName 64
 - enabling
 - instrumentation events 170
 - security 140
 - end MQSeries trace 255
 - ending a queue manager 75
 - ending interactive MQSC commands 101
 - endmqcsv command 251
 - related commands 252
 - endmqm command 74, 253
 - related commands 254
 - endmqtrc command 255
 - related commands 255
 - environment variables 26
 - error log 207
 - error occurring before established 208
 - example 209
 - subvolume 71
 - error messages 100
 - escape PCFs 45
 - euro support 281
 - Event Management Service (EMS) events 171
 - event queue 9
 - event-driven processing 3
 - event-message format 352
 - events
 - channel 170
 - EMS 359
 - instrumentation
 - description 169
 - enabling 170
 - message 171
 - types of 169
 - what they are 169
 - why use them 169
 - queues 170
 - support for in MQSeries for Tandem NSK 351
 - trigger 170
 - types of 169
 - examples
 - altnqls command 222
 - altnqusr command 224
 - cleanrdf command 225
 - cnv1520 command 227
 - cnvclchl command 228
 - cnvmmsgs command 229
 - communications setup 335
 - crtrmqcvx command 232
 - crtrmqm command 237
 - dltmqm command 238
 - dspmqaout command 243
 - dspmqcsv command 244
 - dspmqls command 246
 - dspmqr command 250
 - endmqcsv command 252
 - endmqm command 254

- examples (*continued*)
 - endmqtrc command 255
 - error log 209
 - PAK installation 283
 - programming errors 198
 - runmqsc command 263
 - setmqaut command 270
 - strmqcsv command 271
 - strmqm command 272
 - strmqtrc command 274
 - upgmqm command 276
- executables, specifying location of 39
- ExpectedNumECs 62
- ExtPoolSize entry, QMINI file 67

- F**
 - feedback from MQSC commands 100
 - FEEDBACK keyword, rules table 160
 - FFST
 - examining 212
 - subvolume 70
 - files
 - authorization 153
 - configuration
 - in problem determination 210
 - MQSeries (MQSINI) 183
 - queue manager (QMINI) 185
 - forcing a channel to stop (stop immediate) 94
 - FORMAT keyword, rules table 161
 - FREEZE SERVER, PATHCOM command 48
 - FWDQ keyword, rules table 162
 - FWDQM keyword, rules table 162

- G**
 - glossary 419
 - groups, creating 20

- H**
 - hard disk requirements 20
 - hardware requirements, MQSeries for Tandem NSK 14
 - HEADER keyword, rules table 162
 - highlights of MQSeries for Tandem NSK V2.2.0.1 15
 - home volume of queue manager 62
 - HomeTerminalName parameter, crtmqm command 63
 - HTML (Hypertext Markup Language) xvii
 - Hypertext Markup Language (HTML) xvii

- I**
 - ICE communications example 342
 - idle agent processes 65
 - incorrect output 203
 - inhibit events 351
 - initiation queue
 - channel initiator, changing 49
 - defining 122
 - description 8
 - INPUTQ keyword, rules table 158
 - INPUTQM keyword, rules table 159
 - installable services
 - authorization service 358
 - name service 39, 358
 - object authority manager (OAM) 138
 - disabling 140
 - installation 13
 - directory structure 27
 - preparation 20
 - procedure 21
 - supported code sets 37
 - unsuccessful 34
 - verifying 32
 - installation subvolumes (ISVs) 19
 - ZMQSCONV 32
 - ZMQSEXE 27
 - ZMQSLIB 29
 - ZMQSSMPL 29
 - ZMQSSYS 28
 - installation volume, default 25
 - instmqm command 25, 256
 - instrumentation events
 - description 169
 - enabling 170
 - messages 171
 - purpose of 169
 - supported by MQSeries for Tandem NSK 351
 - types of 169
 - interactive MQSC
 - ending 101
 - feedback from 100
 - using 100
 - issuing MQSeries commands 99
 - ISVs (installation subvolumes) 19

- K**
 - kernel configuration 27

- L**
 - libraries, MQI 29, 318
 - license management 280
 - LIKE attribute, DEFINE QLOCAL 110
 - load balancing 63
 - local administration 99
 - local events 351
 - local queues
 - clearing 113
 - command 9

Index

- local queues (*continued*)
 - copying definitions 110
 - dead-letter 9
 - defining 108
 - deleting 114
 - description 7
 - initiation 8
 - transmission 8
 - undelivered-message 9
- log, error 207, 209
- LQMAgent stanza, QMINI file 186
- LQMAgentPriority 65
- LU 6.2 responder processes 334

- M**
- managing objects for triggering 120
- MaxActiveChannels 66
- Maxapplio, value of 341
- MaxChannels 66
- MaxIdleAgentReuse 65
- MaxIdleAgents 65
- maximum line length for MQSC commands 104
- maxrcv PARAM 342
- MCAAgentPriority 65
- MCA Caller stanza, QMINI file 186
- MCALU62Responder stanza, QMINI file 186
- MCATCPResponder stanza, QMINI file 186
- message
 - administration 384
 - containing unexpected information 205
 - description 4
 - descriptor 4
 - for instrumentation events 171
 - format 363
 - groups 364
 - information 363
 - lengths of 4, 202
 - MQSeries product 374
 - not appearing on queues 203
 - operator 208
 - performance considerations 202
 - queuing 3
 - remote 405
 - retrieval algorithms 5
 - searching for particular 202
 - structure 363
 - undelivered 210
 - variable length 203
 - variables 363
- message length, decreasing 111
- message queue interface (MQI) 3
- Message Queue Management (MQM) 78
- message queue subvolume 71
- message-driven processing 3, 14

- messages
 - common services 371
 - installable services 365
- migration from V1.5.1 13
- migration from V2.2 13
- MinIdleLQMAgents 65
- MinIdleMCA Callers 65
- MinIdleMCALU62Responders 65
- MinIdleMCATCPResponders 65
- model queues
 - defining 119
 - description 8
 - working with 119
- modifying
 - CCSID 64
 - channel 89
 - queue 86
- monitoring
 - TMF status 177
- monitoring a channel 93
- monitoring a queue 87
- monitoring queue managers 169
- MQAT_NSK, ApplType value 313
- MQCFH, PCF header 354
- MQCFIL, PCF integer list parameter 356
- MQCFIN, PCF integer 356
- MQCFSL, PCF string list 356
- MQCFST, PCF string parameter 355
- MQCHANNELEXIT() 329
- MQDATA CONVEXIT() 330
- MQDEFAULTPREFIX, environment variable 26
- MQDLH, dead-letter header 157
- MQEMSEVENTS, environment variable 26, 173
- MQGMO_BROWSE_* 316
- MQGMO_SET_SIGNAL option 302
- MQI
 - authorizations 148
 - description 3
 - libraries 29, 318
 - local administration support 99
 - queue manager calls 7
 - TAL programming language
 - MQBACK 307
 - MQCLOSE 307
 - MQCMIT 312
 - MQCONN 307
 - MQDISC 308
 - MQGET 308
 - MQINQ 311
 - MQOPEN 312
 - MQPUT 310
 - MQPUT1 310
 - MQSET 311
 - MQSYNC 312
- MQIServer stanza, QMINI file 186

- MQM
 - user group 137
 - user ID 137, 144
 - MQM (Message Queue Management) interface 78
 - MQM.MANAGER, user ID 20
 - MQMACHINIFILE, environment variable 26, 183
 - MQMC facility 78
 - MQMREFRESHINT, Pathway parameter 50
 - MQOPEN authorizations 148
 - MQPUT and MQPUT1, performance considerations 203
 - MQPUT authorizations 148
 - MQRC (reason codes) 313
 - MQRC_Q_SPACE_NOT_AVAILABLE return code 109
 - MQRDF, environment variable 26
 - MQRDFUPPROCESSNAME, environment variable 26
 - MQRDFUPPROGNAME, environment variable 26
 - MQS-CHANINIT00, TS/MP server class 48
 - MQS-CMDSERV00, TS/MP server class 48
 - MQS-EC00, TS/MP server class 49
 - MQS-STATUS00 server class 50
 - MQS-TCPLIS00, TS/MP server class 48
 - MQS-TRIGMON00, TS/MP server class 49
 - MQSC 45
 - command files
 - output reports 104
 - running 105
 - ending interactive input 101
 - how to issue commands 99
 - issuing commands interactively 100
 - issuing remotely 130
 - maximum line length 104
 - problems
 - local 106
 - remote 131
 - redirecting input and output 102
 - security requirements on channels 147
 - supported by MQSeries for Tandem NSK, V2.2.0.1 297
 - timed out command responses 130
 - using commands 102
 - verifying commands 105
 - MQSC commands
 - ALTER QLOCAL 111
 - ALTER QREMOTE 134
 - CLEAR QLOCAL 299
 - DEFINE CHANNEL 128
 - DEFINE QALIAS 118
 - DEFINE QLOCAL 110
 - DEFINE QLOCAL LIKE 110
 - DEFINE QLOCAL REPLACE 111
 - DEFINE QMODEL 119
 - DEFINE QREMOTE 132
 - DELETE QLOCAL 114
 - DISPLAY QREMOTE 134
 - MQSC commands (*continued*)
 - maximum line length 104
 - using 44
 - MQSeries
 - prerequisites 13
 - super user, MQM 137
 - MQSeries for Tandem NSK, V2.2.0.1
 - AMQxxxxx messages 363
 - building applications 315
 - client support 349
 - EMS event template used by 359
 - highlights 15
 - installation instructions 19
 - MQSC supported by 297
 - native version of 16
 - running applications 315
 - software requirements 14
 - TM/MP (TMF) support 175
 - volume structure 70
 - MQSeries publications xii
 - MQSINI, configuration file
 - editing 193
 - example 184
 - overview 183
 - path to 106
 - MQSNOAUT, environment variable 26, 140
 - MQSS Server 15
 - adding and removing nondefault 68
 - configuring 40
 - more than one per queue manager 15
 - name of default 68
 - process, recovery and restart of 180
 - processes, adding 50
 - removing 69
 - server class MQS-STATUS00 50
 - MQXCNCV, convert characters call 330
 - MQZAO constants and authority 149
 - Msgld
 - disabling message retrieval by 40
 - performance considerations when using 202
 - MSGTYPE keyword, rules table 161
 - MVS/ESA queue manager 130
- ## N
- name of default MQSS Server 68
 - name service interface 358
 - name service, configuration of 39
 - name transformation, object 72
 - names
 - objects 5
 - names, valid for objects 218
 - naming convention, MQSS Server classes 69
 - naming conventions, national language support 218
 - native version of MQSeries 16

Index

nondefault MQSS Server, adding and removing 68
nondefault TCP/IP process 67
nonpersistent messages 281
NonStop TM/MP (Transaction Manager) 175
notification of events 170
NumECs parameter, crtmqm command 62

O

OAM (object authority manager)
 disabling 140
 dspmqaut command 144
 groups 139
 how it works 139
 introduction to 12
 replacing 358
 sensitive operations 144
 setmqaut command 141, 143
object name transformation 72
objects
 access to 137
 customizing 37
 default
 attributes 109
 creating 74
 for triggering 120
 naming conventions 218
 process definition 10
 queue 7
 queue manager
 MQI calls 7
 prefixes 38
 remote administration 125
 system
 creating 74
 default 10, 293
 types of 5
operator commands, no response from 199
operator messages 208
overview of MQSeries for Tandem NSK 279

P

PAK installation 283
panels, MQM facility 78
parameters
 altnqfls command 221
 altnqusr command 223
 cleanrdf command 225
 cnv1520 command 226
 cnvclchl command 228
 cnvmsgs command 229
 crtmqcvx command 232
 crtmqm command 234
 dltnmqm command 238
 dspmqaut command 240

parameters (*continued*)
 dspmqcsv command 244
 dspmqfls command 245
 dspmqtrc command 248
 dspmqusr command 249
 endmqcsv command 251
 endmqm command 253
 endmqtrc command 255
 instmqm command 256
 runmqchi command 257
 runmqchl command 258
 runmqdlq command 259
 runmqslr command 260
 runmqsc command 262
 runmqtrm command 264
 setmqaut command 267
 strmqcsv command 271
 strmqm command 272
 strmqtrc command 273
 upgmqm command 276
PARAMs (environment variables) 26
partitioned local queue 113
partitioning macro (partit) 38
PATHCOM commands
 ADD SERVER 48
 ALTER SERVER 49
 FREEZE SERVER 48
 START SERVER 48
 STATUS SERVER 48
 STOP SERVER 48
 THAW SERVER 48
PathmonProcName 63
PATHWAY configuration, example 51
Pathway server classes, changing parameters of 57
pattern-matching keywords, rules table 160
PCF command responses 358
PCFs (programmable command formats) 354
 header (MQCFH) 354
 integer (MQCFIN) 356
 integer list parameter (MQCFIL) 356
 introduction to 44
 message descriptor 354
 MQCFH (header) 354
 MQCFIL (integer list parameter) 356
 MQCFIN (integer) 356
 MQCFSL (string list) 356
 MQCFST (string parameter) 355
 string list (MQCFSL) 356
 string parameter (MQCFST) 355
 supported by MQSeries for Tandem NSK 357
PDF (Portable Document Format) xvii
performance considerations
 of application design 202
 when using trace 210
performance events 170, 351

- permanent queues 5
 - PERSIST keyword, rules table 161
 - persistent messages 281
 - PMSEARCH 27
 - Portable Document Format (PDF) xvii
 - PostScript format xviii
 - predefined queues 5
 - preemptive queue manager shutdown 75
 - PRIDB, principals database 153
 - principals 17
 - creating 73, 142
 - database, PRIDB 153
 - naming of 139
 - purpose of 12, 36
 - priority of queue manager processes 65
 - problem determination
 - configuration files 210
 - further checks 199, 202
 - incorrect output 205
 - no response from commands 199
 - programming errors 198
 - things to check first 195
 - process definitions
 - creating 122
 - description 10
 - displaying 123
 - processing, event-driven 3
 - programmable command formats (PCFs) 44, 354
 - programming errors, examples of 198
 - programs, samples supplied 321
 - protected resources 140
 - PUTAUT keyword, rules table 162
- Q**
- QMDefaultVolume 62
 - QMINI, configuration file
 - editing 193
 - example 187
 - overview 185
 - queue depth 110
 - queue manager
 - alias, remote queue 135
 - authorizations 145
 - channel synchronization subvolume 72
 - command server 77
 - configurable properties of 62
 - configuration file
 - backing up 61
 - contents of 185
 - configuration overview 39
 - creating
 - crtmqm command 234
 - default 72
 - guidelines for 59
 - to verify installation 32
 - queue manager (*continued*)
 - data files subvolume 70
 - default 60, 235
 - accidental change 76
 - accidental deletion 235
 - changing 76
 - default volume for 39
 - deleting 76, 238
 - description 6
 - directories 145
 - endmqm command 253
 - error log subvolume 71
 - events 169
 - FFST subvolume 70
 - home volume 62
 - immediate shutdown 75
 - local administration 99
 - message queue subvolume 71
 - monitoring 169
 - numbers of 60
 - object authority manager 138
 - description 138
 - disabling 140
 - objects
 - MQI calls 7
 - prefixes 38
 - on MVS/ESA 130
 - preemptive shutdown 75
 - remote administration 125
 - removing, manually 295
 - restart 75
 - shutdown 74
 - controlled 74
 - quiesce 74
 - specifying on runmqsc 102
 - starting 74
 - stopping 74
 - manually 295
 - unique name 60
 - upgrading from V2.2 34
 - Queue Manager Menu, MQM 79
 - queued mode, of runmqsc 130
 - QueueManager stanza, MQSINI file 184
 - queues
 - alias 7
 - aliases, working with 117
 - application, defining for triggering 121
 - attributes 7
 - attributes, changing 111
 - authorizations to 145
 - browsing 114
 - command 9
 - dead-letter 9, 61
 - defining 7
 - description 4
 - distributed, incorrect output from 205

Index

- queues (*continued*)
 - dynamic 5
 - event 9, 170
 - for MQI applications 99
 - initiation
 - defining 122
 - trigger messages 8
 - local 7
 - clearing 113
 - copying 110
 - defining 108
 - deleting 114
 - model 8
 - defining 119
 - working with 119
 - objects
 - alias 7
 - local 7
 - model 8
 - remote 7
 - physical size of 109
 - predefined 5
 - remote 7
 - creating 132
 - queue manager alias 135
 - working with 135
 - reply-to 9, 135
 - size of 109
 - temporary 5
 - transmission 8
 - creating 134
 - default 61, 134
 - defining 128
 - remote administration 127
 - undelivered-message 9, 61
 - working with 108
 - Queues menu, MQM 81
 - quiesce shutdown 74
- ## R
- railroad diagrams, how to read 218
 - RDF (Remote Database Duplication Facility) 27
 - cleanrdf command 73, 225
 - compatibility with in V2.2.0.1 16
 - configuration of 41
 - disaster recovery using 181
 - MQRDF environment variable 26
 - MQRDFUPPROCESSNAME environment variable 26
 - MQRDFUPPROGNAME environment variable 26
 - reason codes 313
 - REASON keyword, rules table 161
 - redirecting input and output, on MQSC commands 102
 - related publications xix
 - remote
 - events 351
 - issuing of MQSC commands 130
 - object administration 125
 - queue definition, creating 132
 - queue object, working with 135
 - queues
 - as queue manager aliases 135
 - as reply-to queue aliases 135
 - authorizations to 145
 - description of 7
 - queuing
 - description 125
 - recommendations 131
 - security considerations 146
 - remote administration
 - command server 77
 - initial problems 131
 - Remote Database Duplication Facility (RDF) 27
 - removing MQSS Server 69
 - removing nondefault MQSS Server 68
 - removing queue manager manually 295
 - REPLACE attribute, DEFINE commands 104
 - reply-to queue 9
 - reply-to queue aliases 135
 - REPLYQ keyword, rules table 161
 - REPLYQM keyword, rules table 161
 - requirements
 - disk storage 20
 - hardware 279
 - software 279
 - resetting a message sequence number (MSN) 95
 - resolving a channel 96
 - resources, protecting 138
 - restart queue manager 75
 - RESTORE command (installation) 21
 - restrictions
 - access to MQM objects 137
 - object names 218
 - retrieval options for queues, specifying 40
 - RETRY keyword, rules table 163
 - RETRYINT keyword, rules table 159
 - return codes 196
 - altnmqfls command 221
 - altnmqsr command 223
 - cleanrdf command 225
 - crtmqcvx command 232
 - crtmqm command 236
 - dltnmqm command 238
 - dspmqaout command 242
 - dspmqcsv command 244
 - dspmqls command 246
 - dspmqsar command 249
 - endmqcsv command 251
 - endmqm command 254
 - endmqtrc command 255

- return codes (*continued*)
 - runmqchi command 257
 - runmqchl command 258
 - runmqlsr command 260
 - runmqsc command 262
 - runmqtrm command 264
 - setmqaut command 269
 - strmqcsv command 271
 - strmqm command 272
 - strmqtrc command 274
 - reusing exit programs 329
 - rollback 175
 - rules table, DLQ handler 158
 - control data entry
 - INPUTQ keyword 158
 - INPUTQM keyword 159
 - RETRYINT keyword 159
 - WAIT keyword 159
 - example 166
 - patterns and actions (rules)
 - ACTION keyword 161
 - APPLIDAT keyword 160
 - APPLNAME keyword 160
 - APPLTYPE keyword 160
 - DESTQ keyword 160
 - DESTQM keyword 160
 - FEEDBACK keyword 160
 - FORMAT keyword 161
 - FWDQ keyword 162
 - FWDQM keyword 162
 - HEADER keyword 162
 - MSGTYPE keyword 161
 - PERSIST keyword 161
 - PUTAUT keyword 162
 - REASON keyword 161
 - REPLYQ keyword 161
 - REPLYQM keyword 161
 - RETRY keyword 163
 - USERID keyword 161
 - processing of 165
 - syntax 163
 - run listener (runmqlsr command) 260
 - runmqchi command 257
 - runmqchl command 258
 - runmqdlq command 157, 259
 - runmqlsr command 260
 - runmqsc
 - command 261
 - ending 101
 - feedback 100
 - issuing MQSC commands 99
 - problems 106
 - queued mode 130
 - redirecting input and output 102
 - specifying a queue manager 102
 - using 102
 - runmqsc (*continued*)
 - using interactively 100
 - verifying 105
 - runmqtrm command 264
 - running applications 315
- ## S
- SAFEGUARD 13, 279
 - sample programs 29
 - building C versions 322
 - building COBOL versions 323
 - building TAL versions 324
 - supplied with MQSeries for Tandem NSK 321
 - sample trace data 211
 - SAVE-ENVIRONMENT ON, environment variable 26
 - SCF configuration file, example 335
 - security 137
 - enabling 140
 - OAM (object authority manager) 12
 - principals, creating 73
 - remote 146
 - using the commands 141, 144
 - server classes, user defined 57
 - Service stanza, QMINI file 186
 - ServiceComponent stanza, QMINI file 186
 - setmqaut command 265
 - installable services 143
 - related commands 270
 - using 141, 143
 - shutdown, queue manager 74
 - signal option 302
 - single-phase commit 175
 - SNA protocol 333
 - SNAX communications examples 335
 - softcopy books xvii
 - software requirements, MQSeries for Tandem NSK 14
 - specified operating environment 279
 - stanzas
 - MQSINI 184
 - QMINI 185
 - start and stop events 351
 - start MQSeries trace command 273
 - start queue manager command 272
 - START SERVER, PATHCOM command 48
 - starting
 - a queue manager 74
 - channels 94, 129
 - command server 77
 - trace 81
 - status data, maintenance of by the MQSS Server 15
 - STATUS SERVER, PATHCOM command 48
 - stdin, on runmqsc 102
 - stdout, on runmqsc 102
 - STOP SERVER, PATHCOM command 48

Index

- stopping
 - channels 94
 - command server 77
 - queue manager 253, 295
 - trace 81
 - strmqcsv command 271
 - related commands 271
 - strmqm command 272
 - related commands 272
 - strmqtrc command 273
 - related commands 275
 - structure data types, TAL 301
 - super user (MQSeries)
 - MQM 137
 - supported code sets 37
 - swap space allocation 67
 - syncpoint, performance considerations 203
 - syncpointing limits 176
 - syntax diagrams, how to read 218
 - syntax error, in MQSC commands 100
 - system default objects 10
 - system defaults 293
 - system objects, defining 37, 74
- ## T
- TACL environment variables 26
 - TAL programming language
 - elementary data types 301
 - structure data types 301
 - MQDLH, dead letter header 302
 - MQGMO, get message options 303
 - MQMD, message descriptor 304
 - MQOD, object descriptor 304
 - MQPMO, put message options 305
 - MQTM, trigger message 305
 - MQTMC2, trigger message 2 306
 - MQXQH, transmission queue header 306
 - Tandem NSK logged-in user ID 144
 - Tandem NSK, overview of 279
 - TCP/IP channels 334
 - TCP/IP communications example 345
 - TCP/IP listeners 48
 - TCP/IP process name 67
 - TCP/IP protocol 333
 - TCPConfig stanza, QMINI file 187
 - TCPLListener stanza, QMINI file 186
 - TCPLListenerPort 66
 - TCPNumListenerPorts 66
 - TCPPort 66
 - templates, EMS event 171
 - temporary queues 5
 - terminology used in this book 419
 - THAW SERVER, PATHCOM command 48
 - time-independent applications 3
 - timed out responses from MQSC commands 130
 - TM/MP (TMF) support 175, 315
 - TMF audit trail 20
 - trace
 - data sample 211
 - performance considerations 210
 - tracing MQSeries objects 80
 - Transaction Manager (NonStop TM/MP) 175
 - transactional support 175
 - transmission queue
 - creating 134
 - default 61, 134
 - defining 128
 - description 8
 - remote administration 127
 - trigger
 - events 170
 - messages on initiation queue 8
 - monitor
 - description 8
 - start command 264
 - triggered applications 316
 - passing USERDATA to 298
 - triggering
 - defining application queue for 121
 - managing objects for 120
 - troubleshooting 177
 - TS/MP administration 48
 - TS/MP server classes
 - MQS-CHANINIT00 48
 - MQS-CMDSERV00 48
 - MQS-EC00 49
 - MQS-TCPLIS00 48
 - MQS-TRIGMON00 49
 - TuningParameters stanza, QMINI file 186
 - two-phase commit 175
 - types of event 169
 - types of object 5
- ## U
- unauthorized access, protecting from 138
 - unit of work management 315
 - upgmqm command 34, 276
 - upgrading from V1.5.1 13
 - user exits 12, 329
 - user group
 - for authorization 140
 - MQM 137
 - user ID
 - authority 137
 - authorization 144
 - Tandem NSK logged-in user 144
 - user ID MQM.MANAGER 20
 - user-defined server classes 57

USERDATA, process attribute 298

USERID keyword, rules table 161

users

 belonging to more than one user group 139

 creating 20

 groups 139

V

verifying MQSC commands 105

volume structure 70

volume, changing 111

W

WAIT keyword, rules table 159

X

XA interface 316

Z

ZMQSTMPL, EMS event template file 171

Sending your comments to IBM

MQSeries® for Tandem NonStop Kernel

System Management Guide

GC33-1893-01

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

MQSeries® for Tandem NonStop Kernel

System Management Guide

GC33-1893-01

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	

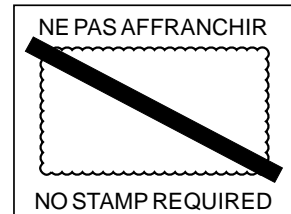
If your country is not listed here, your local IBM representative will be pleased to forward your comments to us. Or you can pay the postage and send the form direct to IBM (this includes mailing in the U.K.).

1 Cut along this line

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 _____

1 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.

GC33-1893-01

