

MQSeries® Integrator



Administration Guide

Version 2.0.2

MQSeries[®] Integrator



Administration Guide

Version 2.0.2

Note!

Before using this information and the product it supports, be sure to read the general information under “Appendix C. Notices” on page 199.

Fifth edition (April 2001)

This edition applies to IBM® MQSeries Integrator Version 2 and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 2000, 2001. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	vii
--------------------------	------------

Tables	ix
-------------------------	-----------

About this book **xi**

Who this book is for	xi
What you need to know to understand this book	xi
Terms used in this book	xi
Where to find more information	xii

Summary of changes **xiii**

Changes for this edition (from SC34-5792-03)	xiii
Changes for the third edition (from SC34-5792-02)	xiii
Changes for the second edition (from SC34-5792-01)	xiii

Part 1. Guidance **1**

Chapter 1. Administration overview **3**

Administration tasks	3
System administration overview	4
Configuring the broker domain	4
Managing the broker domain	4
Problem determination	4
Integration and migration	4
System management	5
National language support	5
Locales	6

Chapter 2. How to configure your MQSeries Integrator network. **9**

Definition and authorization tasks	9
Defining and authorizing MQSeries Integrator user IDs	9
Defining and authorizing database resources	11
Defining MQSeries Integrator components	20
Defining MQSeries resources	21
Connection tasks	21
Connecting Control Center clients to the Configuration Manager (Windows NT only)	21
Connecting two MQSeries Integrator components	22
Initialization tasks	24
Starting MQSeries queue managers as a Windows NT service	24
Starting the Configuration Manager (Windows NT only)	25
Starting a broker	26
Starting the User Name Server	26
Starting the Control Center (Windows NT only)	27
Defining and deploying the configuration in the Control Center	28
Application client and user data tasks	29
Setting up application clients	29

Configuring databases for user data accessed from message flows	30
General guidance	32

Chapter 3. How to manage your MQSeries Integrator network **35**

Managing the broker domain components	35
Managing components	35
Managing databases	36
Enhancing and updating your broker domain	37
Setting up a publish/subscribe network	37
Coordinated transactions	39
Deleting components from the broker domain	40
Importing and exporting message sets	41
Recovery and restart	42
Managing workload and performance	49
Using MQSeries trusted applications	49
Tuning message flow performance	50

Chapter 4. Setting up security. **51**

Securing MQSeries Integrator resources	51
Using Windows NT primary or trusted security domains	53
The IBMMQSI2 superuser	55
Windows NT security domain scenarios	56
Using UNIX security domains	61
Securing MQSeries resources	62
Securing database resources	63
DB2 services	64

Chapter 5. Problem determination **65**

Traces	65
Windows NT event log messages	65
UNIX syslog messages	66
Optional traces	67
Messages	74
Message flow debugger	74
MQSeries facilities	74
MQSeries logs	75
MQSeries events	75
Database logs	75
DB2 logs	75
ODBC tracing	75
Contacting your IBM Support Center	76

Part 2. Reference **79**

Chapter 6. Using MQSeries Integrator commands **81**

Rules for using MQSeries Integrator commands	81
Rules for naming resources	82
Responses to commands	83
Command syntax help	83
How to read syntax diagrams	83

Chapter 7. Using the MQSeries Integrator Command Assistant 87

Overview	87
Invocation	87
Navigation	88
Command processing	88

Chapter 8. Commands 89

mqsicchangebroker (Change broker)	90
Purpose	90
Syntax	90
Required parameters	90
Optional parameters	90
Authorization	91
Responses	91
Examples	92
Related commands	92
mqsicchangeconfigmgr (Change Configuration Manager)	93
Purpose	93
Syntax	93
Optional parameters	93
Authorization	94
Responses	94
Examples	94
Related commands	94
mqsicchangetrace (Change trace settings)	95
Purpose	95
Syntax	96
Required parameters	96
Optional parameters	96
Authorization	98
Responses	98
Examples	98
Related commands	98
mqsicchangeusernameserver (Change User Name Server)	99
Purpose	99
Syntax	99
Optional parameters	99
Authorization	100
Responses	100
Examples	100
Related commands	100
mqsiclearmqpubsub (Remove MQSeries Publish/Subscribe broker as a neighbor)	101
Purpose	101
Syntax	101
Required parameters	101
Authorization	101
Responses	101
Examples	102
Related commands	102
mqsicopymsgset (Copy message set)	103
Purpose	103
Syntax	103
Required parameters	103
Authorization	104
Examples	104
Related commands	104

mqsicreatebroker (Create broker)	105
Purpose	105
Syntax	106
Required parameters	106
Optional parameters	107
Authorization	108
MQSeries queues created	108
Database tables created	108
Responses	109
Examples	110
Related commands	110
mqsicreateconfigmgr (Create Configuration Manager)	111
Purpose	111
Syntax	112
Required parameters	112
Optional parameters	113
Authorization	114
MQSeries queues created	114
MQSeries channels created	114
Database tables created	114
Responses	115
Examples	116
Related commands	116
mqsicreateusernameserver (Create User Name Server)	117
Purpose	117
Syntax	117
Required parameters	117
Optional parameters	118
Authorization	118
MQSeries queues created	118
Responses	119
Examples	119
Related commands	119
mqsideletebroker (Delete broker)	120
Purpose	120
Syntax	121
Required parameters	121
Optional parameters	121
Authorization	121
Responses	121
Examples	121
Related commands	121
mqsideleteconfigmgr (Delete Configuration Manager)	122
Purpose	122
Syntax	122
Optional parameters	122
Authorization	123
Responses	123
Examples	124
Related commands	124
mqsideleteusernameserver (Delete User Name Server)	125
Purpose	125
Syntax	125
Optional parameters	125
Authorization	125
Responses	125
Examples	126

Related commands	126
mqsiformatlog (Format log)	127
Purpose	127
Syntax	127
Required parameters	127
Optional parameters	127
Authorization	127
Responses	127
Examples	127
Related commands	128
mqsiimpexpmsgset (Import/Export message set)	129
Purpose	129
Syntax	129
Required parameters	129
Authorization	130
Examples	130
Related commands	130
mqsijoinmqpubsub (Join broker to MQSeries Publish/Subscribe parent broker)	131
Purpose	131
Syntax	131
Required parameters	131
Authorization	131
Responses	131
Examples	132
Related commands	132
mqsilcc (Start Control Center trace)	133
Purpose	133
Syntax	133
Optional parameters	133
Authorization	134
Responses	134
Examples	134
mqsilist (List resources)	135
Purpose	135
Syntax	135
Optional parameters	135
Authorization	135
Responses	136
Examples	136
mqsilistmqpubsub (List MQSeries Publish/Subscribe neighbor broker status)	137
Purpose	137
Syntax	137
Required parameters	137
Authorization	137
Responses	137
Examples	138
Related commands	139
mqsinrfreload (Reload NEON messages)	140
Purpose	140
Syntax	140
Required parameters	140
Authorization	140
Responses	140
Examples	141
Related commands	141
mqsireadlog (Read log)	142
Purpose	142
Syntax - user trace	142
Syntax - service trace	142

Required parameters	142
Optional parameters	143
Authorization	144
Responses	144
Examples	145
Related commands	145
mqsireporttrace (Report trace settings)	146
Purpose	146
Syntax	146
Required parameters	146
Optional parameters	146
Authorization	147
Responses	147
Examples	147
Related commands	147
mqsisstart (Start component)	148
Purpose	148
Syntax	148
Required parameters	148
Authorization	148
Responses	149
Examples	149
Related commands	149
mqsisstop (Stop component)	150
Purpose	150
Syntax	150
Required parameters	150
Optional parameters	150
Authorization	150
Responses	151
Examples	151
Related commands	151

Part 3. Migration and integration 153

Chapter 9. Previous versions of MQSeries Integrator 155

Migrating from MQSeries Integrator Version 2.0.1	155
Upgrading your Version 2.0.1 Control Center	155
Upgrading your Version 2.0.1 brokers	156
Upgrading your Version 2.0.1 Configuration Manager	156
Migrating from MQSeries Integrator Version 2.0	156
Upgrading your Version 2.0 Control Center	156
Upgrading your Version 2.0 brokers	156
Upgrading your Version 2.0 Configuration Manager	157
Migrating NEON Rules and Formats from previous versions of MQSeries Integrator	158
Using NNfie and NNRie for migration	158
Adding new Rules and Formats	158
Reloading Rules and Formats	159
User exits	159
Logs and log records	159
The MQSeries Integrator Version 1 Rules Daemon	159

Chapter 10. MQSeries Publish/Subscribe 161

Before you start	161
------------------	-----

Figures used in this chapter	161
Commands and options	161
Stream queues	163
Running two independent broker networks	163
Creating and operating a heterogeneous network	163
Adding an MQSeries Integrator broker as a leaf node	165
Adding an MQSeries Integrator broker as a parent node	167
Deleting brokers in a heterogeneous network	169
Migrating MQSeries Publish/Subscribe brokers	170
The Control Center and migration	171
Migrating a single broker	171
Migrating a broker network	173
A network of migrated brokers	178

Part 4. Appendixes 181

Appendix A. Event reporting 183

General architecture	184
Configuration changes	184
Changes to the local configuration of the broker	184
Neighbor changes	185
Examples	185
ACL updates	186
Examples	187
Operational information	188
Subscriptions and topics	189
Subscription registration and deregistration	189
Examples	189
Operational warnings	191
Expired publications and subscriptions	191
Examples	191
Notification message schema	192

Appendix B. Using NEON	195
Installing database schema	195
Setting the NEON_ROOT system environment variable	197
Access to Rules and Formats	197
Editing the database configuration file (neonreg.dat)	197
Importing rules and formats	198

Appendix C. Notices 199

Trademarks	201
----------------------	-----

Glossary of terms and abbreviations 203

Bibliography 209

MQSeries Integrator Version 2.0.2 cross-platform publications	209
MQSeries Integrator Version 2.0.2 platform-specific publications	209
MQSeries Everyplace publications	209
NEONRules and NEONFormatter Support for MQSeries Integrator publications	209
Softcopy books	210
Portable Document Format (PDF)	210
MQSeries publications	211
MQSeries Publish/Subscribe publications	211
MQSeries Workflow publications	211
DB2 publications	212
MQSeries information available on the Internet	212

Index 213

Sending your comments to IBM . . . 219

Figures

1.	Connecting two MQSeries Integrator components	23	8.	Migrating an MQSeries Publish/Subscribe broker network: root node migration.	175
2.	A message flow to reproduce the function of the NEON Rules Daemon	160	9.	Migrating an MQSeries Publish/Subscribe broker network: breaking the connection	176
3.	Key to the integration and migration figures	161	10.	Migrating an MQSeries Publish/Subscribe broker network: rejoining	177
4.	Adding an MQSeries Integrator broker as a leaf node	165	11.	Migrating an MQSeries Publish/Subscribe broker network: second leaf node migration	177
5.	Adding an MQSeries Integrator broker as a parent node	167	12.	Migrating an MQSeries Publish/Subscribe broker network: final state	178
6.	Migrating an MQSeries Publish/Subscribe broker network: initial state.	174	13.	An integrated broker network	179
7.	Migrating an MQSeries Publish/Subscribe broker network: first leaf node migration	175			

Tables

1.	Supported databases for brokers and user data	12	4.	How to read syntax diagrams	84
2.	Summary of administrative task authorization on Windows NT platforms	54	5.	Service trace: qualifiers valid with components	144
3.	Summary of administrative task authorization on UNIX platforms	62	6.	Where to find command information	162
			7.	File names of MQSeries Integrator book PDFs	210

About this book

This book provides guidance and reference information that will help you administer and manage your MQSeries Integrator broker domain.

It provides detailed information on system administration: it references but does not describe in detail the related business administration tasks (for example, defining message flows) that are supported by the Control Center. For information about these tasks, you must refer to *MQSeries Integrator Version 2.0.2 Using the Control Center*.

“Part 1. Guidance” on page 1 gives a brief overview of MQSeries Integrator Version 2.0.2 administration facilities, and provides detailed guidance information on the tasks covered in this book.

“Part 2. Reference” on page 79 provides reference material for the set of commands provided by MQSeries Integrator.

“Part 3. Migration and integration” on page 153 gives migration implementation information for existing users of MQSeries Publish/Subscribe and of previous versions of MQSeries Integrator.

A glossary and bibliography is also provided.

For further information about the product, and planning for its use, refer to the *MQSeries Integrator Version 2.0.2 Introduction and Planning* book.

Who this book is for

This book is for system administrators of systems on which MQSeries Integrator Version 2 components are installed and tested.

What you need to know to understand this book

To understand this book, you need to be familiar with the system facilities of your operating system. You also need to be familiar with the administration facilities of the MQSeries Messaging product and those of any database product you wish to use in association with MQSeries Integrator Version 2.

Terms used in this book

All references in this book to MQSeries Integrator are to MQSeries Integrator Version 2 unless otherwise stated.

All references in this book to Windows NT[®] are also applicable to Windows[®] 2000 unless otherwise stated. MQSeries Integrator components that are installed and operated on Windows NT can also be installed and operated on Windows 2000.

All new terms introduced in this book are defined in “Glossary of terms and abbreviations” on page 203.

The book uses the following shortened names:

About this book

- MQSeries: a general term for IBM MQSeries Messaging products.
- MQSeries Publish/Subscribe: the MQSeries Publish/Subscribe SupportPac™ available on the Internet for several MQSeries server operating systems (the Internet URL is given in “Where to find more information”).
- DB2®: a general term to encompass IBM DB2 Universal Database® Enterprise Edition, Connect Enterprise Edition, and Extended Enterprise Edition.

Where to find more information

You should refer to “Bibliography” on page 209 for sources of further information about MQSeries Integrator and for other MQSeries family publications.

Summary of changes

This section describes changes in this edition of *MQSeries Integrator Administration Guide*. Changes since the previous edition of the book are marked by vertical lines to the left of the changes.

Changes for this edition (from SC34-5792-03)

- A section dealing with national language support has been included in this edition. (See “National language support” on page 5 for details.)
- Two commands for manipulating message sets, **mqsimrmcopymsgset** and **mqsimrmimpexp**, have been deprecated in favor of the new commands **mqsicopymsgset** and **mqsimpexpmsgset**. (See “mqsicopymsgset (Copy message set)” on page 103 and “mqsimpexpmsgset (Import/Export message set)” on page 129 respectively.)
- The **mqsinrfreload** command is deprecated in this version of MQSeries Integrator. (See “mqsinrfreload (Reload NEON messages)” on page 140.)
- There is a new tool to help with problem determination: a message flow debugger. (See “Message flow debugger” on page 74 for details.)
- The discussion of issues relating to migration from earlier versions of MQSeries Integrator has been reinstated to this book as they mostly relate to post-installation activities. Instructions for migrating from MQSeries Integrator Version 1 have been modified to account for the new functionality in the NEONRules and NEONFormatter Support for MQSeries Integrator. (See “Chapter 9. Previous versions of MQSeries Integrator” on page 155 for details.)
- Information about NEONFormatter and NEONRules nodes has been moved to this book from the *MQSeries Integrator Installation Guide* for each supported platform and modified to account for the new functionality in the NEONRules and NEONFormatter Support for MQSeries Integrator. (See “Appendix B. Using NEON” on page 195).
- This edition covers Windows 2000 as well as Windows NT, although the latter name is used throughout as a general term to cover both platforms.

Changes for the third edition (from SC34-5792-02)

- The discussion of issues relating to migration from earlier versions of MQSeries Integrator have been removed from this book and are now dealt with in the *MQSeries Integrator Installation Guide* for your computer platform
- When using the **mqsichangetrace** command, be aware of the size limitations for log files, depending on how you subsequently intend to read the log using the **mqsireadlog** command. (See “mqsichangetrace (Change trace settings)” on page 95.)

Changes for the second edition (from SC34-5792-01)

Most changes relate to operating components of the system on the UNIX[®] platforms which are now supported. However, you should note that, in this release, the Control Center and the Configuration Manager can only be used on Windows NT. Specific points you should note:

- You can access the MQSeries Integrator library, provided in softcopy, on all supported platforms. (See “Bibliography” on page 209.)

Changes

- Major activities within the system are recorded in the local error log. On Windows NT systems, this is the system event log. You will find the equivalent information in the syslog on UNIX platforms. (See “UNIX syslog messages” on page 66.)
- Whereas keywords and broker names are not case sensitive on Windows NT, you should be aware that they are on UNIX platforms. (See “Rules for using MQSeries Integrator commands” on page 81.)
- The implementation of the MQSeries Integrator security architecture is different on UNIX platforms. You should also be aware that MQSeries Integrator task authorizations are limited to eight bytes or less in an environment that includes clients on heterogeneous platforms. (See “Using UNIX security domains” on page 61.)
- Both commands **mqsicreatebroker** and **mqsicreateconfigmgr** will now enable a default dead-letter queue (DLQ). See “mqsicreatebroker (Create broker)” on page 105 and “mqsicreateconfigmgr (Create Configuration Manager)” on page 111 for details of the implications and changes in behavior that a DLQ causes.)
- The **mqsistop** command now has an extra optional parameter to force a broker to stop immediately. (See “mqsistop (Stop component)” on page 150.)
- There is a new command, **mqsicopymsgset**, which you can use to create a copy of a complete message within a message repository. (See “mqsicopymsgset (Copy message set)” on page 103 for full details.)
- It is now possible to set the maximum Java™ virtual machine (JVM) heap size when using the **mqsichangeconfigmgr** command. (See “mqsichangeconfigmgr (Change Configuration Manager)” on page 93.)
- As well as creating broker tables in DB2 and SQL Server databases, Oracle and Sybase databases can also be used. (See “Defining and authorizing database resources” on page 11.)
- You should be aware that there are issues relating to migration from MQSeries Integrator Version 2.0 to Version 2.0.1 which should be addressed.

Part 1. Guidance

Chapter 1. Administration overview	3
Administration tasks	3
System administration overview	4
Configuring the broker domain	4
Managing the broker domain	4
Problem determination	4
Integration and migration	4
System management	5
National language support	5
Locales	6
Changing your locale on Windows NT	6
Changing your locale on UNIX platforms	6
Chapter 2. How to configure your MQSeries Integrator network	9
Definition and authorization tasks	9
Defining and authorizing MQSeries Integrator user IDs	9
Prerequisite tasks	10
Completing the task for service user IDs on Windows NT	10
Completing the task for service user IDs on UNIX platforms	11
Completing the task for Control Center users (Windows NT only)	11
Subsequent tasks	11
Defining and authorizing database resources	11
Database setup and configuration	12
Configuring databases for internal data	15
Defining internal MQSeries Integrator database connections (Windows NT only)	18
Authorizing internal database access	19
Configuring databases for NEON message formats	20
Defining MQSeries Integrator components	20
Prerequisite tasks	20
Completing the task	20
Subsequent tasks	21
Defining MQSeries resources	21
Connection tasks	21
Connecting Control Center clients to the Configuration Manager (Windows NT only)	21
Prerequisite tasks	22
Completing the task	22
Subsequent tasks	22
Connecting two MQSeries Integrator components	22
Prerequisite tasks	23
Completing the task	23
Subsequent tasks	24
Initialization tasks	24
Starting MQSeries queue managers as a Windows NT service	24
Prerequisite tasks	25
Completing the task	25
Subsequent tasks	25

Starting the Configuration Manager (Windows NT only)	25
Prerequisite tasks	25
Completing the task	25
Subsequent tasks	26
Starting a broker	26
Prerequisite tasks	26
Completing the task	26
Subsequent tasks	26
Starting the User Name Server	26
Prerequisite tasks	26
Completing the task	27
Subsequent tasks	27
Starting the Control Center (Windows NT only)	27
Defining and deploying the configuration in the Control Center	28
Prerequisite tasks	28
Completing the task	28
Subsequent tasks	29
Application client and user data tasks	29
Setting up application clients	29
MQSeries resources for client applications	30
Configuring databases for user data accessed from message flows	30
Message flow transactionality and database design	31
General guidance	32
Chapter 3. How to manage your MQSeries Integrator network	35
Managing the broker domain components	35
Managing components	35
Starting and stopping components	35
Viewing and modifying components	35
Managing databases	36
Databases and code pages	36
Managing database access	36
Enhancing and updating your broker domain	37
Setting up a publish/subscribe network	37
Adding a User Name Server to your broker domain	38
Setting up collectives	38
Defining ACLs	39
Coordinated transactions	39
Using DB2 in coordinated transactions	39
Using Oracle in coordinated transactions (Sun Solaris only)	39
Deleting components from the broker domain	40
Deleting a broker	40
Removing topic-based security	41
Importing and exporting message sets	41
Recovery and restart	42
Making sure that messages aren't lost	42
Making sure that subscriptions aren't lost	45
Restart scenarios	45
Backup and recovery	47

Recovery scenarios	48
Managing workload and performance	49
Using MQSeries trusted applications	49
Tuning message flow performance.	50
Chapter 4. Setting up security	51
Securing MQSeries Integrator resources	51
Using Windows NT primary or trusted security domains	53
The IBMMQSI2 superuser	55
Windows NT security domain scenarios	56
Scenario 1: operation in a Windows NT primary domain	56
Scenario 2: operation in a Windows NT trusted domain	58
Scenario 3: operation on a stand-alone machine	60
Using UNIX security domains	61
Securing MQSeries resources	62
Securing database resources	63
DB2 services	64
On Windows NT	64
On UNIX platforms	64
Chapter 5. Problem determination	65
Traces	65
Windows NT event log messages	65
UNIX syslog messages.	66
Optional traces	67
Starting user trace	68
Checking user trace options	69
Changing user trace options.	69
Retrieving user trace information	69
Formatting user trace information	70
Viewing and interpreting user trace information	73
Stopping user trace.	73
Controlling Service traces.	73
Messages	74
Message flow debugger	74
MQSeries facilities	74
MQSeries logs	75
MQSeries events.	75
Database logs.	75
DB2 logs	75
ODBC tracing	75
Contacting your IBM Support Center.	76

Chapter 1. Administration overview

This chapter provides a summary of the administration tasks of MQSeries Integrator. It assumes you are already familiar with the concepts and components of MQSeries Integrator: if this is not the case you are recommended to read the *MQSeries Integrator Introduction and Planning* book for this information.

This chapter provides:

- “Administration tasks”.
- “System administration overview” on page 4.
- “National language support” on page 5.

Administration tasks

Administration for MQSeries Integrator can be considered in two main categories:

- Business administration. Implementation of the tasks introduced in *MQSeries Integrator Introduction and Planning*, Part 2 “Business Planning”. These tasks include the definition and management of message sets and message flows, and the management of topics and subscriptions. These tasks are primarily implemented by the Control Center, which is fully described in the *MQSeries Integrator Using the Control Center* book and in the online help.
- System administration. Implementation of the tasks discussed in *MQSeries Integrator Introduction and Planning*, Part 4 “Systems Planning”. These tasks include the administration of MQSeries Integrator components, the security options that can be put in place, and the MQSeries infrastructure on which MQSeries Integrator depends. These tasks are primarily implemented by the command interface described in “Part 2. Reference” on page 79.

You will find that this book and *MQSeries Integrator Using the Control Center* are complementary. Some significant tasks are unique to one book: for example, you create the Configuration Manager using the commands in this book. Other tasks cover subtasks in both books: for example, to create and use a broker, you must create the physical broker using the commands in this book, but you cannot use that broker until you define it to the configuration repository, and assign resources to it, both of which you must do using the Control Center.

You will also find that there are a very small number of tasks which can be achieved through more than one interface. For example, you can set trace on for a particular message flow using either the command (described in this book) or the options in the Control Center.

Every task that requires action both through commands, and through the Control Center, or that provides a choice of action, contains cross reference information so that you can complete the task in the manner you choose.

You should note that, in this release, the Configuration Manager and the Control Center can only be used on Windows NT platforms.

System administration overview

When you have installed MQSeries Integrator, you can use the facilities provided to help you configure and administer your broker domain and its components.

The tasks and facilities for component administration described in this book are:

- “Configuring the broker domain”.
- “Managing the broker domain”.
- “Problem determination”.
- “System management” on page 5.

Configuring the broker domain

The configuration tasks are:

- Create, modify, and delete the Configuration Manager.
- Create, modify, and delete brokers.
- Create, modify, and delete the User Name Server.

These tasks are discussed in “Chapter 2. How to configure your MQSeries Integrator network” on page 9. They are implemented by a set of commands described in detail in “Chapter 8. Commands” on page 89. The configuration tasks are also supported by a graphical user interface, the MQSeries Integrator Command Assistant, described in detail in “Chapter 7. Using the MQSeries Integrator Command Assistant” on page 87.

Managing the broker domain

The tasks for managing the broker domain are:

- Start and stop brokers, the Configuration Manager, and the User Name Server.
- List components, and some resources, on the local system.

These tasks are discussed in “Chapter 3. How to manage your MQSeries Integrator network” on page 35. They are implemented by a set of commands described in detail in “Chapter 8. Commands” on page 89.

Problem determination

MQSeries Integrator provides facilities that help you to understand what is going on in your broker domain, and to track activity and make changes. These facilities support the following tasks:

- Start and stop tracing for components and subcomponents.
- Retrieve and format log records.

“Chapter 5. Problem determination” on page 65 describes these tasks in more detail. The commands that implement these tasks are described in “Chapter 8. Commands” on page 89

- Use the debugger to debug message flows.

See “Message flow debugger” on page 74 and the *MQSeries Integrator Using the Control Center* book for details of its use.

Integration and migration

If you already have an MQSeries Integrator Version 1 network of brokers, or are using MQSeries Publish/Subscribe, the following tasks are of interest:

- Migration of rules and formats from MQSeries Integrator Version 1. See the *MQSeries Integrator Installation Guide* for your computer platform for details.

- Integration and migration of MQSeries Publish/Subscribe brokers. These tasks are discussed in “Chapter 10. MQSeries Publish/Subscribe” on page 161.

System management

MQSeries Integrator provides facilities that assist in centralized system management. These facilities support the following tasks:

- Monitoring of the status and activity of MQSeries Integrator system components (brokers, the Configuration Manager and the User Name Server). For example, reports are generated whenever a broker starts or stops.
- Monitoring of the status and activity of execution groups.
- Monitoring of the status and activity of message flows.

MQSeries Integrator generates reports (similar in function to MQSeries events) to provide information about the operation and status of the broker domain. The nature and format of these report messages, in XML, is described in “Appendix A. Event reporting” on page 183.

These report messages are published with specific associated topics, and external organizations can therefore subscribe to the topics to support MQSeries Integrator broker domains from a center of competence anywhere within the MQSeries network.

National language support

MQSeries Integrator Version 2.0.2 is enabled for national language support. The user interface and message catalogs are provided in the following languages:

- Brazilian Portuguese
- French
- German
- Italian
- Japanese
- Korean
- Simplified Chinese
- Spanish
- Traditional Chinese
- US English

MQSeries Integrator Version 2.0.2 can process and construct messages in any code page for which MQSeries supports conversion to and from Unicode, on all operating systems. Supported code pages are listed in the *MQSeries Application Programming Reference*.

Note: The NEONRules and NEONFormatter Support nodes (NEONMap, NEONRulesEvaluation, and NEONTransform, and the superseded NEONFormatter and NEONRules) and graphical user interfaces are supplied in US English only

MQSeries Integrator interacts with MQSeries installed in any supported language. All languages for the MQSeries messaging products are included on the MQSeries server CD supplied with MQSeries Integrator.

National language support

All messages generated for internal intercomponent message exchange are generated in code page 1208.

DB2 Version 7.1 is fully NLS-enabled and is provided in all supported languages.

Locales

MQSeries Integrator Version 2.0.2 supports messages for at least the following locales:

Windows NT	AIX	Sun Solaris	HP-UX
English (United States)	en_US	en_US	en_US.iso88591, en_US.roman8
German (Standard)	de_DE, De_DE	de	de_DE.ISO88591, de_DE.roman8
Spanish (Modern Sort)	es_ES, Es_ES	es	es_ES.ISO88591, es_ES.roman8
French (Standard)	fr_FR, Fr_FR	fr	fr_FR.ISO88591, fr_FR.roman8
Italian (Standard)	it_IT, It_IT	it	it_IT.ISO88591, it_IT.roman8
Portugese (Brazilian)	pt_BR, Pt_BR	pt_BR	pt_BR.ISO88591, pt_BR.roman8
Japanese	Ja_JP, ja_JP	ja_JP.PCK, ja	ja_JP.SJIS, ja_JP.eucJP
Simplified Chinese (China)	Zh_CN, zh_CN	zh, zh.GBK	zh_CN.hp15CN
Traditional Chinese (Taiwan)	Zh_TW, zh_TW	zh_TW, zh_TW.BIG5	zh_TW.big5, zh_TW.eucTW
Korean	ko_KR	ko	ko_KR.eucKR

Changing your locale on Windows NT

On Windows NT you can either:

- Install a locale-specific operating system, or
- Alter the platform's locale by using the 'Regional Settings' GUI from the Control Panel to view and alter your user locale or system locale.

The product components are started as services on Windows NT, so are influenced by the system locale, whereas the command line functions are influenced by the users locale. The Windows NT platform has all its locale information installed by default.

Changing your locale on UNIX platforms

On UNIX platforms there is a supplied profile in the directory `sample/profiles` that shows how to set the various locale environment variables; `LANG`, `LC_ALL` and `NLSPATH`. If you use CDE, then you should generally use this to set the locale, rather than setting `LANG` and `LC_ALL` directly. The `NLSPATH` variable respects either method

You can use the executable `locale` to show your current locale. The command `locale -a` displays all the locales currently installed on the machine. You should refer to the UNIX platform documentation for information about adding new locales. If you add a new locale after you have installed this product, you will need to install that locale's message catalogs from the original install media.

National language support

When you start a broker component, the locale of that component is inherited from the shell that it is started from. The broker component uses the LC_MESSAGES environment variable as the search path in the NLSPATH environment variable (LC_MESSAGES is set when variable LC_ALL is exported).

Messages that are sent to the syslog are sent in the codepage set by this locale, if you have multiple brokers writing to this syslog then their messages will be in the codepage of the locale in which they were started, for example:

locale	syslog codepage	ccsid
pt_BR	iso8859-1	819
Pt_BR	ibm-850	850
PT_BR	utf-8	1208

You should set the locale of the userid that runs the syslog daemon to one that is compatible with the locale that the broker runs in. You can do this by setting the default locale. On Sun Solaris and HP-UX, set the LANG and LC_ALL variables in /etc/default/init. On AIX, these variables are in /etc/environment.

National language support

Chapter 2. How to configure your MQSeries Integrator network

This chapter describes the tasks that are needed to configure and activate an MQSeries Integrator network. It builds on the planning information provided in *MQSeries Integrator Introduction and Planning*, and on the configuration of a simple broker domain described in the *MQSeries Integrator Installation Guide* for your computer platform.

The tasks are presented in related groups, in the order in which you are recommended to complete them after installation. However, you can implement many of the tasks in isolation, at a later time, if appropriate. For example, if you want to add a new broker into an existing domain, you can follow the steps to create a broker described in “Defining MQSeries Integrator components” on page 20.

Each task identifies its prerequisite tasks, and the tasks that you are required or recommended to complete following that task. This helps you to ensure you do everything you need to do to complete each specific task.

All the tasks illustrated use sample names. You can use the same names, or you can choose your own names. You must remember to substitute your names for the sample names wherever they are used in these examples, if appropriate.

The tasks are grouped as follows:

- “Definition and authorization tasks”
- “Connection tasks” on page 21
- “Initialization tasks” on page 24
- “Application client and user data tasks” on page 29

Following the task descriptions, some general guidance and recommendations are provided (“General guidance” on page 32).

“Chapter 3. How to manage your MQSeries Integrator network” on page 35 describes additional tasks that change the configuration you have created by completing the tasks in this chapter.

Definition and authorization tasks

When you configure a broker domain, you must define the resources you need, and grant users the authority to use them:

- “Defining and authorizing MQSeries Integrator user IDs”
- “Defining and authorizing database resources” on page 11
- “Defining MQSeries Integrator components” on page 20
- “Defining MQSeries resources” on page 21

Defining and authorizing MQSeries Integrator user IDs

MQSeries Integrator components (brokers, the Configuration Manager, and the User Name Server) run under nominated user IDs known as Service User IDs. You must define these user IDs to your security domain, and add them to the

Defining user IDs

groups necessary to authorize their operation. You must also define one or more user IDs for Control Center use. Control Center users must belong to specific groups to achieve specific tasks.

For a detailed discussion of users, groups, and security, see “Chapter 4. Setting up security” on page 51.

Prerequisite tasks

You must complete the following task before you start this task:

- Installation of MQSeries Integrator. See the *MQSeries Integrator Installation Guide* for details.

Completing the task for service user IDs on Windows NT

You must first decide what user IDs you will nominate as service user IDs for the Configuration Manager, the User Name Server (if you have one), and for each broker. If you decide to create new user IDs, follow all the steps below. If you are using existing user IDs, start with Step 3. You can check the requirements for service user IDs in Table 2 on page 54.

You must take the following steps:

- Step 1.** Invoke the Windows NT User Manager on the system that is your security domain controller. You can access this program from the Windows NT Start menu (the default is *Start* → *Programs* → *Administrative Tools* → *User Manager*).
- Step 2.** Create a new user ID.
 - a. Select the *User* menu and select *New user...*
 - b. Fill in the fields on the dialog presented to create the user ID `mqsuid`, with password `mqsipw`.
Repeat Step 2 if you want to create additional user IDs.
- Step 3.** Add the user ID to the MQSeries Integrator group **mqbrkrs** on the local system, or to **Domain mqbrkrs** on the domain controller if you are using a Windows NT security domain:
 - a. Click the **Groups** button on the *New User* or the *User properties* dialog. This presents another dialog, *Group Memberships*.
 - b. Add the user to **mqbrkrs** or **Domain mqbrkrs**.
- Step 4.** If this user ID is the Configuration Manager service user ID, you must also add this id to the MQSeries group **mqm** on the local system. (See Table 2 on page 54 for full options for this step.)
- Step 5.** If this user ID is the broker service user ID, and you plan to run this broker as a trusted MQSeries application, you must also add this id to the local MQSeries group **mqm**. (See Table 2 on page 54 for full options for this step.) (For details of running brokers as trusted applications, see “Using MQSeries trusted applications” on page 49.)
- Step 6.** Click **OK**. The User Manager returns you to the *New User* or *User properties* dialog.
- Step 7.** Click **OK**.

You can change the service user ID for a component by invoking the appropriate **mqsichangexxxx** command or the Command Assistant. However, if the service user ID is also used for database access, you must follow the instructions given in “Managing databases” on page 36.

Completing the task for service user IDs on UNIX platforms

To set up a new user ID on UNIX platforms, type the following command on Sun Solaris or HP-UX:

```
useradd -g mqm -G mqbrkrs <mqsiuid>
```

or the following on AIX®:

```
mkuser pgrp=mqm groups=mqbrkrs <mqsiuid>
```

Where <mqsiuid> is the new user ID.

Completing the task for Control Center users (Windows NT only)

You must add all user IDs that will use the Control Center to the MQSeries Integrator group or groups that will authorize their use of the Control Center. For details of the roles the Control Center users can assume, see *MQSeries Integrator Version 2.0.2 Using the Control Center*.

You can do this on the domain controller of the security domain specified when you created the Configuration Manager, or you can do this in the local domain of the system on which the user will run the Control Center.

- Step 1. If you want to create new user IDs for the Control Center users, follow the instructions for creating new users in “Completing the task for service user IDs on Windows NT” on page 10 to define the user IDs, either to your security domain or to the local system on which they will run the Control Center.
- Step 2. For each user ID you want to authorize, double click the user ID to bring up the *User properties* dialog.
- Step 3. Click the **Groups** button on this dialog. This presents another dialog, *Group Memberships*.
- Step 4. If you are on the security domain controller, add the user ID to:
 - **Domain mqbrasgn**, if this user is a “Message flow and Message Set Assigner”.
 - **Domain mqbrdevt**, if this user is a “Message flow and Message Set Developer”.
 - **Domain mqbrops**, if this user is an “Operational domain controller”.
 - **Domain mqbrtpic**, if this user is a “Topic security administrator”.
- Step 5. If you are on the local system, add the user ID to:
 - **mqbrasgn**, if this user is a “Message flow and Message Set Assigner”.
 - **mqbrdevt**, if this user is a “Message flow and Message Set Developer”.
 - **mqbrops**, if this user is an “Operational domain controller”.
 - **mqbrtpic**, if this user is a “Topic security administrator”.
- Step 6. Click **OK**. The User Manager returns you to the *User Properties* dialog.
- Step 7. Click **OK**.

Subsequent tasks

When you have completed this task, continue with:

- “Defining and authorizing database resources”

Defining and authorizing database resources

MQSeries Integrator depends on databases for its own configuration and control purposes. These databases must be defined before you create the MQSeries Integrator components. You must also authorize specific users to access these databases.

Defining database resources

Table 1 gives information about the supported databases on different operating system platforms.

This task contains the following subtasks:

- “Database setup and configuration”
- “Configuring databases for internal data” on page 15
- “Defining internal MQSeries Integrator database connections (Windows NT only)” on page 18
- “Authorizing internal database access” on page 19
- “Configuring databases for NEON message formats” on page 20

Table 1. Supported databases for brokers and user data

Database	AIX	HP-UX	Sun Solaris	Windows NT	Windows 2000
DB2 ^{1 2}	6.1 ³ 7.1 ³	7.1 ³	6.1 ³ 7.1 ³	6.1 ³ 7.1 ³	6.1 ³ 7.1 ³
Microsoft SQL Server	not applicable	not applicable	not applicable	6.5 plus SP5a 7.0 plus SP1	7.0 plus SP1 2000
Oracle ¹	8.1.6 8.1.7	8.1.6 8.1.7	8.1.6 8.1.7	8.1.6 8.1.7	8.1.6 8.1.7
Sybase	12	not supported	12	12	12

Notes:

1. DB2 6.1 and DB2 7.1 on all supported operating systems, and Oracle 8.1.6 and Oracle 8.1.7 **on Sun Solaris only**, are the only DBMS supported by MQSeries Integrator that permit a database to participate as a Resource Manager in a distributed XA transaction, and coordinated by MQSeries as the XA Transaction Manager. In MQSeries Integrator, this is referred to as supporting a globally coordinated message flow.
2. You must use DB2 for the configuration and message repository databases maintained by the Configuration Manager. No other database is supported for this purpose.
3. Please check the Readme.txt file for your product to check if a Fixpack is required.

Database setup and configuration

On both Windows NT and UNIX platforms, you need to configure your database as an ODBC data source:

Windows NT: On Windows NT, an ODBC data source is configured by using the ODBC Data Source Administrator. (Select *Start* → *Settings* → *Control Panel* and then select the ODBC icon.) When defining a new data source, select the appropriate driver for your database and complete the dialog which is displayed. Refer to your relevant database product documentation for more information. Setup considerations specific to MQSeries Integrator are described below for the supported databases:

- **DB2 UDB**

When defining a data source for DB2 UDB you must choose the driver named “IBM DB2 ODBC DRIVER”.

- **Microsoft® SQLServer**

When defining a data source for Microsoft SQLServer you must choose the driver named “SQL Server”.

- **Sybase Adaptive Server Enterprise**

When defining a data source for Sybase Adaptive Server Enterprise you must choose the driver named “MQSeries MERANT 3.7 32-BIT Sybase”. When configuring this driver for use with MQSeries Integrator, in addition to its own requirements, you need to ensure that the following are set:

Defining database resources

- Under the *Advanced* tab, ensure the *Enable Describe Parameter* box is checked; and
- Under the *Performance* tab, ensure the *Prepare Method* setting is *1 - Partial*.

• Oracle8

When defining a data source for Oracle8 you must choose the driver named "MQSeries MERANT 3.7 32-BIT Oracle8". When configuring this driver for use with MQSeries Integrator, in addition to its own requirements, you need to ensure that the following are set:

- Under the *Advanced* tab, ensure the *Enable SQLDescribeParam* box is checked.

UNIX platforms: In the UNIX environment, there is no ODBC Administrator or 'Driver Manager'. To configure an ODBC data source name (DSN) definition, you must edit the required system information, which is held in a plain text file called `.odbc.ini` (note that the name of this file starts with a dot). This file must be created in the `/var/mqsi/odbc` directory which is created when MQSeries Integrator is installed. The file must have file permissions of `mqm:mqbrkr`s. A sample template is provided as the file `<mqsi_root>/mqsi/merant/odbc.ini` which contains examples of how to configure a DSN residing in each of the databases supported by MQSeries Integrator. The sample template files for the supported platforms are displayed below. They show the recommended configuration for the supported databases with keyword values which need to be configured to your local requirements in italics. Descriptions of these keywords are on 15.

• AIX

```
[ODBC Data Sources]
MQSIBKDB=IBM DB2 ODBC Driver
MYDB=IBM DB2 ODBC Driver
ORACLEDB=MERANT 3.70 Oracle8 Driver
SYBASEDB=MERANT 3.70 Sybase ASE Driver
```

```
[MQSIBKDB]
Driver=/u/db2inst1/sqllib/lib/db2.o
Description=MQSIBKDB DB2 ODBC Database
Database=MQSIBKDB
```

```
[MYDB]
Driver=/u/db2inst1/sqllib/lib/db2.o
Description=MYDB DB2 ODBC Database
Database=MYDB
```

```
[ORACLEDB]
Driver=/usr/opt/mqsi/merant/lib/UKor816.so
Description=Oracle8
ServerName=YourServerName
LogonID=scott
Password=tiger
EnableDescribeParam=1
OptimizePrepare=1
```

```
[SYBASEDB]
Driver=/usr/opt/mqsi/merant/lib/UKase16.so
Description=Sybase11
Database=sybasedb
ServerName=YourServerName
WorkstationID=id
LogonID=sa
Password=
EnableDescribeParam=1
OptimizePrepare=1
SelectMethod=0
NetworkAddress=10.30.14.72,5000
```

Defining database resources

```
[ODBC]
Trace=0
TraceFile=/var/mqsi/odbc/odbctrace.out
TraceDll=/usr/opt/mqsi/merant/lib/odbctrac.so
InstallDir=/usr/opt/mqsi/merant
```

- **Sun Solaris**

```
[ODBC Data Sources]
MQSIBKDB=IBM DB2 ODBC Driver
MYDB=IBM DB2 ODBC Driver
ORACLEDB=MERANT 3.70 Oracle8 Driver
SYBASEDB=MERANT 3.70 Sybase ASE Driver
```

```
[MQSIBKDB]
Driver=/u/db2inst1/sqllib/lib/libdb2.so
Description=MQSIBKDB DB2 ODBC Database
Database=MQSIBKDB
```

```
[MYDB]
Driver=/u/db2inst1/sqllib/lib/libdb2.so
Description=MYDB DB2 ODBC Database
Database=MYDB
```

```
[ORACLEDB]
Driver=/opt/mqsi/merant/lib/UKor816.so
Description=Oracle8
ServerName=YourServerName
LogonID=scott
Password=tiger
EnableDescribeParam=1
OptimizePrepare=1
```

```
[SYBASEDB]
Driver=/opt/mqsi/merant/lib/UKase16.sl
Description=Sybase11
Database=sybasedb
ServerName=YourServerName
WorkstationID=id
LogonID=sa
Password=
EnableDescribeParam=1
OptimizePrepare=1
SelectMethod=0
NetworkAddress=10.30.14.72,5000
```

```
[ODBC]
Trace=0
TraceFile=/var/mqsi/odbc/odbctrace.out
TraceDll=/opt/mqsi/merant/lib/odbctrac.sl
InstallDir=/opt/mqsi/merant
```

- **HP-UX**

```
[ODBC Data Sources]
MQSIBKDB=IBM DB2 ODBC Driver
MYDB=IBM DB2 ODBC Driver
ORACLEDB=MERANT 3.70 Oracle8 Driver
SYBASEDB=MERANT 3.70 Sybase ASE Driver
```

```
[MQSIBKDB]
Driver=/u/db2inst1/sqllib/lib/libdb2.sl
Description=MQSIBKDB DB2 ODBC Database
Database=MQSIBKDB
```

```
[MYDB]
Driver=/u/db2inst1/sqllib/lib/libdb2.sl
Description=MYDB DB2 ODBC Database
Database=MYDB
```

```
[ORACLEDB]
Driver=/opt/mqsi/merant/lib/UKor816.sl
Description=Oracle8
ServerName=YourServerName
LogonID=scott
Password=tiger
EnableDescribeParam=1
OptimizePrepare=1
[ODBC]
Trace=0
TraceFile=/var/mqsi/odbc/odbctrace.out
TraceDll=/opt/mqsi/merant/lib/odbctrac.so
InstallDir=/opt/mqsi/merant
```

The above files should be tailored as follows:

[ODBC Data Sources]

This section describes the data source names (DSNs) for the databases which are configured in the `.odbc.ini` file.

Driver=

This should only be tailored when using DB2, since the location of the driver library is user-defined during installation and configuration of a DB2 instance; modify the path accordingly. When using Oracle and Sybase, use the path exactly as shown.

Database=

The name of the DSN specified in the [ODBC Data Sources] section.

ServerName=

The server name for the database in which the DSN resides.

NetworkAddress=

Type the network address of your Sybase server. Specify an IP address as follows: "`<servername or IP address>, <portnumber>`". For example "`Sybaseserver,5000`". You can also specify the IP address directly such as "`199.226.224.34, 5000`". You can find the port number in the Sybase interfaces file which is usually named `'interfaces'`, `'interfac'`, or `'sql.ini'`, depending on the operating system.

Note: Environment Setup. You should ensure that the appropriate library search path environment variable (`LD_LIBRARY_PATH` on Solaris) is set to reflect the given database products to be used. Refer to your database product documentation for more details.

Note: Symbolic links. If you are going to access Oracle8 databases on Solaris you should set up the following symbolic link:

```
ln -s <ORACLE_INSTALL_DIR>/lib/libclntsh.so /usr/lib/libclntsh.so
```

Where `<ORACLE_INSTALL_DIR>` is where the Oracle8 product is installed on your machine.

Configuring databases for internal data

MQSeries Integrator creates and maintains essential configuration information in databases. When you have completed installation, you must create the following databases for this information.

- On all platforms:
 - The broker database (local persistent store)
- On Windows NT:
 - The configuration repository

Defining database resources

- The message repository

The three types of information are held in distinct database tables. You can therefore choose to create separate databases to hold these tables, or a single database to hold the tables for all three types of information. The choice depends on how your databases have been configured, not on MQSeries Integrator; your database administrator might be of assistance. When you consider how to define the database or databases to hold these tables, you must consider:

- The broker database

Broker information includes control data for resources defined to the broker (for example, deployed message flows). You can create a single database to hold information for all the brokers: a single set of tables is created when you create the first broker, and the tables are populated with rows for every broker. Every row in every table identifies the broker to which it belongs. Therefore there are no conflicts and the data is unique to each defined broker. If you prefer, you can define a separate database for each broker.

Your own enterprise data should not be stored in the database used for broker information. You should keep the data separate for operational and administration reasons, for example, when backing up your own databases and tables. In addition, the operation of the broker can be affected in certain application scenarios.

Broker tables can be created in a DB2 database, a SQL Server database, an Oracle database, or a Sybase database. If you create these tables in a DB2 database, you can use the same database for the configuration repository, or the message repository, or both.

- The configuration repository

The configuration repository holds configuration data for all components and resources defined in the broker domain, and maintains status information to indicate whether they have been assigned, or deployed, or both. The repository must be created in a DB2 database. You can create the configuration repository in the same database as the message repository, or in the same database as the broker tables (if you create these in DB2), or both.

- The message repository

Message information is all definition information created for messages defined to, or imported into, the Control Center. The repository must be created in a DB2 database. You can create the message repository in the same database as the configuration repository, or in the same database as the broker tables (if you create these in DB2), or both.

The tasks illustrated here assume you have decided to create three separate databases.

When you issue the commands that create a broker and the Configuration Manager, tables are created within the database to hold the information required by that component.

Prerequisite tasks: You must complete the following task before you start this task:

- “Defining and authorizing MQSeries Integrator user IDs” on page 9

Completing the task for a DB2 database on Windows NT: This task is illustrated using the DB2 Control Center. You are recommended to use this facility, but you can use any other method supported by DB2 (including command line or batch files) if you prefer. Refer to the DB2 documentation for details of how to do this.

Defining database resources

Step 1. Start the DB2 Control Center.

DB2 Version 7.1 does not ask for a user ID or a password. But for DB2 Version 6.1, you must enter a valid user ID and password on the Sign On dialog presented, or you can use the default DB2 administrator user ID. The user ID you use must have full administrator authority to be able to create a database.

Step 2. For each database you want to create (at least one):

- a. Expand the Object tree in the DB2 Control Center until you find Databases. Right-click Databases and select *Create Database Using Wizard (...using SmartGuide* in Version 6.1).
- b. Enter a name and alias for your database. If you have a naming convention for databases, choose a compatible name. The alias name can be the same as the database name. Database names are limited to eight characters. For example, enter MQSIBKDB.
- c. Click **Done**.

When you have completed these steps for every database you created, click **OK**.

Note: If you are using DB2 Version 7.1, you should increase the database heap size to at least 900 (in units of 4KB pages).

On Windows NT, this operation can be carried out using the DB2 Control Center. Expand the object tree until you find the database, then right click on the database name. From the pop up menu, select *Configure... —> Performance —> Database heap size*. The default value is typically 600 (4KB pages). You can reset this within the range 32 to 60000, but should choose a value of at least 900.

You should refer to the product specific database documentation for more information or assistance on this task. Your Database Administrator might also be able to offer advice and assistance.

Completing the task for a DB2 database on UNIX platforms:

Step 1. Logon as root, and create a database instance using, for example, the following command on Sun Solaris or HP-UX:

```
/opt/IBMd2/V7.1/instance/db2icrt -u <fence userID> <username>
```

or the following on AIX:

```
usr/lpp/db2_07_01/instance/db2icrt -u <fence userID> <username>
```

Step 2. Logon as <username> and then create a database using the following commands (note that on some platforms, an explicit path name is required):

```
./sqllib/db2profile
db2start
db2 create database MQSIBKDB
db2 connect to MQSIBKDB
db2 bind ~/sqllib/bnd/@db2cli.lst grant public CLIPKG 5
```

Note: If you are using DB2 Version 7.1, you should increase the database heap size, using the DB2 command line. For example:

```
UPDATE DATABASE CONFIGURATION FOR MQSIBKDB USING DBHEAP 900
```

Defining database resources

The default value is typically 600 (4KB pages). You can reset this within the range 32 to 60000, but should choose a value of at least 900.

Completing the task for a SQL Server database (Windows NT only): If you are using SQL Server for the broker database, you must complete the following steps:

- Step 1. Invoke the SQL Server Enterprise Manager.
- Step 2. Create the database.

For further information about how to define your SQL Server databases, refer to the product documentation. Your Database Administrator might also be able to offer advice and assistance.

Subsequent tasks: When you have completed this task, continue with:

- “Defining internal MQSeries Integrator database connections (Windows NT only)”

Defining internal MQSeries Integrator database connections (Windows NT only)

MQSeries Integrator connects to its internal databases as follows:

- It uses the Open Database Connectivity (ODBC) to connect to the broker database and to the message repository. Each database product provides an ODBC driver.
- It uses the Java Database Connectivity (JDBC) to connect to the configuration repository.

The ODBC connections require an additional definition. The JDBC™ connection does not. You must remember to create the additional definition for each broker database, if you use more than one.

Prerequisite tasks: You must complete the following tasks before you start this task:

- “Configuring databases for internal data” on page 15

Completing the task for a DB2 database: To establish the required ODBC access for a DB2 database, you must:

- Step 1. From the Windows NT Start menu, select Settings —> Control Panel. Within the Control Panel, double-click ODBC. Click the *System DSN* tab.
- Step 2. Click **Add**. The *Create New Data Source* window appears.
- Step 3. Double-click IBM DB2 ODBC DRIVER.
- Step 4. Enter the data source name (the database name) and the database alias.
- Step 5. Click **OK**.

You must refer to the product specific database documentation for more information or assistance on this task. Your Database Administrator might also be able to offer advice and assistance.

Completing the task for a SQL Server database: To establish the required ODBC access for a SQL Server database, you must:

- Create the ODBC DSN reflecting the values you have defined for this database according to the instructions in the documentation you have for SQL Server.

For further information about how to define an ODBC connection for SQL Server databases, refer to the product documentation. Your Database Administrator might also be able to offer advice and assistance.

Subsequent tasks: When you have completed this task, continue with:

- “Authorizing internal database access”

Authorizing internal database access

When you have created your internal databases, you must authorize a certain user ID to access those databases.

Prerequisite tasks:

- “Configuring databases for internal data” on page 15

Completing the task for a DB2 database: To authorize the user ID to be used for access to a DB2 database, you must:

Step 1. Start the DB2 Control Center, if it is not already active. Log on with the DB2 administrator user ID (the default is `dbadmin`).

Step 2. Complete the following tasks for each database you created in “Configuring databases for internal data” on page 15:

- Expand the object tree until you find the database.
- Expand the tree under this database and left-click the *User and Group Objects* folder. The *DB Users* and *DB Groups* folders are displayed in the right pane.
- Right-click the *DB Users* folder in the right pane and select *Add* from the pop-up menu. The Add User notebook opens.
- Select the user ID you want to authorize to access the database (for example, `mqsuid`) from the drop-down list. This must be:
 - For the configuration repository, the user ID you specify with the flag `-u` when you create the Configuration Manager.
 - For the message repository, the user ID you specify with the flag `-e` when you create the Configuration Manager.
 - For the broker tables, the user ID you specify with the flag `-u` when you create the broker.

Select the appropriate options from the choices in the box labelled *Choose the appropriate authorities to grant to the selected user*, for all the databases you have created for MQSeries Integrator.

This user ID must have the following authority to select each of the databases you have created for MQSeries Integrator:

- Connect database.
- Create tables.
- Create packages.
- Register functions to execute in database manager’s process.

e. Click **OK**. The authorities are granted. The dialog is closed.

Step 3. Close the DB2 Control Center.

Step 4. On UNIX platforms, you must also invoke the following command (having logged on as root) to set the appropriate environment variables:

```
. ~/sql1lib/db2profile
```

If you need further guidance about any of these tasks, use the online help facility of the DB2 Control Center. Your Database Administrator might also be able to offer advice and assistance.

Defining database resources

Completing the task for a SQL Server database (Windows NT only): To authorize the user ID to be used for access to the broker's SQL Server database, you must:

1. Create a SQL Server login ID, defining the Server role as *Systems Administration*. You must specify this user ID as the user ID that will access this database on the **mqsicreatebroker** command (or in the Command Assistant), using the **-u** flag or the **-i** flag (in task "Creating an MQSeries Integrator broker").
2. Specify that this new user ID can access your new broker database and indicate the permissions allowed.

For further information about how to authorize access to your SQL Server databases, refer to the product documentation. Your Database Administrator might also be able to offer advice and assistance.

Subsequent tasks: When you have completed this task, continue with:

- "Defining MQSeries Integrator components"

Configuring databases for NEON message formats

MQSeries Integrator also supports message formats defined by the NEONFormatter GUI. These message formats can be stored in the following databases:

- DB2
- SQL Server
- Oracle
- Sybase

The levels at which these databases are supported are the same as those shown in Table 1 on page 12.

For information about how you must configure these databases, see "Appendix B. Using NEON" on page 195.

Defining MQSeries Integrator components

This section describes how to create the MQSeries Integrator components.

Prerequisite tasks

You must complete the following task before you start this task:

- "Authorizing internal database access" on page 19

Completing the task

The steps you take to complete this task depend on the configuration of your broker domain. The minimum you must define are the Configuration Manager and one broker. The User Name Server is optional and is required only if you plan to implement topic-based security.

Creating the MQSeries Integrator Configuration Manager (Windows NT only):

To create the Configuration Manager you must use the command described in "mqsicreateconfigmgr (Create Configuration Manager)" on page 111 or the Command Assistant. The parameters on this command provide the Configuration Manager with all the additional information it requires to be ready for action as soon as it is started, including the identifiers for both the configuration repository and the message repository.

Creating an MQSeries Integrator broker: You must create the broker on the system on which you have installed the broker component. You must give the broker a unique name, for example, MQSI_SAMPLE_BROKER. Remember that, on

UNIX platforms, broker names are case sensitive. You can use the command “mqsicreatebroker (Create broker)” on page 105 or the Command Assistant (see “Chapter 7. Using the MQSeries Integrator Command Assistant” on page 87).

Creating the MQSeries Integrator User Name Server: You must create the User Name Server on the system on which you have installed the User Name Server component. You can use the command described in “mqsicreateusernameserver (Create User Name Server)” on page 117 or the Command Assistant.

Subsequent tasks

When you have completed this task:

- If the component or components you have created do not share a queue manager with other components, continue with “Connecting two MQSeries Integrator components” on page 22.
- If all of your MQSeries Integrator components share a single queue manager, continue with “Initialization tasks” on page 24.

Defining MQSeries resources

All MQSeries resources, apart from those required to connect together MQSeries Integrator components that are supported by different queue managers (whether on the same or on different physical systems) are defined for you when you define the MQSeries Integrator components that depend on them.

For details of connecting components, see “Connecting two MQSeries Integrator components” on page 22.

Connection tasks

The components of the MQSeries Integrator system (brokers, the Configuration Manager, and the User Name Server) communicate using MQSeries facilities. The Control Center clients also use MQSeries connections to the Configuration Manager. The exact requirements for connecting the components together depend largely on the way in which you set up your environment.

The connection tasks described are:

- “Connecting Control Center clients to the Configuration Manager (Windows NT only)”
- “Connecting two MQSeries Integrator components” on page 22

Connecting Control Center clients to the Configuration Manager (Windows NT only)

You can start up multiple instances of the Control Center on one or more systems. Each of them must be able to connect to the Configuration Manager’s queue manager using an MQSeries Client for Java connection. The Configuration Manager’s end of the connection is defined when you create the Configuration Manager (see “Creating the MQSeries Integrator Configuration Manager (Windows NT only)” on page 20), and the Control Center end of the connection is defined when you start up the Control Center for the first time (see “Starting the Control Center (Windows NT only)” on page 27).

You must take one extra step to enable the two ends of the connection to communicate successfully.

Control Center connections

Prerequisite tasks

You must complete the following task before you start this task:

- “Defining MQSeries Integrator components” on page 20

Completing the task

To enable communications you must:

Step 1. Start the listener on the Configuration Manager’s queue manager. You can use either one of two methods to do this:

- a. You are recommended to use MQSeries Services (*Start* → *Programs* → *IBM MQSeries* → *MQSeries Services*). Left-click the queue manager (for example, MQSI_SAMPLE_QM) to see its services in the right-hand pane. If the Listener is listed, right-click the Listener, and select *All Tasks* → *Start*. This starts the listener as a background task. If the Listener is not listed, right-click the queue manager and select *New* → *Listener*. This creates a listener with default properties of transport type TCP and port 1414. When it has been created, right-click the Listener and select *Start*.

This starts the listener as a background task.

- b. If you prefer, you can use the `runmq1sr` command. For example:

```
runmq1sr -t tcp -p 1414 -m MQSI_SAMPLE_QM
```

When you use this command the listener is started as a foreground task and is not displayed in the MQSeries Services window.

Note: If the default MQSeries port 1414 is not available (perhaps because it is already in use by another queue manager), you must assign a different port number that is suitable. The port value must be set in the Listener properties dialog (Parameters tab), or as the `-p` parameter on the `runmq1sr` command. If the port is already in use, the Control Center will not be able to contact the Configuration Manager. For example, if you have set up a default queue manager on this system, it probably uses port 1414 for its listener. You can check what listeners are already active using MQSeries Services.

Subsequent tasks

When you have completed this task, continue with:

- “Connecting two MQSeries Integrator components”

Connecting two MQSeries Integrator components

If you have defined all the components in your broker domain on a single queue manager (task “Defining MQSeries Integrator components” on page 20), you can omit this step and continue with task “Initialization tasks” on page 24.

If the components in your broker domain are supported by different queue managers, you must establish MQSeries connections between those queue managers to enable messages to be exchanged if necessary. Message exchange is required between:

- Every broker and the Configuration Manager.
- Every broker and the User Name Server, if you have one in your broker domain.
- The Configuration Manager and the User Name Server, if you have one in your broker domain.

Figure 1 on page 23 illustrates the basic requirements for this type of connection. You are recommended to read “General guidance” on page 32 before you complete this task for further guidance on completing this task.

Connecting two components

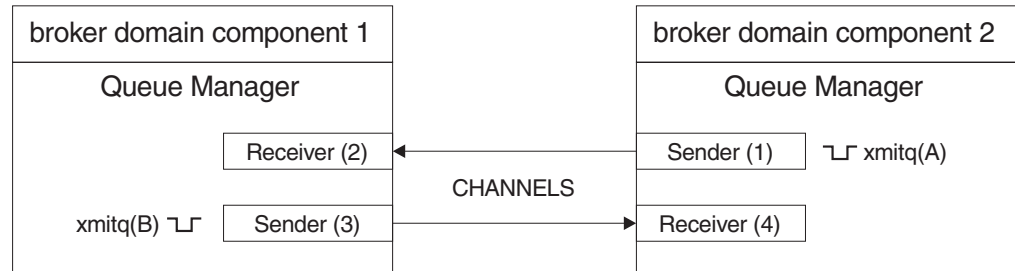


Figure 1. Connecting two MQSeries Integrator components

Prerequisite tasks

Before you start this task, the following task must be complete for the components you want to connect:

- “Defining MQSeries Integrator components” on page 20

Completing the task

When you create an MQSeries Integrator component, you identify the queue manager that supports it. If this queue manager does not already exist, it is created at the same time. A number of mandatory fixed-name MQSeries resources are created on the supporting queue manager when the MQSeries Integrator component is created. These are listed in “mqsicreatebroker (Create broker)” on page 105, “mqsicreateconfigmgr (Create Configuration Manager)” on page 111, and “mqsicreateusername server (Create User Name Server)” on page 117.

All the steps here are illustrated with MQSC examples. You can use any appropriate method for defining these resources. These examples assume that the queue managers are called COMP1 and COMP2.

To achieve the connection illustrated in Figure 1, you must complete the following steps:

- Step 1.** Define a transmission queue on each component’s queue manager (xmitq(A) and xmitq(B)). These queues will collect messages ready for transmission between components.

For example, on queue manager COMP1:
`define qlocal('COMP2') usage(XMITQ) replace`

and on queue manager COMP2:
`define qlocal('COMP1') usage(XMITQ) replace`

- Step 2.** Define the sender channel on the first component’s queue manager (Sender(3)). This will transport messages sent by the first component to the second component.

You must allocate connection names according to your MQSeries network conventions, and you must specify the protocol you are using for this connection and the port on which the listener is listening.

For example, on queue manager COMP1:
`define channel('COMP1_TO_COMP2') chltype(sdr) trptype(tcp)
conname('MQSISYS1(1415)') xmitq('COMP2') replace`

- Step 3.** Define a receiver channel on the first component’s queue manager (Receiver(2)). Messages sent by the second component to the first will be received by this channel.

Connecting two components

This receiver channel must have the same name as the sender channel on COMP2, defined in Step 4. For example, on queue manager COMP1:

```
define channel('COMP2_TO_COMP1') chltype(rcvr) trdtype(tcp) replace
```

- Step 4. Define the sender channel on the second component's queue manager (Sender(1)). This will transport messages sent by the second component to the first component.

You must allocate connection names according to your MQSeries network conventions, and you must specify the protocol you are using for this connection.

For example, on queue manager COMP2:

```
define channel('COMP2_TO_COMP1') chltype(sdr) trdtype(tcp)
conname('MQSISYS1(1414)') xmitq('COMP1') replace
```

- Step 5. Define a receiver channel on the second component's queue manager (Receiver(4)). Messages sent by the first component to the second will be received by this channel.

This receiver channel must have the same name as the sender channel on COMP2, defined in Step 2 on page 23. For example, on queue manager COMP2:

```
define channel('COMP1_TO_COMP2') chltype(rcvr) trdtype(tcp) replace
```

- Step 6. Create and start a listener for each protocol in use ("Connecting Control Center clients to the Configuration Manager (Windows NT only)" on page 21 illustrates this task).

- Step 7. Start the sender channels (1) and (3) on the respective queue managers. (You are recommended to set up channel initiators to do this automatically).

Subsequent tasks

When you have completed this task, continue with:

- "Initialization tasks"

Initialization tasks

When you have completed the setup tasks required to prepare a broker domain for operation, you must initialize the components to establish a working system. The tasks covered in this section are:

- "Starting MQSeries queue managers as a Windows NT service"
- "Starting the Configuration Manager (Windows NT only)" on page 25
- "Starting a broker" on page 26
- "Starting the User Name Server" on page 26
- "Starting the Control Center (Windows NT only)" on page 27

The examples used to illustrate these tasks assume that a broker called MQSI_SAMPLE_BROKER is defined on queue manager MQSI_SAMPLE_QM, and the queue manager MQSI_SAMPLE_QM2 supports the Configuration Manager.

Starting MQSeries queue managers as a Windows NT service

When you start the MQSeries Integrator components using the `mqsistart` command (illustrated in the following sections), the command will start the queue manager for this component if it is not already running.

Although the MQSeries Integrator component is started as a service on Windows NT, the queue manager is not. You can change the properties of the queue manager service to set start up type to automatic to enable the queue manager to

run as a service. This is an optional task. If you do not want to complete this task, you can continue with “Starting the Configuration Manager (Windows NT only)”.

Prerequisite tasks

You must complete the following task before you start this task:

- “Defining MQSeries Integrator components” on page 20

Completing the task

To complete this task you must:

- Step 1. End the queue managers for the Configuration Manager, the User Name Server (if you have defined one in your broker domain), and your brokers, using the **endmqm** command or MQSeries Services. The queue managers are already running, because they are started when you create the MQSeries Integrator components.
- Step 2. Select *Start* —> *Programs* —> *IBM MQSeries* —> *MQSeries Services*.
- Step 3. Right-click each queue manager you want to change.
- Step 4. Select *Properties*, and the General tab.
- Step 5. Update the Startup Type to Automatic.
- Step 6. This setting ensures that the queue manager is started whenever the MQSeries Service (a Windows NT service) is started.
- Step 7. You are also recommended to change the properties of the MQSeries Services service by updating its Startup Type to automatic using the Control Panel. This setting starts MQSeries Services when Windows NT itself starts up. This isolates the operation of the MQSeries Services service from any logged on user.
- Step 8. Restart the queue managers for the Configuration Manager, the User Name Server (if you define one in your broker domain), and your brokers, using the **strmqm** command or MQSeries Services. You must do this before you start the MQSeries Integrator components.

The changes to the queue managers’s start up type will take effect when you restart Windows NT.

Subsequent tasks

When you have completed this task, continue with:

- “Starting the Configuration Manager (Windows NT only)”
- “Starting a broker” on page 26
- “Starting the User Name Server” on page 26

Starting the Configuration Manager (Windows NT only)

The Configuration Manager is started as a Windows NT service.

Prerequisite tasks

You must complete the following task before you start this task:

- “Defining MQSeries Integrator components” on page 20

Completing the task

Start your Configuration Manager as follows:

- Step 1. Issue the following command at the command line (you cannot do this using the Command Assistant):

```
mqsisstart configmgr
```

Starting the Configuration Manager

This command initiates the start up of the Configuration Manager's Windows NT service and can only report on whether that service is started successfully.

- Step 2. Check the Application view of the Windows NT Event Viewer to ensure that the Configuration Manager has initialized successfully.

If you have not already started the queue manager, it is started by this command.

Subsequent tasks

When you have completed this task, continue with:

- "Starting a broker"

Starting a broker

The broker is started as a Windows NT service or as a background process on UNIX platforms.

Prerequisite tasks

- "Defining MQSeries Integrator components" on page 20

Completing the task

Start your broker as follows:

- Step 1. Issue the following command on the command line (you cannot do this using the Command Assistant). For example:

```
mqsistart MQSI_SAMPLE_BROKER
```

This command initiates the start up of the broker's Windows NT service or UNIX daemon and can only report on whether that process started successfully.

- Step 2. Check the Application view of the Windows NT Event Viewer or the UNIX syslog to ensure that the broker has initialized and continues to run successfully.

If you have not already started the queue manager, it is started by this command.

The Configuration Manager cannot contact a broker until a reference to that broker has been defined in the configuration repository, and the topology deployed. You must create this reference to the broker using the Control Center. This task is described in "Defining and deploying the configuration in the Control Center" on page 28.

Subsequent tasks

When you have completed this task, continue with:

- "Starting the User Name Server"

Starting the User Name Server

If you have not defined a User Name Server in your broker domain (task "Creating the MQSeries Integrator User Name Server" on page 21), you can omit this step and continue with task "Starting the Control Center (Windows NT only)" on page 27.

The User Name Server is started as a Windows NT service or as a background process on UNIX platforms.

Prerequisite tasks

- "Defining MQSeries Integrator components" on page 20

Completing the task

Start your User Name Server as follows:

- Step 1. Issue the following command on the command line (you cannot do this using the Command Assistant), noting that on UNIX platforms, it is case-sensitive:

```
mqsisstart UserNameServer
```

This command initiates the start up of the User Name Server's Windows NT service or UNIX daemon and can only report on whether that process started successfully.

- Step 2. Check the Application view of the Windows NT Event Viewer or the UNIX syslog to ensure that the User Name Server has initialized and continues to run successfully.

If you have not already started the queue manager, it is started by this command.

Subsequent tasks

When you have completed this task, continue with:

- "Starting the Control Center (Windows NT only)"

Starting the Control Center (Windows NT only)

The basic tasks to define your broker domain are now complete. You must now initialize the Control Center (on a Windows NT system) so that you can continue with your configuration by defining your configuration to the Configuration Manager, and thus to the configuration repository.

- Step 1. Start the Control Center by double-clicking the Control Center icon in the MQSeries Integrator program folder, or by using the Windows NT Start menu (*Start* → *Programs* → *IBM MQSeries Integrator 2.0* → *Control Center*).

- Step 2. Complete the initial dialog presented by the Control Center, *Configuration Manager Connection*, to provide the information needed to connect your Control Center session to the Configuration Manager. The fields are:
- a. Hostname. This is initially blank. Enter the network hostname of the system on which the Configuration Manager has been created. For example, the system MQSISYS1 was used in the channel definitions in "Connecting two MQSeries Integrator components" on page 22.
 - b. Port. This is initially blank. Enter the number of the port on which the queue manager is listening. The default value is 1414. You must use the port number that is in use by the Configuration Manager's queue manager: check which port you used when you started the listener ("Connecting Control Center clients to the Configuration Manager (Windows NT only)" on page 21).
 - c. Queue Manager name. This is initially blank. Enter the name of the Configuration Manager's queue manager (for example, MQSI_CONFIG_QM). This queue manager already has a definition for the server connection required by the Control Center (the channel SYSTEM.BKR.CONFIG of type SVRCONN), that was created when the Configuration Manager was created.

When you have completed these fields, click **OK**. The Control Center now contacts the Configuration Manager, which might take a few minutes.

Check for MQSeries or MQSeries Integrator entries in the Windows NT Event log (Application view) to check the success of this connection.

Starting the Control Center

If you want to check, or change, these settings at a later time, click *File* —> *Connection* to bring up the connection dialog.

- Step 3. Set your user role. This must be appropriate for the tasks you will perform, and have been authorized to complete (see “Defining and authorizing MQSeries Integrator user IDs” on page 9). The default is “All roles”.

Complete the following steps:

- a. Click *File* —> *Preferences*.
- b. Click *User’s role* and select the appropriate role (see *MQSeries Integrator Using the Control Center* for details of user roles in the Control Center).
- c. Click **OK** to save this setting. This dismisses the dialog and returns you to the main window of the Control Center where you will see the views appropriate to the role selected.

- Step 4. If you want, you can access the *Log* view. This view provides feedback on all the deploy actions you take in the Control Center.

The *Log* view is initially empty. You must click the *Refresh* button (top left hand end of the toolbar) each time you check the *Log* view to ensure you have the latest information.

Defining and deploying the configuration in the Control Center

You must now define your topology to the Configuration Manager using the Control Center. This saves the definitions you make in the configuration repository, and allows you to deploy the changes. When you deploy, the Configuration Manager contacts the brokers that are affected by the changes, and sends them messages that define their updated configuration. The brokers receive the messages, make the changes, and respond to the Configuration Manager which passes on the results of the deploy to the Control Center.

Prerequisite tasks

You must complete the following task before you start this task:

- “Initialization tasks” on page 24

Completing the task

To save your changes in the configuration repository, and deploy them through the broker domain, you must:

- Step 1. Check out the broker domain topology. This locks the topology and allows you to make changes to it.
- Step 2. Select *Create* —> *Broker*. This displays a dialog where you can enter the name of the broker you want to add to the topology. You must specify the name that you used to create the broker (in “Creating an MQSeries Integrator broker” on page 20). You must also enter the same queue manager name as the queue manager name you specified when you created the broker. Repeat this step for each broker you have created.
- Step 3. You must now save the changes that you have made. Select *File* —> *Check In* —> *All (Save to Shared)*. This causes two things to happen:
 - a. The contents of the configuration repository are updated with the new definitions and assignments and everything is checked in to the repository.
 - b. The updated workspace is saved locally. If you are working with a new workspace (the title bar indicates this by displaying *Untitled*), you are asked for a name for this workspace. Enter a name (for example *Topology1*) and click **Save**. This name now appears in the title bar.

Deploying the configuration

- Step 4. Now you must deploy your changes. When you deploy, the Configuration Manager sends information to the brokers about the resources needed to support the message flow services.
- Step 5. Select the *Log* view and refresh the contents by clicking the green refresh icon. It can take a few minutes for all the deployment messages and responses flowing between the Configuration Manager and the brokers to be displayed. Keep refreshing this view until you see the completion messages.

For more details of how to complete all these steps, refer to the Control Center online help, or to *MQSeries Integrator Using the Control Center*.

Subsequent tasks

When you have completed this task, continue with:

- “Application client and user data tasks”

Application client and user data tasks

The following sections provide general information for setting up application clients and user databases.

- “Setting up application clients”
- “Configuring databases for user data accessed from message flows” on page 30

In this context, the term application client refers to an application program written to the AMI or MQI. It uses the services provided by the message flows deployed within one or more brokers in the broker domain by interacting with the queues serviced by those message flows.

Setting up application clients

When you have identified the applications that are to interact with message flows, you can decide where those applications will execute. Because MQSeries Integrator clients must use MQSeries facilities to connect to the broker, and to interact with it (using the MQI and AMI), the setup of clients for MQSeries Integrator is identical to that for clients for an MQSeries server.

Application clients can use one of two techniques for gaining access to a broker’s services:

- An application can use an MQSeries client connection. The application can be running on the same system as the queue manager to which it connects, or on a different system. It can connect to a queue manager supporting a broker, or to any other queue manager in the MQSeries network that has a defined path to the broker’s queue manager.

You can use all of the MQSeries clients supported by MQSeries Version 5.1, giving you the freedom to connect applications running in a wide variety of environments into your broker domain.

- An application can use a local connection to a queue manager. If it uses this method, the client must execute on the same system. It can connect to a queue manager supporting a broker, or to any other queue manager in the MQSeries network that has a defined path to the broker’s queue manager.

For more details about applications, putting and getting messages, and the use of MQSeries clients, see *MQSeries Clients* and the *MQSeries Application Programming Guide*. MQSeries Integrator does not impose any particular conditions or restrictions on applications.

Application clients

MQSeries resources for client applications

An application client can run on a system anywhere in the MQSeries network. The application can access MQSeries Integrator services in two ways.

1. The application can make a local connection to either:
 - The broker's queue manager
You do not have to define any MQSeries resources to support this client configuration.
 - Another queue manager in the network
You must ensure that definitions are in place to support communications between the queue manager to which the client has connected and the queue manager that hosts the broker that provides the required service.
2. The application can make an MQSeries client connection to either:
 - The broker's queue manager
You must set up the appropriate client connection and server connection definitions to support this option.
 - Another queue manager in the network
You must set up the appropriate client connection and server connection definitions to support this option, and ensure that definitions are in place to support communications between the queue manager to which the client has connected and the queue manager that hosts the broker that provides the required service.

An application can only get messages from queues owned by the queue manager to which it is connected (this is true for all MQSeries applications). Therefore, if an application expects to receive messages from a queue populated by a service within a particular broker and owned by that broker's queue manager, it must connect to that broker's queue manager (using a local or MQSeries client connection).

An application that puts messages, however, can be connected to any queue manager in the network, as long as the queue manager can resolve the target destination in some way. In all cases, the queue manager to which the client application is connected must know the whereabouts of the queue or queues to which the application puts messages (for example using remote queue definitions).

Configuring databases for user data accessed from message flows

The following databases are supported for message flow access from message processing nodes.

- DB2
- SQL Server
- Oracle
- Sybase

The following message processing nodes allow database access:

- Compute
- Database
- Filter
- DataInsert
- DataDelete
- DataUpdate
- Warehouse

Complete the following steps to set up these databases:

- Step 1.** Create the database. If you are using DB2 or SQL Server, you can follow the instructions in "Configuring databases for internal data" on page 15.
- Step 2.** From the Control Panel, select ODBC and create a Data Source Name for the database you have created. If you are using DB2 or SQL Server, you can follow the instructions in "Configuring databases for internal data" on page 15.
- Step 3.** Set the *Data Source* property of the node to the DSN for the appropriate database.

For more details of message processing nodes, how to set properties, and how to use them in message flows, see *MQSeries Integrator Version 2.0.2 Using the Control Center*. For additional information about coordinated transactions, see "Coordinated transactions" on page 39.

Message flow transactionality and database design

It is possible to mix the transaction types of nodes that operate on external databases. That is, some nodes in a message flow might specify "automatic" transactionality, meaning that any work they do is not committed until the message flow successfully completes, and some might specify "commit" transactionality, meaning that any work they do is committed, regardless of the subsequent success or failure of the message flow.

In order to mix nodes with "automatic" and "commit" transactionality in the same message flow, where the nodes operate on the same external database, you must use separate ODBC connections; one for the nodes which are not to commit until the completion of the message flow, and one for the nodes which are to commit immediately. Otherwise the nodes which commit immediately will also cause all operations carried out by preceding "automatic" nodes to be committed as well.

You should be aware that using more than one ODBC connection to the same database requires care as it can lead to locking problems. In particular, if an "automatic" node carries out an operation, such as an INSERT or an UPDATE, which causes a database object (such as a table) to be locked, and a subsequent node tries to access that database object using a different ODBC connection, an infinite lock (deadlock) will occur. The second node will wait for the lock acquired by the first to be released, but the first node will not commit its operations and release its lock until the message flow completes - which will never happen because the second node is waiting for the first node's database lock to be released. Such a situation will not be detected by any DBMS automatic deadlock-avoidance routines because the two operations are interfering with each other indirectly, via the broker.

There are two ways to avoid this sort of locking problem:

- Design your message flow so that uncommitted ("automatic") operations do not lock database objects which subsequent operations using a different ODBC connection need to access.
- Configure your database's lock time-out parameter so that an attempt to acquire a lock fails after a specified length of time. If a database operation fails due to a lock time-out, an exception will be thrown which the broker will handle in the normal way.

For information concerning which database objects are locked by particular operations, and how to configure your database's lock time-out parameter, consult your database product documentation.

General guidance

In general, you can set up your MQSeries infrastructure as you want. Any exceptions to this are documented in the previous sections. However, there are some general guidelines that you can follow when you set up your configurations. For more specific guidance, you must refer to the MQSeries documentation (for example, the *MQSeries Intercommunication* book, and the *MQSeries Command Reference*).

- On Windows NT, you might want to change the properties of the queue managers that you define to support MQSeries Integrator components so that they run as part of the MQSeries Windows NT service. You can also change the properties of the MQSeries Service to start up automatically when the Windows NT system starts or restarts. These tasks are described in “Starting MQSeries queue managers as a Windows NT service” on page 24.
 - You are recommended to use sender-receiver pairs of channels for all two-way communications between queue managers that host MQSeries Integrator components. These are used in the task description in “Connecting two MQSeries Integrator components” on page 22.
 - For simple distributed queueing configurations, you are recommended to use the name of the target queue manager as the name of the transmission queue through which messages to that queue manager are sent. This is illustrated in the task description in “Connecting two MQSeries Integrator components” on page 22.
 - You can set up channel initiators for the channels between the queue managers supporting your MQSeries Integrator components. This will reduce overhead by allowing the channels to stop when there is no message traffic, but ensure automatic start up when there are messages to transport.
 - You can set up a single receiver channel on the Configuration Manager’s queue manager to support all sender channels created for the brokers. This requires a single definition on the Configuration Manager and a single sender definition on each broker, which would therefore have to have the same name on each broker. You can also use this receiver channel on the Configuration Manager to support communications from the User Name Server.
- Likewise, you can set up a single receiver channel on the User Name Server to support communications from every broker that connects to it.

- All MQSeries connections between MQSeries Integrator components, and between clients and MQSeries Integrator components, can be set up using any of the communications protocols supported by MQSeries (TCP/IP and SNA on all platforms; also, NetBIOS and SPX on Windows NT), with the following exception:
 - The client/server connection between the Control Center and the Configuration Manager must be a TCP/IP connection.
- You are strongly recommended to set up a dead letter queue (DLQ) on every queue manager. The DLQ is referenced by MQSeries Integrator when errors occur processing messages in message flows. If an MQSeries queue manager is created as a result of using the **mqsicreatebroker** or **mqsicreateconfigmgr** commands, the default DLQ provided by MQSeries (SYSTEM.DEAD.LETTER.QUEUE) is automatically enabled for you.
- You can use MQSeries clusters if you choose. This will simplify your configuration in most cases. Refer to the *MQSeries Integrator Introduction and Planning* book for specific guidance on the use of clusters with MQSeries Integrator.

General guidance

- You can use MQSeries events to provide additional information in analyzing your system and resolving any problems. See “Part 1 Events monitoring” of the *MQSeries Programmable System Management* book for more details about these events.
- The MQInput node defaults to getting messages from the input queue without specifying conversion. You must check each message flow to ensure you set an appropriate value for the conversion property for the MQInput node. For more details about node properties, see *MQSeries Integrator Version 2.0.2 Introduction and Planning*. For more details about conversion of messages, see *MQSeries Application Programming Reference*.

General guidance

Chapter 3. How to manage your MQSeries Integrator network

This chapter discusses the following aspects of broker domain management:

- “Managing the broker domain components”.
- “Managing workload and performance” on page 49.

Managing the broker domain components

When you have configured your broker domain, you must manage it to gain the most robust operation and the most efficient use of the components.

- “Managing components”
- “Managing databases” on page 36
- “Enhancing and updating your broker domain” on page 37
- “Recovery and restart” on page 42

Managing components

When your configuration work is complete, you need to manage the components on a day-to-day basis. MQSeries Integrator provides a set of commands that enable you to control the broker domain in two ways.

Starting and stopping components

You can use the following commands to start and stop components:

- “mqsistart (Start component)” on page 148. This command starts the Configuration Manager, the User Name Server, and the brokers you have created. You must identify which component is to be started as the first parameter on the command. The associated queue manager is also started, if it is not already running.
- “mqsistop (Stop component)” on page 150. This command terminates the component specified by the first parameter on this command. You can also request that the associated queue manager is stopped by this command.

Stopping queue managers: When you invoke the **mqsistop** command and specify the **-q** flag, the command initiates a controlled shutdown of the queue managers on which the MQSeries Integrator component identified on the command depends. The command cannot complete until shutdown of the queue manager has completed. All other users of the queue manager are notified that it is shutting down, and should respond by closing down (and therefore disconnecting) themselves.

If you are using a single queue manager to support more than one MQSeries Integrator component (the Configuration Manager and the User Name Server can share a queue manager, and a single broker can also be defined on the same queue manager as the Configuration Manager, or the User Name Server, or both) you are recommended to stop each component and only specify the **-q** flag on the final stop command.

Viewing and modifying components

Several commands are provided that allow you to view the existence and status of components on your system. Commands are also supplied to allow you to modify a large number (but not all) of the attributes of a component, although you must stop the component before you can do so.

Managing components

Listing components: You can use the command `mqsilist` to return a list of the components created on this system, with the name of the queue manager that supports them.

You can also request a list of a broker's execution groups and message flows. Execution groups are always returned, but the message flows can only be returned if the broker is currently active.

Modifying components: If you want to update the parameters currently set for a component, you must first stop that component, and then you can either:

- Use the MQSeries Integrator Command Assistant. For a description of the Command Assistant, see "Chapter 7. Using the MQSeries Integrator Command Assistant" on page 87.
- Use the `mqsichangebroker`, `mqsichangeconfigmgr`, or `mqsichangeusernameserver` command. These commands are all described in "Chapter 8. Commands" on page 89.

Managing databases

This section describes some aspects of database management within your broker domain.

Databases and code pages

Subscription data retrieved from client applications (for example, topics from publishers and subscribers, and content filters from subscribers) and the character data entered using the Control Center (for example, message flow names) are stored in the configuration and message repositories. This data is translated from its originating code page to the code page of the process in which the broker or Configuration Manager is running, and then by the database manager to the code page in which the database or databases were created.

To preserve data consistency and integrity, you must ensure that all this subscription data and Control Center character data is originated in a compatible code page to the two code pages to which it is translated. If you do not do so, unpredictable results and loss of data might result.

Data stored in the broker's local persistent store is not affected in this way.

The restrictions described above are not applicable to user data in messages. It is your responsibility to ensure that any data in messages generated by your applications is compatible with the code page of any database you access from your message flows.

SQL statements generated as a result of explicit reference to databases within message processing nodes can contain character data that has a variety of sources. For example, it might have been entered through the Control Center, derived from message content, or read from another database. All this data is translated from its originating code page to the code page in which the broker was created, and then by the database manager to the code page in which the database was created. You must also ensure that these three code pages are compatible to avoid data conversion problems.

Managing database access

You can change access to databases by components if you want to, by taking the following actions:

- If you want to change the user ID through which the Configuration Manager accesses the configuration repository (the `DataBaseUserID` parameter on **`mqsicreateconfigmgr`**) or message repository (the `MRMDataSourceUserID` parameter on **`mqsicreateconfigmgr`**) you must:
 - Back up the database tables in the repository (these are listed in “`mqsicreateconfigmgr` (Create Configuration Manager)” on page 111).
 - Back up the workspace, by saving it from the Control Center and backing up the XML document created by the export, and the message sets stored in the message repository, as described in “Importing and exporting message sets” on page 41.
 - Stop the Configuration Manager using the **`mqsistop`** command.
 - Delete the Configuration Manager (command **`mqsdeleteconfigmgr`** or the Command Assistant).
 - Restore the database tables to the new location.
 - Recreate the Configuration Manager specifying the new user ID (or IDs) on the **`mqsicreateconfigmgr`** command (or the Command Assistant).
- If you want to change the user ID through which a broker accesses its database tables (the `DataSourceUserID` parameter on the **`mqsicreatebroker`** or the Command Assistant, you must:
 - Back up the database tables (these are listed in “`mqsicreatebroker` (Create broker)” on page 105).
 - Back up the workspace, by saving it from the Control Center and backing up the XML document created by the export, and the message sets stored in the message repository, as described in “Importing and exporting message sets” on page 41.
 - Stop the broker using the **`mqsistop`** command.
 - Delete the broker following the instructions provided in “Deleting components from the broker domain” on page 40).
 - Restore the database tables to the new location.
 - Recreate the broker specifying the new user ID on the **`mqsicreatebroker`** command or the Command Assistant.

Enhancing and updating your broker domain

When you have completed your initial broker domain configuration, you might need to make alterations or enhancements. The tasks listed here are the most common things you might want to do:

- “Setting up a publish/subscribe network”
- “Coordinated transactions” on page 39
- “Deleting components from the broker domain” on page 40
- “Importing and exporting message sets” on page 41

Setting up a publish/subscribe network

If your applications exploit publish/subscribe services, you must consider whether you want to implement topic-based security. This feature is optional, but you are recommended to use it to control access to publications generated within your MQSeries Integrator network. You must ensure that all messages have an associated topic to take advantage of topic-based security. See the *MQSeries Integrator Introduction and Planning* book for more detailed information about the benefits of topic-based security.

If you decide you want to implement topic-based security, you must complete the following tasks:

Publish/subscribe network

- “Adding a User Name Server to your broker domain”
- “Setting up collectives”
- “Defining ACLs” on page 39

Adding a User Name Server to your broker domain

If you decide that you require topic-based security in your broker domain when you first define it, you will find the tasks you must complete simpler than if you add it later. However, it is possible to add it later: this is illustrated in the *MQSeries Integrator Installation Guide*.

- Step 1. You must stop your Configuration Manager and every broker in your broker domain. Use the **mqsistop** command.
- Step 2. You must create your User Name Server using the **mqsicreateusername** command or the Command Assistant. See “Creating the MQSeries Integrator User Name Server” on page 21 for more details.
When you create the User Name Server, you must specify the security domain from which all principals participating in publish/subscribe will be gathered. You are strongly recommended to specify the same security domain as the one you specified when you created the Configuration Manager. For more information about security and domains, see “Chapter 4. Setting up security” on page 51.
- Step 3. You must create additional MQSeries resources that are required to connect the User Name Server and its queue manager to all other queue managers that support components in the broker domain. See “Connecting two MQSeries Integrator components” on page 22 for more details.
- Step 4. You must modify the Configuration Manager and all your brokers by invoking the appropriate **mqsichangexxxx** command to specify the User Name Server queue manager.
- Step 5. You must start the User Name Server, and restart the Configuration Manager and the brokers.
- Step 6. You can now implement topic-based security through the Control Center.

Setting up collectives

You can connect brokers together to form collectives to improve performance and network traffic. Each broker can belong to at most one collective. See the *MQSeries Integrator Introduction and Planning* book for more detailed information about the benefits of collectives.

You must set up collectives by implementing two subtasks in the following order:

- Step 1. “Connecting two MQSeries Integrator components” on page 22
- Step 2. “Creating and populating collectives in the Control Center”

Creating and populating collectives in the Control Center: When you have completed the MQSeries connections between the brokers that are to be defined to a collective, following the guidance given in “Connecting two MQSeries Integrator components” on page 22, you can define the collective to the topology.

- Step 1. Select the *Topology* view in the Control Center.
- Step 2. Right-click the topology and select the option to create a collective. You must enter the name of the collective in the properties dialog that appears. Close the dialog.
- Step 3. The collective now appears in the right pane. You can now assign (drag and drop) the brokers to the new collective.
- Step 4. Check in and deploy the updated topology.

Defining ACLs

You must create ACLs using the Control Center. Detailed instruction for this task are in the online help for the Control Center. In summary:

- Step 1. Select the *Topics* view of the Control Center.
- Step 2. Select the topics you want to control.
- Step 3. Select the option you want to control (publication, subscription, or persistent delivery).
- Step 4. Select the principals you want to authorize for this option.
- Step 5. Check in and deploy the changes you have made.

Coordinated transactions

To understand how to configure MQSeries as a transaction manager, you should refer to the information on database coordination in the *MQSeries System Administration* book. MQSeries Integrator Version 2.0.2 fully supports DB2 on all supported platforms, and Oracle on Sun Solaris as underlying SQL databases in coordinated transactions.

Note: If you configure a message flow as partially globally coordinated (that is, you have set the message flow property Coordinated Transaction to yes, but a database access node within that message flow has specified the Transaction property as commit), any database access node that causes an immediate commit must logically precede any database nodes that are part of the global transaction.

If a commit is attempted after a global transaction has been started, a failure is detected by MQSeries acting as the transaction manager.

Using DB2 in coordinated transactions

If you want to use DB2 in coordinated transactions, you must follow the instructions in the section called "DB2 configuration" in the *MQSeries System Administration* book.

To enable a message flow to act as a coordinated transaction, set the 'Coordinated transaction' property to 'yes' in the assigned message flow. The default is 'no'. For further details, see "Setting the properties of an assigned message flow" in the "Assigning resources to a broker" chapter of the *MQSeries Integrator Using the Control Center* book.

You must define to the broker queue manager, an XAResourceManager stanza for each of your DB2 databases that will participate in a coordinated transaction message flow. Additionally, if your message flows reference message formats in the message repository or contain Publication nodes, you must also define an XAResourceManager stanza for the broker database.

Fully globally coordinated message flows that involve a DB2 resource manager are supported using DB2 Version 6.1 or later.

Using Oracle in coordinated transactions (Sun Solaris only)

If you want to use Oracle in coordinated transactions, you must follow the instructions in the section called "Oracle configuration" in the *MQSeries System Administration* book. In particular, you must provide a symbolic link to the Oracle client library: `libc1ntsh` from `/usr/lib`.

To enable a message flow to act as a coordinated transaction, set the 'Coordinated transaction' property to 'yes' in the assigned message flow. (The default is 'no'.)

MQSeries and transactions

For further details, see “Setting the properties of an assigned message flow” in the “Assigning resources to a broker” chapter of the *MQSeries Integrator Using the Control Center* book.

You must define to the broker queue manager, an XAResourceManager stanza for each of your Oracle systems that will participate in a coordinated transaction message flow. Additionally, if your message flows reference message formats in the message repository or contain Publication nodes, you must also define an XAResourceManager stanza for the broker database. You do not need to build your own switchloadfile, MQSeries Integrator provides one for you, so when you are adding the XAResourceManager configuration information for Oracle, specify /opt/mqsi/merant/lib/UKor8dtc16.so as the SwitchFile. The following is an example of what is needed in the qm.ini file XAResourceManager stanza:

```
XAResourceManager:  
  Name=Oracle8  
  SwitchFile=/opt/mqsi/merant/lib/UKor8dtc16.so  
  XAOpenString=ORACLE_XA+SQLNET=myserver+ACC=P/scott/tiger+sestm=0+threads=TRUE  
  XACloseString=  
  ThreadOfControl=PROCESS
```

Fully globally coordinated message flows that involve an Oracle resource manager are supported using Oracle Versions 8.1.6, and 8.1.7.

Deleting components from the broker domain

You might need to change the configuration of your broker domain by deleting brokers, or by removing topic-based security.

Deleting a broker

When you create a broker, you have to take two actions to complete the task:

- Creating the actual broker on the target machine using the **mqsicreatebroker** command or the Command Assistant.
- Creating it in the configuration repository and deploying it in the broker domain topology, using the Control Center.

When you delete a broker, you must perform tasks to reverse the actions you took to create it. The tasks depend on whether you deployed the broker in the topology.

- If you created the physical broker, but never created the reference in the configuration repository, you must complete just one task to delete the broker:
 - Step 1. Stop the broker using the **mqsisistop** command.
 - Step 2. Delete the physical broker using the **mqsideletebroker** command or the Command Assistant.
- If you have created the physical broker and the reference in the configuration repository, but never deployed the broker, you must:
 - Step 1. Stop the broker using the **mqsisistop** command.
 - Step 2. Delete the broker using the *Topology* view of the Control Center. This removes all trace of the broker from the configuration repository immediately, including any contained execution groups.
 - Step 3. Delete the physical broker using the **mqsideletebroker** command or the Command Assistant.
- If you deployed the broker, you must:
 - Step 1. Delete the broker using the *Topology* view of the Control Center. The delete operation does not immediately remove the broker from the shared configuration, but the broker is marked as logically deleted and

Deleting components

no longer appears in the *Topology* or *Assignments* views of the Control Center, although it still exists in the shared configuration.

This is because the actual broker still contains configuration data that has previously been deployed to it. It is not until this data has been removed from the broker itself that the Configuration Manager is able to remove all trace of the broker from the configuration repository.

- Step 2. Deploy the change to the topology by right-clicking the *Topology* root and selecting *Deploy* → *Delta configuration (all types)*. This removes all the deployed configuration data from the broker. If it is successful (and you are advised to check the *Log* view to check that it is), the Configuration Manager removes all trace of the broker from the configuration repository.
- Step 3. Stop the broker using the **mqsistop** command.
- Step 4. You can now delete the broker using the command **mqsdeletebroker** or the Command Assistant.

If you are deleting a deployed broker, and you delete the physical broker before deleting the reference in the configuration repository, the Configuration Manager detects that the broker is not present when it deploys the updated topology. The deploy action removes all trace of the logically deleted broker from the configuration repository.

A warning message is generated and displayed in the *Log* view to inform you that the physical broker has already been deleted.

Removing topic-based security

Topic-based security is established by the presence of the User Name Server in your broker domain. If you want to remove this function, you must remove the User Name Server and the reference made to it by your Configuration Manager and all your brokers.

To remove topic-based security:

- Step 1. Stop the major components of your broker domain by issuing **mqsistop** for all brokers, the Configuration Manager, and the User Name Server.
- Step 2. Modify the properties of the Configuration Manager and all the brokers to remove the reference to the User Name Server. You can use the **mqsichangexxxx** commands or the Command Assistant to do this. For example: `mqsichangeconfigmgr -s ""`
- Step 3. Delete the User Name Server using the **mqsdeleteusernameserver** command or the Command Assistant.
- Step 4. Restart the Configuration Manager and all your brokers.
- Step 5. You can, but do not have to, remove all ACLs from the Control Center (from the *Topics* view). If you remove the ACLs, you must deploy your change to ensure the brokers are notified of the update. If the User Name Server does not exist, they are no longer applicable, but are simply not referenced.

Importing and exporting message sets

You can export message sets from the message repository and recreate them in an additional or replacement message repository by importing them. This allows you to reuse message sets in different domains, or to save and restore your message sets if you need to recreate your database, or your Configuration Manager, or both, for any reason.

Message sets

- To export a message set:
 - Step 1. Invoke the **mqsiiimpexpmsgset** command, specifying the **-e** flag, the name of the message set you want to export (you can export one message set on each command), and the name of the file in which the message set is to be stored. See “mqsiiimpexpmsgset (Import/Export message set)” on page 129 for details of this and other information required by the command.

You must invoke this command on the system on which the Configuration Manager exists.
 - Step 2. When the command completes, the definition of the message set has been written in XML. Back up the message set file.
 - Step 3. Copy the message set file to the system on which the additional or replacement Configuration Manager is created.
- To import a message set:
 - Step 1. Invoke the **mqsiiimpexpmsgset** command, specifying the **-i** flag, and the name of the file that contains the definition of the message set to be imported. See “mqsiiimpexpmsgset (Import/Export message set)” on page 129 for details of this and other information required by the command.

You must invoke this command on the system on which the Configuration Manager exists.
 - Step 2. When the command completes, the message definition has been stored in the message repository.
 - Step 3. You must stop and restart the Configuration Manager and the Control Center after the import has completed to be able to access the imported message set.
 - Step 4. Select the *Message Sets* view in the Control Center to add and view the new message set.

Recovery and restart

This section describes the actions you must take if you need to recover from errors, and restart some or all of the components of your broker domain. It covers the following topics:

- “Making sure that messages aren’t lost”
- “Making sure that subscriptions aren’t lost” on page 45
- “Restart scenarios” on page 45
- “Backup and recovery” on page 47

Making sure that messages aren’t lost

It is important to safeguard messages flowing through your broker domain, both application-generated messages and those used internally for inter-component communication. MQSeries provides two techniques that protect against message loss:

- Message persistence
If a message is persistent, MQSeries ensures it is not lost when a failure occurs, by hardening it to disk.
- Syncpoint control
An application can request that a message is processed in an atomic manner in a synchronized unit-of-work (UOW)

For more information about how to use these options, refer to *MQSeries System Administration*.

Broker internal messages: MQSeries Integrator components use MQSeries messages to communicate events and data between broker processes and subsystems. The broker ensures that the MQSeries features listed are exploited to protect against message loss. Therefore, you do not need to take any additional steps to configure MQSeries or MQSeries Integrator to protect against loss of internal messages.

Application messages: If delivery of application messages is critical, you must design application programs and the message flows they use to ensure that messages are not lost.

- Using persistent messages

The default action of a message flow is to respect the persistence of each incoming message. The client program must therefore specify the required message persistence when it puts the message to the input queue of a message flow.

If a message is not modified by the message flow, and is subsequently written to an output queue by an MQOutput node, the message is put with its persistence unchanged.

If a message passes through a publication node, the persistence of messages sent to subscribers is determined by the subscribers' registration options. The default behavior is that the persistence of the original message is maintained. The subscriber can override this behavior to request that the message is sent persistently or non-persistently regardless of the original persistence of the message.

If a new message is created by a message flow (for example, in a Compute node), the persistence in the MQMD of the new message is copied from the persistence in the MQMD of the incoming message.

You should note that SCADA does not have a concept of queues, nor of normal persistence. Clients connect to a SCADA node by specifying the port number that the node is listening on. Messages are sent to clients using a `clientId`. Within a SCADA subscription message, you can specify a maximum QoS (Quality of Service), which is similar to persistence:

- **QoS0** Non-persistent.
- **QoS1** Persistent, but might be delivered more than once.
- **QoS2** Once and once only delivery.

If a persistent SCADA message is published, it might be downgraded to the highest level that the client can accept. In some circumstances this might mean that the message becomes non-persistent.

- Processing messages under syncpoint control

The default action of a message flow is to process incoming messages under syncpoint in a broker-controlled transaction. This means that a message that fails to be processed for any reason is backed out by the broker. Because it was received under syncpoint, the failing message is reinstated on the input queue and can be processed again. Backout processing (described in "Backout Processing" on page 44) might be invoked.

You can define a message flow to handle exceptions or errors by using processing logic within the flow. Exceptions can be handled in two ways:

1. The failure terminal of a node can be connected to another part of the message flow. If an error occurs when a message is processed by that node, the message is propagated through the failure terminal to the next node in the sequence.

Recovery and restart

2. A TryCatch node can be incorporated in the message flow to catch all exceptions that are generated by subsequent nodes in the flow, propagating the message to the catch terminal.

If the TryCatch node's catch terminal is not connected, the message *will be lost* and the transaction committed. You should note that this behavior is true for all disconnected terminals, with the exception of failure terminals.

Warning

If you handle exceptions in either of these two ways, the message flow becomes responsible for handling the exception condition. The default broker exception handling mechanism is not invoked unless the message flow throws an exception explicitly.

If an error occurs that is not expected, which might cause termination of the message flow and hence a backout of the message, an exception must be thrown by the message flow. You can use the Throw node to achieve this function.

Backout Processing: If your message flow processes messages in a UOW, an error in your message flow causes the UOW to be backed out. If the input message has been read under syncpoint, the message is reinstated on the input queue, and is therefore processed again.

If the error condition persists, the message continues to be passed through the message flow and backed out, causing a processing loop. This is repeated until the value of the MQMD *BackoutCount* equals or exceeds the value of the backout threshold for the input queue (attribute *BOTHRESH*). The *BackoutCount* is incremented automatically by MQSeries every time a message is backed out.

Note: The *BOTHRESH* value of a MQSeries queue is set, by default, to '0'. MQSeries Integrator takes this to represent a threshold of '1' so that the default message flow action is to back out a message once and then to invoke the backout processing described below. *BOTHRESH* values of '0' and '1' are therefore equivalent in MQSeries Integrator.

MQSeries Integrator invokes backout processing by attempting to propagate the message as follows:

1. To the failure terminal of the current node.
2. To the queue specified as the input queue's backout requeue name (queue attribute *BOQNAME*).
3. To the queue manager's dead-letter-queue (DLQ).

If none of these queues exist, the message cannot be handled safely without risk of loss. The message cannot be discarded, therefore the message flow continues to attempt to backout the message. It records the error situation by writing errors to the local error log. A second indication of this error is the continual incrementing of the *BackoutCount* of the first message in the input queue.

To correct this situation, you can define one of the backout queues mentioned above. If the condition preventing the message from being processed has cleared, you can temporarily increase the value of the *BOTHRESH* attribute. This forces the message through normal processing.

You should note that SCADA does not have a concept of queues, nor of a backout threshold. If the failure terminal of a SCADA node is not connected, an error condition causes the message to be continually processed until it succeeds.

Making sure that subscriptions aren't lost

Publish/subscribe applications put subscription registration messages to the queue `SYSTEM.BROKER.CONTROL.QUEUE`. If this queue becomes full or damaged, or contains an invalid message, processing of subscription registrations is affected.

You must ensure that applications do not put invalid messages to this queue. If the broker detects an invalid message, it rejects the message and generates an Event log entry.

Every time the broker receives a subscription registration, it stores it in its database. The broker can therefore retrieve all current subscriptions if it is stopped and restarted.

Subscription commands that are in progress when the broker terminates might be lost. You can remove this exposure by ensuring that applications send all subscription registration messages as persistent messages.

Managing unused subscriptions: Because subscriptions are stored in the broker's database, you are recommended to ensure applications deregister their subscriptions when they no longer require publications. If obsolete subscriptions are deregistered, the database tables do not build up unused subscriptions.

You can use the following techniques to manage the build-up of unused subscriptions:

- Ensure that your applications register subscriptions with a limited expiry period. By default subscriptions have unlimited expiry. Subscriptions with an expiry period are automatically cleaned up by the broker and are therefore removed from the database. If subscriptions need to be active for longer periods, your applications must periodically re-register their subscriptions.
- If your applications are not long-lived, or are subject to unexpected termination, you are recommended to specify a temporary dynamic queue on the broker's queue manager to receive publications in those applications. If you do this, the temporary dynamic queue becomes invalid when the application ends, and the broker automatically discards the subscription.
- You can view and manage the current subscriptions from the *Subscriptions* view of the Control Center. If this view shows subscriptions that are no longer required, you can delete them immediately on this view.

Restart scenarios

This section illustrates the actions you must take to restart the run-time components of MQSeries Integrator and other software on which they are dependent.

Broker: If you need to restart a broker and its environment, you must take the following actions in this order:

1. Stop the broker using the **mqsistop** command.
2. Stop the broker's queue manager using the **endmqm** command.
3. Stop the database manager. Refer to the documentation for your database for instructions on how to complete this task.
4. When everything has stopped, components must be restarted in the following order:

Recovery and restart

- a. Start the database manager. Refer to the documentation for your database for instructions on how to complete this task.
- b. Start the broker using the **mqsistart** command. This automatically restarts the queue manager.

The broker does not tolerate abnormal or out of sequence termination of the MQSeries queue manager or the database manager. If this occurs, the broker must be stopped using the **mqsistop** command, and all components restarted in the order listed.

If the problem is caused by the queue manager stopping, reissue the MQSeries **endmqm** command specifying the immediate option (-i) before issuing **mqsistop**.

You do not have to restart the broker execution group processes if they terminate abnormally, because the broker does this automatically.

Configuration Manager: The Configuration Manager operates independently of the brokers, and can be stopped and restarted without affecting the operation of other MQSeries Integrator components in the broker domain. If you need to restart the Configuration Manager and its environment, you must take the following actions:

1. Stop the Control Center. (This is not absolutely necessary, but it will make the following tasks much quicker.)
2. Stop the Configuration Manager using the **mqsistop** command.
3. Stop the MQSeries queue manager using the **endmqm** command.
4. Stop DB2. Refer to the DB2 documentation for instructions on how to complete this task.

You are recommended to complete these tasks in the order shown, but the Configuration Manager tolerates the queue manager and DB2 stopping first.

When everything has been stopped, you are recommended to take the following actions in this order:

1. Restart DB2. Refer to the DB2 documentation for instructions on how to complete this task.
2. Restart the Configuration Manager using the **mqsistart** command.
This automatically restarts the queue manager.

If you restart the Configuration Manager first, it will automatically retry its initialization until DB2 is started.

The Configuration Manager also tolerates abnormal termination of the MQSeries queue manager and DB2: if the Configuration Manager detects that either has terminated abnormally, it restarts automatically. If either DB2, or MQSeries, or both, are not running, initialization is automatically retried every 30 seconds until it is successful.

User Name Server: The User Name Server operates independently of the brokers, and can be stopped and restarted without affecting the operation of other MQSeries Integrator components in the broker domain. If you need to restart the User Name Server and its environment, you must take following actions:

1. Stop the User Name Server using the **mqsistop** command.
2. Stop the MQSeries queue manager using the **endmqm** command.

Recovery and restart

You are recommended to complete these tasks in the order shown, but the User Name Server tolerates the queue manager stopping first.

When everything has been stopped, you can restart the User Name Server using the **mqsistart** command. This automatically restarts the queue manager.

The User Name Server also tolerates abnormal termination of the MQSeries queue manager. If this occurs the User Name Server restarts automatically. If MQSeries is not up and running, initialization is retried every 30 seconds until it is successful.

Backup and recovery

Brokers and the Configuration Manager rely on a database manager to maintain and control all their configuration data. Brokers, the Configuration Manager and the User Name Server rely on MQSeries to transport and guarantee messages between components. You must establish a backup process that includes these sources of information to preserve the integrity and consistency of your broker domain.

- You must back up database tables. You are strongly recommended to perform backup frequently and on a regular basis to prevent loss of configuration data if damage occurs.

You must backup all configuration repository tables, and all message repository tables, and the following broker database tables:

- BSUBSCRIPTIONS
- BCLIENTUSER
- BUSERCONTEXT
- BRETAINEDPUBS
- BPUBLISHERS
- BMQPSTOPOLOGY

The broker's configuration is completed when it is redeployed by the Configuration Manager when the broker domain is restarted. This recreates the remaining database tables.

The configuration repository and message repository tables contain associated data: you must coordinate their backup procedures to ensure that a consistent image is available if recovery is necessary. The complete set of tables that make up these two repositories is detailed in Appendix A "System changes after installation" in the *MQSeries Integrator Installation Guide* for your computer platform.

If your message flows use any user-defined database tables, you must also back these up.

Refer to the appropriate database documentation for details of backup procedures.

- You must back up MQSeries configuration data. Refer to *MQSeries System Administration* for further details.
- You must also consider backing up files that are created by users of the Control Center. When a Control Center user saves a workspace, the workspace XML document, any newly created objects, and any checked out objects are saved to the local file system. You are recommended to instruct users to backup this data by exporting the workspace and backing up the XML documents created (in a file) by the export. You should similarly export and backup message sets stored in the message repository as described in "Importing and exporting message sets" on page 41.

Recovery and restart

Recovery scenarios

You can recover the run-time components of MQSeries Integrator if the environment becomes damaged (for example, if MQSeries objects used by the broker are damaged), or if database contents are damaged.

Broker: If the environment for a particular broker becomes damaged, or if one or more of the broker database tables are unusable, you must perform the following sequence of operations to recover it:

- Step 1. Ensure that no Control Center users are deploying to brokers. You must wait until these actions have completed.
- Step 2. Stop the broker using the **mqsistop** command.
- Step 3. Stop the broker's queue manager using the **endmqm** command.
- Step 4. If there is no damage to any one of broker database tables listed in "Backup and recovery" on page 47, take a backup of these tables. (These tables are interdependent and must all be in a consistent state when restored. You cannot backup or restore individual tables.)
- Step 5. Delete the broker using the **mqsdeletebroker** command or the Command Assistant.
- Step 6. Recreate the broker using the **mqscreatebroker** command or the Command Assistant.
- Step 7. Restore the broker database tables listed in "Backup and recovery" on page 47, either from the backup you have just taken, or from a previously successful backup version.
- Step 8. Start the broker using the **mqsistart** command.
- Step 9. Restart the Control Center if it is not currently running. Select the *Topology* view.
- Step 10. Redeploy the domain configuration by selecting *File* → *Deploy* → *Complete configuration (all types)* → *Normal* to ensure that the configuration across the broker domain is consistent.

Configuration Manager: If the Configuration Manager environment is damaged, or one or more of the database tables are corrupted, you must perform the following sequence of operations to recover it:

- Step 1. Ensure that all Control Center sessions are stopped.
- Step 2. Stop the Configuration Manager using the **mqsistop** command.
- Step 3. Stop the Configuration Manager's queue manager using the **endmqm** command.
- Step 4. Delete the Configuration Manager using the **mqsdeleteconfigmgr** command or the Command Assistant:
 - a. If you are recovering the Configuration Manager because one or more of the configuration repository or message repository tables is damaged, you must **include** the flags **-n -m** on the **mqsdeleteconfigmgr** command. As this will destroy all information pertinent to the broker domain and not just data internal to the Configuration Manager first, ensure that you have exported everything you can from the domain.
 - b. If the database tables are undamaged, you must **omit** the flags **-n -m**. This preserves your configuration data in both repositories.
- Step 5. If you are recovering the Configuration Manager because one or more of the configuration repository or message repository tables is damaged, you must restore both repositories from a previously successful backup

version. (The data in the two repositories is interdependent, and you must restore the entire contents of both. You cannot restore individual tables.)

- Step 6. Recreate the Configuration Manager using the **mqsicreateconfigmgr** command or the Command Assistant.
- Step 7. Start the Configuration Manager using the **mqsistart** command.
- Step 8. Start the Control Center, and select the *Topology* view.
- Step 9. If you have completed Steps 4a on page 48 and 5 on page 48, you must also redeploy the domain configuration by selecting *File* → *Deploy* → *Complete configuration (all types)* → *Normal* to ensure that the configuration across the broker domain is consistent.

User Name Server: If the User Name Server environment becomes damaged, you must perform the following sequence of operations to recover it:

- Step 1. Stop the User Name Server using the **mqsistop** command.
- Step 2. Stop the User Name Server's queue manager using the **endmqm** command.
- Step 3. Delete the User Name Server using the **mqsdeleteusernameserver** command or the Command Assistant.
- Step 4. Recreate the User Name Server using the **mqscreateusernameserver** command or the Command Assistant.
- Step 5. Start the User Name Server using the **mqsistart** command.

Managing workload and performance

You can make some changes to your broker domain configuration to influence the performance and workload. The following topics are covered:

- "Using MQSeries trusted applications"
- "Tuning message flow performance" on page 50

Using MQSeries trusted applications

You can configure the message broker as an "MQSeries trusted (fastpath) application" by selecting the **-t** (fastpath) option on the **mqsicreatebroker** command. This causes the broker and the MQSeries queue manager agent to run in the same process, thus improving overall system performance. By default the broker does not run as a trusted application.

For example, enter the following command to create broker **MQSI_SAMPLE_BROKER** as a trusted application:

```
mqsicreatebroker MQSI_SAMPLE_BROKER -i  
mqsiuid -a mqsipw -q MQSI_SAMPLE_QM -s MQSI_SAMPLE_UNQ_QM -n MQSIBRDB  
-t
```

MQSeries trusted applications must run with an effective user ID and group ID of **mqm**. On UNIX platforms, you must therefore specify the user ID **mqm** as the service user ID when you create the broker. On Windows NT, any service user ID which is a member of the **mqm** group can be used.

You can change the option you selected when you created the broker using the **mqsichangebroker** command. You must stop the broker using the **mqsistop** command before you change its properties. When you have made the change, you can restart the broker using the **mqsistart** command. You might also have to change the service user ID (and password) if you did not originally create the broker to use an appropriate service user ID.

Managing workload and performance

For further details on the implications of running the broker as a trusted MQSeries application, see *MQSeries Integrator Introduction and Planning*.

The Configuration Manager always runs as an MQSeries fastpath application for performance reasons. You cannot change this setting.

Tuning message flow performance

When you have assigned a message flow to a broker, you can modify its default properties to improve its throughput.

The following properties control the frequency with which the message flow commits transactions:

- **Commit Count.** This represents the number of messages processed from the input queue before an MQCMIT is issued.
- **Commit Interval.** This represents the time interval that will elapse before an MQCMIT is invoked.

You can also increase the throughput of a message flow, and therefore the efficiency of message processing, by updating the **Additional Instances** property of the MQInput node. This instructs the broker to start additional threads to read messages from the input queue and process them concurrently. You must consider the impact on message order if you change this property: because several threads can be processing messages concurrently there is no guarantee that message processing will complete in the same order, so that messages might be received by clients in a different order.

If you increase this message flow property you must ensure that the input queue has been defined with the share attribute which enables multiple threads to read from the same queue.

You might also want to consider changing the **Order Mode** property of an assigned message flow. For more information about additional instances and message order, see Chapter 4, “Message Flows” in *MQSeries Integrator Introduction and Planning*.

You should note that none of the above options are supported by either the MQSeries Everyplace or SCADA nodes. Commits are performed on a ‘per-message’ basis.

For more details of these properties and how you can change their values, see *MQSeries Integrator Using the Control Center* and the Control Center online help.

Chapter 4. Setting up security

This chapter describes the actions you must take to ensure security of system administration tasks. You must grant specific users the authority to complete certain tasks, before you take the steps that achieve those tasks.

The information in this chapter builds on the planning information provided in *MQSeries Integrator Introduction and Planning*, which you should read for a fuller understanding of security, which options you must implement, and which you can choose to implement.

Tasks in the following areas are discussed:

- “Securing MQSeries Integrator resources”.
- “Securing MQSeries resources” on page 62.
- “Securing database resources” on page 63.

Securing MQSeries Integrator resources

MQSeries Integrator employs a variety of mechanisms to secure its different components and these vary between operating systems, as indicated:

Queues and queue managers

MQSeries access control.

- See “Securing MQSeries resources” on page 62 and *MQSeries System Administration*(SC33-1873) for further information.

Topic-based security

MQSeries Integrator User Name Server. (Each MQSeries Integrator domain can contain one User Name Server.)

- If you choose to run the User Name Server on a Windows NT node, it will map to the Windows systems security domain architecture
- If you choose to run the User Name Server on a UNIX node, it will map to the UNIX user/group database.
- See “Topic based security” in *MQSeries Integrator Introduction and Planning* for further information.

Operational control of brokers and other components

Operating system access control.

- On Windows NT, this is performed by the Windows Security Domains.
- On UNIX platforms, this is the UNIX user/group database.
- See Table 2 on page 54 and Table 3 on page 62 for a summary of authorizations.

Roles in Control Center

Operating system access control.

- On Windows NT, this is performed by the Windows Security Domains.
- This is not applicable on UNIX platforms; the Control Center will only operate on Windows NT.
- See “Control Center tasks” in *MQSeries Integrator Using the Control Center* for further information.

MQSeries Integrator resources

Depending on the task you want to perform, membership of some or all of the groups listed below is required to implement security control of MQSeries Integrator components, resources and tasks.

Users and groups (principals) are defined in the security subsystem of the operating system (through the User Manager on Windows NT, or the users/groups database on UNIX platforms). MQSeries Integrator always creates a set of local groups on the system on which it is installed.

These local groups are:

- **mqrbrks**
- **mqrbrasn**
- **mqrbrdevt**
- **mqrbrps**
- **mqrbrtpic**

For a description of each group, see "Security and Principals" in the *MQSeries Integrator Introduction and Planning* book.

MQSeries Integrator security architecture is designed to be platform independent. If you are running MQSeries Integrator in an environment that includes clients on heterogeneous platforms, you should ensure that all the principals you define for MQSeries Integrator task authorizations are limited to eight bytes or less. If you have a Windows NT homogeneous environment, you can create principals of up to twelve bytes (the limit set by the user identifier field in the MQSeries MQMD, which MQSeries Integrator uses) but you should only use these longer names if you are sure there will never be UNIX systems in your MQSeries Integrator domain.

On Windows NT, MQSeries Integrator draws principals from either the Windows NT local account security domain, or a Windows NT primary domains, or a Windows NT trusted domain. Principals must be defined to a specific Windows NT security domain. You must decide which domain you want to use for MQSeries Integrator, and define your principals to that domain (using the Windows NT User Manager on the security domain server). If you already have a security domain set up to control access to MQSeries resources, you are advised to use this same domain for MQSeries Integrator: this will not cause any conflict and will ease your security administration.

If, on Windows NT, you create a Configuration Manager (**mqscreateconfigmgr**) or a User Name Server (**mqscreateusername server**), specifying a SecurityDomainName that is your local account security domain (that is, you want to draw users and groups from your machine's local user and group repository rather than from a primary or trusted domain), you should not specify a ServiceUserID that has the same name as your machine. For example, if you are working on a machine named \\BROKER1, (account domain BROKER1), you should not specify a user "broker1" as ServiceUserID. If you attempt to start the Configuration Manager or User Name Server with the same machine name and ServiceUserID, the Windows NT operating system call will incorrectly retrieve the Security Identifier (SID) of the machine rather than that of the account and you will receive the message: BIP8026 Unable to start the component.

Use the **-d** option on both the **mqscreateusername server** and **mqscreateconfigmgr** commands to specify the domain that principals are drawn from.

Although not a requirement, you are recommended to specify the same security subsystem for both the Configuration Manager and the User Name Server.

On UNIX platforms, MQSeries Integrator draws principals from the operating system's user and group tables.

You must assign users (or other groups) to the MQSeries Integrator groups in your security domain to allow them to perform specific tasks. The authorizations required are summarized in Table 2 on page 54 for Windows NT and in Table 3 on page 62 for UNIX platforms.

For a more general discussion on security for MQSeries Integrator see *MQSeries Integrator Introduction and Planning*. For further details about Windows NT security domains, see the information on the Microsoft Web site at

<http://www.microsoft.com/ntserver/security/deployment/default.asp>

In particular, you are advised to review the contents of the *Security Deployment Resources Roadmap* on this Web page.

Using Windows NT primary or trusted security domains

If you are using a primary or trusted security domain, you must define global groups to your primary or trusted security domain that mirror the local groups that are created during installation. MQSeries Integrator requires five global groups:

- **Domain mqbrkrs**
- **Domain mqbrasgn**
- **Domain mqbrdevt**
- **Domain mqbroops**
- **Domain mqbrtpic**

These groups must be made members of the local security domain's equivalent MQSeries Integrator groups (**Domain mqbrkrs** must be a member of **mqbrkrs**, and so on).

- If you install MQSeries Integrator on the domain controller of a primary or a trusted security domain, the MQSeries Integrator installation program creates the local and global groups, and adds the global groups to the local groups.

If you do not intend to install MQSeries Integrator on the domain controller, you can create these groups yourself using the Windows NT User Manager.

These groups should be defined exactly as shown in the above list. In the Windows NT User Manager, select *Groups* → *New Global Group* and enter **Domain mqbrasgn** in the *Group Name* field, and so on for the other groups.

- If you install MQSeries Integrator on a workstation member of a primary security domain, the MQSeries Integrator installation program creates the local groups. If the global groups already exist in the primary security domain, it also adds each global group to the appropriate local group in the local domain.

If you intend to install MQSeries Integrator on a primary domain controller, you are recommended to perform the installation on the domain controller first so that the domain groups are created for you. They will then be automatically added to the local groups created during a subsequent installation on a workstation.

- If you install MQSeries Integrator on a workstation member of a trusted domain, MQSeries Integrator cannot recognize the trusted domain, and does not add the global groups to the local groups. You must do this step yourself.

MQSeries Integrator resources

- If you install MQSeries Integrator on a workstation that is a member of both a trusted security domain and a primary security domain, the installation program creates the local groups. If the global groups already exist in the primary security domain, it also adds each global group to the appropriate local group in the local domain. It cannot detect the trusted domain and therefore does not add the global groups of the trusted security domain to the local groups. If you want these trusted security domain global groups in the local groups instead of, or in addition to, the primary security global groups, you must make these updates yourself.

When you define a new user ID to your security domain, you must assign this ID to the domain group that it authorized for the tasks this user ID is to perform, so that it is authorized globally.

Table 2 summarizes the security requirements for the MQSeries Integrator administrative tasks. It illustrates what group membership is required if you are using a local security domain defined on your local system **SALONE**, or a primary domain named **PRIMARY**, or a trusted domain named **TRUSTED**. The contents of this table assume that you have created both the Configuration Manager and the User Name Server with the same security domain (specified on flag -d).

For information about database authorization, see “Securing database resources” on page 63.

Table 2. Summary of administrative task authorization on Windows NT platforms

User is...	Local domain (SALONE)	Primary Domain (PRIMARY)	Trusted domain (TRUSTED)
Creating broker, Configuration Manager, User Name Server	<ul style="list-style-type: none"> • Must be a user ID defined in SALONE • Member of Administrators 	<ul style="list-style-type: none"> • Must be a user ID defined in PRIMARY • Member of SALONE\Administrators 	<ul style="list-style-type: none"> • Must be a user ID defined in TRUSTED • Member of SALONE\Administrators
Changing broker, Configuration Manager, User Name Server	<ul style="list-style-type: none"> • Must be a user ID defined in SALONE • Member of Administrators 	<ul style="list-style-type: none"> • Must be a user ID defined in PRIMARY • Member of SALONE\Administrators 	<ul style="list-style-type: none"> • Must be a user ID defined in TRUSTED • Member of SALONE\Administrators
Deleting broker, Configuration Manager, User Name Server	<ul style="list-style-type: none"> • Member of Administrators 	<ul style="list-style-type: none"> • Member of SALONE\Administrators 	<ul style="list-style-type: none"> • Member of SALONE\Administrators
Starting broker, Configuration Manager, User Name Server	<ul style="list-style-type: none"> • Member of Administrators 	Not applicable.	Not applicable.
Listing broker, Configuration Manager, User Name Server	<ul style="list-style-type: none"> • Must be a user ID defined in SALONE • Member of mqrbrks 	<ul style="list-style-type: none"> • Must be a user ID defined in PRIMARY • Member of PRIMARY\Domain mqrbrks 	<ul style="list-style-type: none"> • Must be a user ID defined in TRUSTED • Member of TRUSTED\Domain mqrbrks
Changing, displaying, retrieving trace information	<ul style="list-style-type: none"> • Must be a user ID defined in SALONE • Member of mqrbrks 	<ul style="list-style-type: none"> • Must be a user ID defined in PRIMARY • Member of PRIMARY\Domain mqrbrks 	<ul style="list-style-type: none"> • Must be a user ID defined in TRUSTED • Member of TRUSTED\Domain mqrbrks
Running User Name Server (service user ID)	<ul style="list-style-type: none"> • Must be a user ID defined in SALONE • Member of mqrbrks 	<ul style="list-style-type: none"> • Must be a user ID defined in PRIMARY • Member of PRIMARY\Domain mqrbrks 	<ul style="list-style-type: none"> • Must be a user ID defined in TRUSTED • Member of TRUSTED\Domain mqrbrks

Table 2. Summary of administrative task authorization on Windows NT platforms (continued)

User is...	Local domain (SALONE)	Primary Domain (PRIMARY)	Trusted domain (TRUSTED)
Running Configuration Manager (service user ID)	<ul style="list-style-type: none"> • Must be a user ID defined in SALONE • Member of mqbrkrs • Member of mqm 	<ul style="list-style-type: none"> • Must be a user ID defined in PRIMARY • Member of PRIMARY\Domain mqbrkrs • Member of SALONE\mqm (see note 1) 	<ul style="list-style-type: none"> • Must be a user ID defined in TRUSTED • Member of TRUSTED\Domain mqbrkrs • Member of SALONE\mqm (see note 2)
Running broker (MQSeries fastpath off) (service user ID)	<ul style="list-style-type: none"> • Must be a user ID defined in SALONE • Member of mqbrkrs 	<ul style="list-style-type: none"> • Must be a user ID defined in PRIMARY • Member of PRIMARY\Domain mqbrkrs 	<ul style="list-style-type: none"> • Must be a user ID defined in TRUSTED • Member of TRUSTED\Domain mqbrkrs
Running broker (MQSeries fastpath on) (service user ID)	<ul style="list-style-type: none"> • Must be a user ID defined in SALONE • Member of mqbrkrs • Member of mqm 	<ul style="list-style-type: none"> • Must be a user ID defined in PRIMARY • Member of PRIMARY\Domain mqbrkrs • Member of SALONE\mqm 	<ul style="list-style-type: none"> • Must be a user ID defined in TRUSTED • Member of TRUSTED\Domain mqbrkrs • Member of SALONE\mqm
Clearing, joining, listing MQSeries Publish/Subscribe brokers	<ul style="list-style-type: none"> • Must be a user ID defined in SALONE • Member of mqbrkrs 	<ul style="list-style-type: none"> • Must be a user ID defined in PRIMARY • Member of PRIMARY\Domain mqbrkrs 	<ul style="list-style-type: none"> • Must be a user ID defined in TRUSTED • Member of TRUSTED\Domain mqbrkrs
Running Control Center (see note 3)	<ul style="list-style-type: none"> • Must be a user ID defined in SALONE (see note 4) For example, SALONE\User1 is valid, PRIMARY\User2 and TRUSTED\User3 are not • Member of one or more of mqbrasgn, mqbrdevt, mqbrops, mqbrtpic 	<ul style="list-style-type: none"> • Must be a user ID defined in PRIMARY (see note 4) For example, PRIMARY\User2 is valid, SALONE\User1 and TRUSTED\User3 are not. • Member of one or more of PRIMARY\Domain mqbrasgn, PRIMARY\Domain mqbrdevt, and so on. 	<ul style="list-style-type: none"> • Must be a user ID defined in TRUSTED (see note 4) For example, TRUSTED\User3 is valid, SALONE\User1 and PRIMARY\User2 are not. • Member of one or more of TRUSTED\Domain mqbrasgn TRUSTED\Domain mqbrdevt, and so on.
Running publish/subscribe applications	<ul style="list-style-type: none"> • Must be a user ID defined in SALONE For example, SALONE\User1 is valid, PRIMARY\User2 and TRUSTED\User3 are not. 	<ul style="list-style-type: none"> • Must be a user ID defined in PRIMARY For example, PRIMARY\User2 is valid, SALONE\User1 and TRUSTED\User3 are not. 	<ul style="list-style-type: none"> • Must be a user ID defined in TRUSTED For example, TRUSTED\User3 is valid, SALONE\User1 and PRIMARY\User2 are not.
<p>Notes:</p> <ol style="list-style-type: none"> 1. If you are running in a primary domain, you can also: <ul style="list-style-type: none"> • Define the user ID in the domain PRIMARY. • Add this ID to the group PRIMARY\Domain mqm. • Add the PRIMARY\Domain mqm group to the group SALONE\mqm. 2. If you are running in a trusted domain, you can also: <ul style="list-style-type: none"> • Define the user ID in the domain TRUSTED. • Add this ID to the group TRUSTED\Domain mqm. • Add the TRUSTED\Domain mqm group to the group SALONE\mqm. 3. All Control Center users need read access to the MQSeries java\lib subdirectory of the MQSeries home directory (the default is X:\Program Files\MQSeries, where X: is the operating system disk). This access is restricted to users in the local group mqm by MQSeries. MQSeries Integrator installation overrides this restriction and gives read access for this subdirectory to all users. 4. If a valid user ID is defined in the domain used by the Configuration Manager, (for example, PRIMARY\User4) an identical user defined in a different domain (for example, DOMAIN2\User4) will be able to access the Control Center with the authorities of PRIMARY\User4. 			

The IBMMQSI2 superuser

A *superuser* user ID is recognized by the Configuration Manager. This user ID, IBMMQSI2, is a privileged user ID that provides these essential functions:

- It has the authority to unlock any resources locked to another user ID. If a user ID is removed for any reason (for example, if an employee leaves the company)

MQSeries Integrator resources

and resources are left locked to that user ID, you can start the Control Center with the privileged user ID and unlock the locked resources.

- The IBM primitive message processing nodes (described in *MQSeries Integrator Using the Control Center* and online help) are locked under this user ID. If maintenance that includes updates to these nodes is supplied by IBM, you must use this user ID to import the replacement nodes, check them in to the configuration repository, and re-lock them.

If you find that it is necessary to invoke any of these functions, you must define the user ID `IBMMQSI2` yourself (using the Windows NT User Manager) to the security domain specified when you create the Configuration Manager using the `mqsicreateconfigmgr` command. You must also add this user ID to the MQSeries Integrator groups necessary for it to be authorized to complete the task required on the system on which you are running the Control Center:

- If you are using a primary or trusted security domain, you must add this user ID to the appropriate **Domain mqbrxxx** groups.
- If you are using a local security domain, you must add this user ID to the appropriate local **mqbrxxx** groups.

Windows NT security domain scenarios

This section describes three possible scenarios that illustrate the use of primary, trusted, and local security domains. The domains identified in Table 2 on page 54 to define task authorizations are used in these scenarios.

Scenario 1: operation in a Windows NT primary domain

This scenario describes the security setup required if you are using a primary security domain. It makes the following assumptions:

1. The broker, User Name Server, and Configuration Manager are to be installed on separate machines `BROKER`, `UNS`, and `CFGMGR`.
2. The queue manager supporting the User Name Server is to be named `UNSQMGR`.
3. These three systems are configured into a primary security domain named `PRIMARY`.
4. Control Center users and application users will be defined in the `PRIMARY` domain.
5. Publish/subscribe access control for topic-based security is to be enabled.

Installation: During installation, the installation program performs the following configuration tasks:

1. If any of the systems on which you are installing MQSeries Integrator is the primary domain controller, the global groups **Domain mqbrkr**, **Domain mqbrdevt**, **Domain mqbrasgn**, **Domain mqbro**, and **Domain mqbrtopic** are created in the `PRIMARY` domain. If you install on the primary domain controller, you are recommended to complete this installation before you do any others.

If you are not installing on the primary domain controller, you must create these groups yourself using the Windows NT User Manager. You are strongly recommended to create these groups **before** you install any components on any system in the primary security domain. If you do, the installation program populates the local groups it creates with these global groups. If you do not, you must add the global groups to the local groups on every system on which you have installed MQSeries Integrator yourself.

2. When you install the broker component on system BROKER, a local **mqbrkrs** group is created in the BROKER account domain and **PRIMARY\Domain mqbrkrs** is added to **BROKER\mqbrkrs**.
3. When you install the User Name Server component on system UNS, a local **mqbrkrs** group is created in the UNS account domain and **PRIMARY\Domain mqbrkrs** is added to **UNS\mqbrkrs**.
4. When you install the Configuration Manager component on system CFGMGR:
 - a. A local **mqbrkrs** group is created in the CFGMGR account domain and **PRIMARY\Domain mqbrkrs** is added to **CFGMGR\mqbrkrs**.
 - b. The four remaining MQSeries Integrator local groups (**mqbrasgn**, **mqbrdevt**, **mqbrops**, and **mqbrtpic**) are created in the CFGMGR account domain, and the **PRIMARY\Domain mqbrxxxx** global groups are added to the equivalent **CFGMGR\mqbrxxxx** groups.

Service user IDs: When installation is complete, you must take the following actions on the PRIMARY domain's domain controller:

1. Define a user ID BRKSID for the broker's service user ID in PRIMARY. Add this ID to the group **PRIMARY\Domain mqbrkrs**.
2. Define a user ID UNSSID for the User Name Server's service user ID in PRIMARY. Add this ID to the group **PRIMARY\Domain mqbrkrs**.
3. Define a user ID CMSID for the Configuration Manager's service user ID in PRIMARY. Add this ID to the group **PRIMARY\Domain mqbrkrs**, and to the local **mqm** group.

Creating the MQSeries Integrator components: You can now create the components:

1. On the system UNS, issue the **mqsicreateusernameserver** command, specifying **-d PRIMARY, -i UNSSID, and -q UNSQMGR**.
2. On the system CFGMGR, issue the **mqsicreateconfigmgr** command, specifying **-d PRIMARY, -i CMSID, and -s UNSQMGR**.
3. On the system BROKER, issue the **mqsicreatebroker** command, specifying **-i PRIMARY\BRKSID and -s UNSQMGR**.

Note: If you use an unqualified form for the service user ID on the **-i** flag, for example **BRKSID**, the operating system searches for this user ID throughout the domain, starting with the local system. If you have a duplicate ID on both the local system and the domain, the local ID will be found first. To ensure that the domain ID is used, you should qualify the ID with the domain name, for example **PRIMARY\BRKSID**.

Note: When you invoke the **mqsicreatexxxx** commands, your user ID must be a member of the Windows NT **Administrators** group on the system where the command is being executed.

User Name Server run-time operation: When you start the User Name Server the following are true:

- The User Name Server Windows NT service runs under the user ID UNSSID.
- UNSSID is an indirect member of **UNS\mqbrkrs**, **CFGMGR\mqbrkrs** and **BROKER\mqbrkrs** (through **PRIMARY\Domain mqbrkrs**) and can therefore access appropriate MQSeries Integrator queues.
- Users and groups are sourced from the PRIMARY security domain.

Broker run-time operation: When you start the broker the following are true:

Security scenarios

- The broker Windows NT service runs under the user ID BRKSID.
- BRKSID is an indirect member of BROKER**mqbrkrs**, UNS**mqbrkrs**, and CFGMGR**mqbrkrs**, and can therefore access the appropriate MQSeries Integrator queues.
- Users and groups are supplied by the User Name Server.

Configuration Manager run-time operation: When you start the Configuration Manager the following are true:

- The Configuration Manager Windows NT service runs under the user ID CMSID.
- CMSID is an indirect member of BROKER**mqbrkrs**, UNS**mqbrkrs**, and CFGMGR**mqbrkrs**, and can therefore access the appropriate MQSeries Integrator queues.
- Control Center users are members of the PRIMARY domain whose roles are determined by membership of the PRIMARY global groups (PRIMARY**Domain mqbrdevt** and so on).
- The list of principals from which ACLs are created are supplied by the User Name Server.

Application clients: Clients running application programs must run under user IDs that are defined to the security domain PRIMARY.

Scenario 2: operation in a Windows NT trusted domain

This scenario describes the security setup required if you are using a trusted security domain. It makes the following assumptions:

1. The broker, User Name Server, and Configuration Manager are to be installed on separate machines BROKER, UNS, and CFGMGR.
2. These three systems are configured into the primary security domain PRIMARY.
3. Control Center users and application users will be defined in the TRUSTED domain.
4. Publish/subscribe access control for topic-based security is to be enabled.

Installation: During installation, the installation program performs the following configuration tasks:

1. If any of the systems on which you are installing MQSeries Integrator is the primary domain controller, the global groups **Domain mqbrkrs**, **Domain mqbrdevt**, **Domain mqbrasgn**, **Domain mqbrops**, and **Domain mqbrtopic** are created in the PRIMARY domain.

Note: These groups are created, although in this particular scenario they are not exploited.

2. When you install the broker component on system BROKER, a local **mqbrkrs** group is created in the BROKER account domain and PRIMARY**Domain mqbrkrs** is added to BROKER**mqbrkrs**.
3. When you install the User Name Server component on system UNS, a local **mqbrkrs** group is created in the UNS account domain and PRIMARY**Domain mqbrkrs** is added to UNS**mqbrkrs**.
4. When you install the Configuration Manager component on system CFGMGR:
 - a. A local **mqbrkrs** group is created in the CFGMGR account domain and PRIMARY**Domain mqbrkrs** is added to CFGMGR**mqbrkrs**.

- b. The four remaining MQSeries Integrator local groups (**mqbrasgn**, **mqbrdevt**, **mqbrotps**, and **mqbrtpic**) are created in the CFGMGR account domain, and the PRIMARY\Domain **mqbrxxxx** global groups are added to the equivalent CFGMGR**mqbrxxxx** groups.

Additional group configuration: After installation, you must perform the following additional configuration tasks:

1. Create global groups **Domain mqbrkrs**, **Domain mqbrdevt**, **Domain mqbrasgn**, **Domain mqbrotps**, and **Domain mqbrtpic** in the TRUSTED domain.
2. Add TRUSTED\Domain **mqbrkrs** to BROKER**mqbrkrs**.
3. Add TRUSTED\Domain **mqbrkrs** to UNS**mqbrkrs**.
4. Add TRUSTED\Domain **mqbrkrs** to CFGMGR **mqbrkrs**.
5. Add TRUSTED\Domain **mqbrasgn** to CFGMGR**mqbrasgn**, TRUSTED\Domain **mqbrdevt** to CFGMGR**mqbrdevt**, TRUSTED\Domain **mqbrotps** to CFGMGR**mqbrotps**, and TRUSTED\Domain **mqbrtpic** to CFGMGR**mqbrtpic**.

Service user IDs: When installation is complete, you must take the following actions on the TRUSTED domain's domain controller:

1. Define a user ID BRKSID for the broker's service user ID in TRUSTED. Add this ID to the group TRUSTED\Domain **mqbrkrs**.
2. Define a user ID UNSSID for the User Name Server's service user ID in TRUSTED. Add this ID to the group TRUSTED\Domain **mqbrkrs**.
3. Define a user ID CMSID for the Configuration Manager's service user ID in TRUSTED. Add this ID to the group TRUSTED\Domain **mqbrkrs**, and to the local **mqm** group.

Creating the MQSeries Integrator components: You can now create the components:

1. On the system UNS, issue the **mqsicreateusernameserver** command, specifying **-d TRUSTED**, **-i UNSSID**, and **-q UNSQMGR**.
2. On the system CGFMGR, issue the **mqsicreateconfigmgr** command, specifying **-d TRUSTED**, **-i CMSID**, and **-s UNSQMGR**.
3. On the system BROKER, issue the **mqsicreatebroker** command, specifying **-i TRUSTED\BRKSID** and **-s UNSQMGR**.

Note: If you use an unqualified form for the service user ID on the **-i** flag, for example BRKSID, the operating system searches for this user ID throughout the domain, starting with the local system. If you have a duplicate ID on both the local system and the domain, the local ID will be found first. To ensure that the domain ID is used, you should qualify the ID with the domain name, for example TRUSTED\BRKSID.

Note: When you invoke the **mqsicreatexxx** commands, your user ID must be a member of the Windows NT **Administrators** group on the system where the command is being executed.

User Name Server run-time operation: When you start the User Name Server the following are true:

- The User Name Server Windows NT service runs under the user ID UNSSID.

Security scenarios

- UNSSID is an indirect member of UNS**mqbrkrs**, CFGMGR**mqbrkrs** and BROKER**mqbrkrs** (through TRUSTED**Domain mqbrkrs**) and can therefore access appropriate MQSeries Integrator queues.
- Users and groups are sourced from the TRUSTED security domain.

Broker run-time operation: When you start the broker the following are true:

- The broker Windows NT service runs under the user ID BRKSID.
- BRKSID is an indirect member of BROKER**mqbrkrs**, UNS**mqbrkrs**, and CFGMGR**mqbrkrs**, and can therefore access the appropriate MQSeries Integrator queues.
- Users and groups are supplied by the User Name Server.

Configuration Manager run-time operation: When you start the Configuration Manager the following are true:

- The Configuration Manager Windows NT service runs under the user ID CMSID.
- CMSID is an indirect member of BROKER**mqbrkrs**, UNS**mqbrkrs**, and CFGMGR**mqbrkrs**, and can therefore access the appropriate MQSeries Integrator queues.
- Control Center users are members of the TRUSTED domain whose roles are determined by membership of the TRUSTED global groups (TRUSTED**Domain mqbrdevt** and so on).
- The list of principals from which ACLs are created are supplied by the User Name Server.

Application clients: Clients running application programs must run under user IDs defined to the security domain TRUSTED.

Scenario 3: operation on a stand-alone machine

This scenario describes the security setup required if you are using a single system and a local security domain. It makes the following assumptions:

1. The broker, User Name Server, and Configuration Manager are to be installed on a single system named SALONE.
2. The system SALONE is not configured into a primary domain.
3. Control Center users and application users will be defined in the SALONE domain.
4. Publish/subscribe access control for topic-based security is to be enabled.

Installation: During installation, the installation program performs the following configuration tasks when you install all components on system SALONE:

1. A local **mqbrkrs** group is created in the SALONE account domain.
2. The local groups **mqbrasgn**, **mqbrdevt**, **mqbrops**, and **mqbrtpic** are created in the SALONE account domain.

Service user IDs: When installation is complete, you must take the following actions:

1. Define a user ID BRKSID for the broker's service user ID in SALONE. Add this ID to the group SALONE**mqbrkrs**.
2. Define a user ID UNSSID for the User Name Server's service user ID in SALONE. Add this ID to the group SALONE**mqbrkrs**.

3. Define a user ID CMSID for the Configuration Manager's service user ID in SALONE. Add this ID to the group SALONE\mqbrkrs, and to SALONE\mqm group.

Creating the MQSeries Integrator components: You can now create the components:

1. Issue the **mqsicreateusername** command, specifying **-i UNSSID** and **-q UNSQMGR**. You must also either specify **-d SALONE**, or omit the **-d** parameter, in which case SALONE is configured by default.
2. Issue the **mqsicreateconfigmgr** command, specifying **-i CMSID** and **-s UNSQMGR**. You must also either specify **-d SALONE**, or omit the **-d** parameter, in which case SALONE is configured by default.
3. Issue the **mqsicreatebroker** command, specifying **-i BRKSID** and **-s UNSQMGR**.

Note: When you invoke the **mqsicreatexxx** commands, your user ID must be a member of the Windows NT **Administrators** group on system SALONE.

User Name Server run-time operation: When you start the User Name Server:

1. The User Name Server Windows NT service runs under the user ID UNSSID.
2. UNSSID is a member of SALONE\mqbrkrs and can therefore access appropriate MQSeries Integrator queues.
3. Users and groups are sourced from the SALONE security domain.

Broker run-time operation: When you start the broker:

1. The broker Windows NT service runs under the user ID BRKSID.
2. BRKSID is a member of SALONE\mqbrkrs, and can therefore access the appropriate MQSeries Integrator queues.
3. Users and groups are supplied by the User Name Server.

Configuration Manager run-time operation: When you start the Configuration Manager:

1. The Configuration Manager Windows NT service runs under the user ID CMSID.
2. CMSID is a member of SALONE\mqbrkrs and can therefore access the appropriate MQSeries Integrator queues.
3. Control Center users are members of the SALONE domain whose roles are determined by membership of the SALONE local groups (SALONE\mqbrdevt and so on).
4. The list of principals from which ACLs are created are supplied by the User Name Server.

Application clients: Clients running application programs must run under user IDs defined to the security domain SALONE.

Using UNIX security domains

Table 3 on page 62 summarizes the security requirements for the MQSeries Integrator administrative tasks.

For information about database authorization, see "Securing database resources" on page 63.

MQSeries Integrator resources

Table 3. Summary of administrative task authorization on UNIX platforms

User is...	UNIX domain
Creating broker, User Name Server	<ul style="list-style-type: none"> Member of mqbrkrs and mqm The component will run as this user (the service user ID), except when 'root' issues the create command, any user can be nominated as the service user ID.
Changing broker, User Name Server	<ul style="list-style-type: none"> Member of mqbrkrs
Deleting broker, User Name Server	<ul style="list-style-type: none"> Member of mqbrkrs and mqm
Starting and stopping broker, User Name Server	<ul style="list-style-type: none"> Member of mqbrkrs
Listing broker, User Name Server	<ul style="list-style-type: none"> Member of mqbrkrs
Changing, displaying, retrieving trace information	<ul style="list-style-type: none"> Member of mqbrkrs
Running User Name Server (service user ID)	<ul style="list-style-type: none"> Member of mqbrkrs. The component will run under the service user ID specified in the create command.
Running Configuration Manager (service user ID)	<ul style="list-style-type: none"> Not applicable on UNIX platforms
Running broker (MQSeries non-trusted application: fastpath off) (service user ID)	<ul style="list-style-type: none"> Member of mqbrkrs The broker will run under the service user ID specified in the create command.
Running broker (MQSeries trusted application: fastpath on) (service user ID)	<ul style="list-style-type: none"> Service user ID must be userID mqm UserID mqm must be a member of mqbrkrs
Clearing, joining, listing MQSeries Publish/Subscribe brokers	<ul style="list-style-type: none"> Member of mqbrkrs
Running Control Center	<ul style="list-style-type: none"> Not applicable on UNIX platforms
Running publish/subscribe applications	<ul style="list-style-type: none"> Any user, subject to MQSeries Integrator topic and MQSeries queue access control

Securing MQSeries resources

MQSeries Integrator depends on a number of MQSeries resources to operate successfully. You must control access to these resources to ensure that the product components can access the resources they depend on, and that these same resources are protected from other users.

Some authorizations are granted on your behalf when commands are issued. Others depend on the configuration of your broker domain.

- When you issue the command **mqsicreatebroker**, put and get authority is granted on your behalf to the group **mqbrkrs** for the following queues, by the command:
 - SYSTEM.BROKER.ADMIN.QUEUE
 - SYSTEM.BROKER.CONTROL.QUEUE
 - SYSTEM.BROKER.EXECUTIONGROUP.QUEUE
 - SYSTEM.BROKER.EXECUTIONGROUP.REPLY
 - SYSTEM.BROKER.INTERBROKER.QUEUE
 - SYSTEM.BROKER.MODEL.QUEUE
- When you issue the command **mqsicreateconfigmgr**:
 1. Put and get authority is granted on your behalf to the group **mqbrkrs** for the following queues, by the command:
 - SYSTEM.BROKER.CONFIG.QUEUE
 - SYSTEM.BROKER.CONFIG.REPLY
 - SYSTEM.BROKER.ADMIN.REPLY
 - SYSTEM.BROKER.SECURITY.REPLY
 - SYSTEM.BROKER.MODEL.QUEUE

2. Put and get authority is granted on your behalf to the groups **mqrdevt**, **mqrbrdevt**, **mqrbrdevt**, and **mqrbrtpic** for the following queues by the command:
 - SYSTEM.BROKER.CONFIG.QUEUE
 - SYSTEM.BROKER.CONFIG.REPLY
- When you issue the command **mqsicreateusername**, put and get authority is granted on your behalf to the group **mqrbrkrs** for the following queues, by the command:
 - SYSTEM.BROKER.SECURITY.QUEUE
 - SYSTEM.BROKER.MODEL.QUEUE
- If you have created MQSeries Integrator components to run on different queue managers, the transmission queues you define to handle the message traffic between the queue managers must have put and setall authority granted to the local **mqrbrkrs** group, or to the service user ID of the component supported by the queue manager on which the transmission queue is defined. (See “Connecting two MQSeries Integrator components” on page 22 for more details of these queues and channels).
- When you start up the Control Center, it connects to the Configuration Manager using an MQSeries client/server connection. For details of MQSeries channel security refer to “Setting up MQSeries client security” in the *MQSeries Clients* book.
- When you create, assign, and deploy a message flow:
 1. You must grant get and inq authority to each input queue identified in an MQInput node, for the broker’s ServiceUserID.
 2. You must grant put and inq authority to each output queue identified in an MQOutput node, or by an MQReply node, for the broker’s ServiceUserID.
 3. You must grant get authority to each output queue identified in an MQOutput node or an MQReply node to the user ID under which a receiving or subscribing client application runs.
 4. You must grant put authority to each input queue identified in an MQInput node to the user ID under which a sending or publishing client application runs.

You should also refer to “Application Security” in *MQSeries Integrator Introduction and Planning*.

Securing database resources

MQSeries Integrator creates and maintains essential configuration information in databases. When you have completed installation, you must create a minimum of one database that MQSeries Integrator can use for this information. When you issue the commands that create a broker and the Configuration Manager, tables are created within the database to hold the information required by that component. Other commands also need access to the database tables.

You must set up authorizations as follows:

- Creating, changing, or deleting a broker:
 1. Grant full access authority to the user ID specified as ServiceUserID. Follow the steps described in “Authorizing internal database access” on page 19 for the database you have used for your broker tables.
- Creating, changing, or deleting the Configuration Manager:
 1. Grant full access authority to the user ID specified as ServiceUserID, as described in “Authorizing internal database access” on page 19.

Database resources

- Migrating MQSeries Publish/Subscribe brokers:
 1. Grant read access authority to the user ID invoking the command, as described in “Authorizing internal database access” on page 19 for the database you have used for your broker tables.

DB2 services

On Windows NT

Some of the DB2 services run as Windows NT services and use and retain the user ID and password you specified when you installed this product. If you later change the password for this user ID, you must change it where it is retained in these services. You can do this by accessing the services from the Control Panel. If you do not do this, DB2 will fail to operate successfully.

For DB2 Version 6.1, the relevant services are DB2DAS00, DB2 Governor, and DB2 JDBC Applet Server.

For DB2 Version 7.1, the relevant services are DB2DAS00, DB2CTLSV and DB2 JDBC Applet Server - Control Center.

On UNIX platforms

DB2 on UNIX systems is started and stopped without needing a password.

Chapter 5. Problem determination

MQSeries Integrator provides commands and facilities that help you understand what is happening in your broker domain, and to allow you to find out more information when you need to.

It provides three major sources of information:

- "Traces".
- "Messages" on page 74.
- "Message flow debugger" on page 74.

You might also find it helpful to refer to additional information provided in the MQSeries Integrator SupportPac IH01. This has the latest problem determination information in a useful question-and-answer format. You can access this SupportPac from:

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

You can also access information recorded by other products that MQSeries Integrator interacts with:

- "MQSeries facilities" on page 74.
- "Database logs" on page 75.

Traces

MQSeries Integrator always records a minimum level of trace activity in the broker domain. You can activate further traces of the major product components (broker, Configuration Manager, and User Name Server), of the execution groups and message flows you define in a broker, and for utility programs, if you choose.

Trace records are available from the following sources:

- "Windows NT event log messages".
- "UNIX syslog messages" on page 66.
- "Optional traces" on page 67.

Windows NT event log messages

MQSeries Integrator components running on Windows NT use the Windows NT event log to record information about major activities within the system. All components provide diagnostic information in this form whenever error or warning conditions affect broker operation. These conditions include:

- Unsuccessful attempts to write a message to an MQSeries output queue.
- Errors interacting with databases.
- Inability to parse an input message.

When an error occurs, you are recommended to check the event log first. You can access the event log from the Windows NT Start menu, by selecting *Programs* → *Administrative Tools (Common)* → *Event Viewer*. You must select the Application view.

You cannot use an MQSeries Integrator command to switch off the event log. It is an operating system facility, and you must therefore control it using Windows NT commands. By default, it is always active. If you switch off this facility, you will lose valuable diagnostic information.

Traces

The MQSeries Integrator entries in the event log are all identified by the string MQSIv202 in the *Source* field of each record. If you press enter, or double-click a particular entry, further details are displayed in a separate dialog. The event identifier (shown in the *Event* field of the Application view, and as the *Event ID* value on the details view), is an MQSeries Integrator message number. The first part of the message shown in the dialog identifies the MQSeries Integrator component. You can look up these messages in *MQSeries Integrator Messages* using the number, preceded by the characters BIP, as the key.

UNIX syslog messages

MQSeries Integrator components running on UNIX platforms use syslog to record information about major activities within the system. All components send diagnostic information to syslog whenever error or warning conditions affect broker operation. These conditions include:

- Unsuccessful attempts to write a message to an MQSeries output queue.
- Errors interacting with databases.
- Inability to parse an input message.

When an error occurs, you are recommended to check syslog first. Where syslog messages are sent depends on how your UNIX system has been configured. To enable syslog to direct readable output from MQSeries Integrator to a text file called, for example, `mqsisislog`:

Step 1. Put the following line in the syslog configuration file (the default file is `/etc/syslog.conf`):

```
user.info /tmp/mqsisislog
```

Step 2. The output file `mqsisislog` must already exist before it can be written to. Therefore, create the file by using, for example:

```
touch /tmp/mqsisislog
```

Step 3. The `syslogd` daemon needs to re-read the configuration file to activate the command. Therefore, make the `syslogd` re-scan its configuration file. On AIX, enter the command:

```
refresh -s syslogd
```

On other UNIX platforms, enter the command:

```
kill -HUP `cat /etc/syslog.pid`
```

For other syslog options, see the documentation for your particular UNIX platform.

The MQSeries Integrator syslog entries are all identified by the string MQSIv202 and all have the form:

```
Date Time HostName MQSIv202: (BrokerName)[ThreadNumber]BIPnnnnX: Message:
SystemInformation
```

Where:

- `HostName` is the name of the machine where the message was produced.
- `BrokerName` is the name of the broker where the message was produced.
- `ThreadNumber` is the threadID of the broker process where the message was produced.
- `BIPnnnnx` is the MQSeries Integrator message identifier, where:
 - `nnnn` is the number of the message, and
 - `X` is a letter indicating the severity level of the message (one of I: information, W: warning, E: error, and S: serious).

- Message is the Message associated with the message identifier, which can be referenced in the *MQSeries Integrator Messages* book.
- SystemInformation is internal information which might be of use if you need to contact your IBM Support Center.

Note: On UNIX platforms, MQSeries Integrator uses the locale in which the broker is running to determine the local code page. (See “National language support” on page 5 for further details.)

MQSeries Integrator will match the locale with a codepage. If this codepage is not the one you require, you can override the local code page with the environment variable `MQSI_LOCAL_CCSDID` to inform MQSeries Integrator which local codepage you would like to use.

For example, if you have an unusual locale which uses code set ‘xyz’, you will need to export `MQSI_LOCAL_CCSDID=xyz` in the environment in which MQSeries Integrator is to run.

Optional traces

MQSeries Integrator provides optional trace facilities to provide additional recording of activity in the MQSeries Integrator components. These facilities are offered in two categories:

- **User tracing.** You can activate tracing of brokers, execution groups and assigned message flows.

You are very likely to use this option when you are resolving problems or unexpected behavior exhibited by your message flows. You can use the Control Center to control most of the trace activity you will need. It also enables you to start and stop tracing on systems that are remote to the Control Center.

You can also use the `mqsichangetrace` command which provides a wider range of parameters if you need wider flexibility. This command, like the commands to report trace options, and retrieve and format trace output, must be invoked on the system on which the component being traced is active.

You are also recommended to include a Trace node in your message flows when you are developing and testing them. This option not only gives you the ability to trace messages and activity in the flow, but also allows you to specify an alternate target file for the trace contents to isolate the detail you are interested in. For details of how to do this, see *MQSeries Integrator Using the Control Center*.

- **Service tracing.** You can activate more comprehensive broker tracing, and start tracing for the Control Center, Configuration Manager, and User Name Server. You can activate tracing for all the commands described in “Chapter 8. Commands” on page 89, including the trace commands themselves. You must use the commands to work with service trace: you cannot use the Control Center.

You are recommended to activate these traces only when you receive an error message that instructs you to start service trace, or when directed to do so by your IBM Support Center.

All optional traces are inactive by default, and must be explicitly activated by command when information is required over and above that provided by the entries in the Event log. When you set tracing on, you are causing additional processing to be executed for every activity in the component you are tracing. Large quantities of data are generated by the components.

Optional traces

You must therefore expect to see some impact in performance while trace is active, but you can get the best results by being selective about what you trace and by restricting the time during which trace is active.

MQSeries Integrator provides commands to enable you to control the optional traces:

- “Starting user trace”.
- “Checking user trace options” on page 69.
- “Changing user trace options” on page 69.
- “Retrieving user trace information” on page 69.
- “Formatting user trace information” on page 70.
- “Viewing and interpreting user trace information” on page 73.
- “Stopping user trace” on page 73.
- “Controlling Service traces” on page 73.

Starting user trace

You can start MQSeries Integrator user trace facilities by using the command **mqsichangetrace**, and, for execution groups and assigned message flows, from the Control Center. You can select only one broker on each invocation of the command, but you can activate concurrent traces for more than one broker if you want, by invoking the command more than once. For a full description of this command, see “mqsichangetrace (Change trace settings)” on page 95.

You must specify an individual execution group or message flow within the specified broker to limit the scope of a trace. The events recorded when you select the message flow option include:

- The sending of a message from one message processing node to the next.
- The detailed evaluation of expressions in a filter or compute node.

You can start trace at two levels, normal and debug:

- normal: tracks events that affect objects that you create, delete and can check in and out, such as nodes.
- debug: tracks the beginning and the end of processes as well as monitoring objects that are affected by that process.

The examples of “formatted normal level user trace” on page 71 and “formatted debug level user trace” on page 72 show the respective effects of these two settings.

Before you start to trace a broker, or any of its execution groups or message flows, you must have deployed these resources using the Control Center. For details of how to do this, see *MQSeries Integrator Using the Control Center*.

The user trace log files: When trace is active for any component (including the Control Center), information is recorded in binary form in files in the \log subdirectory of the MQSeries Integrator home directory.

The file names reflect the component and subcomponent for which the trace is active. For example, the broker name and unique execution group identifier form part of the file name when you are tracing activity within that execution group.

For example, if you have created a broker called MQSI_SAMPLE_BROKER, you might see the following files in the log subdirectory:

```
MQSI_SAMPLE_BROKER.682ec116-dc00-0000-0080-ce28a236e03d.userTrace.bin.1
MQSI_SAMPLE_BROKER.682ec116-dc00-0000-0080-ce28a236e03d.userTrace.bin.2
```

If the trace cannot be associated with a specific component, the component name part of the file name is set to *utility*, for example, when tracing a command such as `mqsilist`, when no arguments are used.

You cannot view these files directly, you must use the commands provided to access the trace information and convert it to a viewable format. See “Retrieving user trace information” and “Formatting user trace information” on page 70 for more details.

Example of starting user trace: If, for example, you want to start ‘normal’ level tracing for the execution group *default* on a broker you have created with the name `MQSI_SAMPLE_BROKER`, enter the command:

```
mqsichangetrace MQSI_SAMPLE_BROKER -u -e default -l normal
```

This command is described in full in “`mqsichangetrace` (Change trace settings)” on page 95. You can start the same trace from the Control Center. To do this, select the *Operations* view, right-click the *default* execution group, and select *User trace* → *Normal*. If you start the trace in this way, you can check that it has started successfully by selecting the *Log* view, and reviewing the log entries.

Checking user trace options

You can check what tracing options are currently active for your brokers by using the `mqsireporttrace` command. This command is described in full in “`mqsireporttrace` (Report trace settings)” on page 146.

You must specify the component for which the check is required. The command responds with the current trace status for the component you have specified.

Example of checking user trace options: If you want to check what options are currently set for the broker `MQSI_SAMPLE_BROKER` and its execution group *test*, enter the command:

```
mqsireporttrace MQSI_SAMPLE_BROKER -u -e default
```

If you had started tracing by following the example in “Starting user trace” on page 68, the response to the check command is:

```
BIP8098I: Trace level: debug, mode: safe, size: 1024 KB  
BIP8071I: Successful command completion
```

Changing user trace options

If you want to change the trace options you have set, you can use the `mqsichangetrace` command and set different values for the parameters. The command is fully described in “`mqsichangetrace` (Change trace settings)” on page 95. You can also use the Control Center to change the trace options for execution groups and assigned message flows.

Example of changing user trace options: To change from a debug level of trace to normal, enter the following command:

```
mqsichangetrace MQSI_SAMPLE_BROKER -u -e default -l normal
```

Retrieving user trace information

You can access the trace information recorded by the user trace facilities using the command `mqsireadlog`. This command retrieves the trace details according to parameters you specify on the command, and writes the requested records to a file, or to the command line window, in XML format. For a full description of this command, see “`mqsireadlog` (Read log)” on page 142.

Optional traces

Example of retrieving user trace information: To retrieve information for the trace activated with the `mqsischangeTrace` command (see “Starting user trace” on page 68), and write it to an output file, enter the command:

```
mqsireadlog MQSI_SAMPLE_BROKER -u -e default -o trace.xml
```

This sends a log request to the broker to retrieve the user trace log, and stores the responses in the file `trace.xml`. You can view this file using an XML editor or viewer which shows, for example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<UserTraceLog uuid="UserTraceLog"
userTraceLevel="none" traceLevel="none"
userTraceFilter="debugTrace" traceFilter="none" fileSize="1048576" fileMode="safe">
<UserTrace timestamp='2001-01-30 13:35:08.168000' thread='367'
function='ImbConfigurationNode::evaluate' type='ComIbmConfigurationNode'
name='ConfigurationNode' label='ConfigurationMessageFlow.ConfigurationNode'
text='Configuration changed successfully and committed to persistent store'
catalog='MQSIv202' number='4040'
file='F:/build/S202_P/src/DataFlowEngine/ImbConfigurationNode.cpp' line='778'>
<Insert type='string'>default</Insert>
<Insert type='string'>2d193932-dd00-0000-0080-ab4b84e10d89</Insert>
<Insert type='string'>ConfigurationMessageFlow.ConfigurationNode</Insert>
</UserTrace>
<UserTrace timestamp='2001-01-30 13:35:08.187999' thread='367'
function='ImbMqOutputNode::putMessage' type='ComIbmMQOutputNode'
name='OutputNode' label='ConfigurationMessageFlow.outputNode'
text='MQPUT issued to put message to the specified output queue'
catalog='MQSIv202' number='2638'
file='F:/build/S202_P/src/DataFlowEngine/ImbMqOutputNode.cpp' line='1474'>
<Insert type='string'>MQSI_SAMPLE_QM</Insert>
<Insert type='string'>SYSTEM.BROKER.EXECUTIONGROUP.REPLY</Insert>
<Insert type='integer'>0</Insert>
<Insert type='integer'>0</Insert>
<Insert type='string'>ConfigurationMessageFlow.outputNode</Insert>
</UserTrace>
```

Formatting user trace information

The trace information generated by the `mqsireadlog` command is not easy to read unless you use an XML viewer. MQSeries Integrator provides the command `mqsiformatlog` to format the trace information to a flat file, so you can view it using a text editor.

The format command takes a file generated by `mqsireadlog` as input, and flattens the XML log into structured records. It also retrieves the inserts for the XML message in the user’s current locale. You can specify the formatted output to be directed to a file, or viewed in the command line window.

Each user trace entry contains a timestamp and an MQSeries Integrator message, which contains a number (for example, BIP2622) and a text string containing variable inserts.

This command is described in full in “`mqsiformatlog (Format log)`” on page 127.

Example of formatting user trace information: To format the trace file created in “Retrieving user trace information” on page 69, enter the command:

```
mqsiformatlog -i trace.xml -o formattrace.log
```

This command reads the trace information in the file `trace.xml`, formats it, and writes it to the file `formattrace.log`. The following example shows a portion of the output when the extract of an unformatted file is formatted by `mqsiformatlog`:

Optional traces

Timestamps are formatted in local time, local time is GMT.

```
.  
. .  
2001-02-12 12:57:21.895999 388 UserTrace BIP2638E:  
MQPUT to queue 'SYSTEM.BROKER.EXECUTIONGROUP.REPLY' on queue manager 'MQSI_SAMPLE_QM':  
MQCC=0, MQRC=0; node 'ConfigurationMessageFlow.outputNode'. The node  
'ConfigurationMessageFlow.outputNode' attempted to write a message to the specified  
queue 'SYSTEM.BROKER.EXECUTIONGROUP.REPLY' connected to queue manager 'MQSI_SAMPLE_QM'.  
The MQCC was 0 and the MQRC was 0.  
No user action required.  
  
2001-02-12 12:57:21.895999 388 UserTrace BIP2622I:  
Message successfully output by output node 'ConfigurationMessageFlow.outputNode' to  
queue 'SYSTEM.BROKER.EXECUTIONGROUP.REPLY' on queue manager 'MQSI_SAMPLE_QM'. The  
MQSeries output node 'ConfigurationMessageFlow.outputNode' successfully wrote an output  
message to the specified queue 'SYSTEM.BROKER.EXECUTIONGROUP.REPLY' connected to queue  
manager 'MQSI_SAMPLE_QM'.  
No user action required.  
  
2001-02-12 12:58:20.319999 335 UserTrace BIP6060I:  
Parser type 'Properties' created on behalf of node 'Postcard.Receive Message' to handle  
portion of incoming message of length 0 bytes beginning at offset '0'. The message  
broker has created a parser of type 'Properties' on behalf of node 'Postcard.Receive  
Message' to handle the first part of an incoming message. This parser has been given  
the portion of the message starting at offset '0' and '0' bytes long.  
No user action required.  
  
2001-02-12 12:58:20.319999 335 UserTrace BIP6061I:  
Parser type 'MQMD' created on behalf of node 'Postcard.Receive Message' to handle  
portion of incoming message of length '364' bytes beginning at offset '0'. Parser type  
selected based on value 'MQHMD' from previous parser. The message broker has created a  
parser of type 'MQMD' on behalf of node 'Postcard.Receive Message' to handle  
a subsequent part of an incoming message. This parser has been given the portion of the  
message starting at offset '0' and '364' bytes long. This parser type was selected as  
the best match given the value 'MQHMD' from the previous parser.  
No user action required.  
  
2001-02-12 12:58:20.319999 335 UserTrace BIP2632I:  
Message being propagated to the output terminal; node 'Postcard.Receive Message'.  
An input message received from MQSeries input queue in node 'Postcard.Receive Message'  
is being propagated to any nodes connected to the output terminal.  
No user action required.  
  
2001-02-12 12:58:20.690000 335 UserTrace BIP6061I:  
Parser type 'MRM' created on behalf of node 'Postcard.Receive Message' to handle portion  
of incoming message of length '400' bytes beginning at offset '364'. Parser type  
selected based on value 'MRM' from previous parser. The message broker has created a  
parser of type 'MRM' on behalf of node 'Postcard.Receive Message' to handle a subsequent  
part of an incoming message. This parser has been given the portion of the message  
starting at offset '364' and '400' bytes long. This parser type was selected as the best  
match given the value 'MRM' from the previous parser.  
No user action required.  
  
2001-02-12 12:58:20.879999 335 UserTrace BIP4121I:  
Message propagated to out terminal from compute node 'Postcard.AddCountry'. The compute  
node 'Postcard.AddCountry' has received a message and is propagating it to any nodes  
connected to its out terminal.  
No user action required.  
  
2001-02-12 12:58:20.879999 335 UserTrace BIP2638E:  
MQPUT to queue 'MQSI_POSTCARD_OUTPUT_QUEUE' on queue manager '': MQCC=0, MQRC=0; node  
'Postcard.Send Message'. The node 'Postcard.Send Message' attempted to write a message  
to the specified queue 'MQSI_POSTCARD_OUTPUT_QUEUE' connected to queue manager ''.  
The MQCC was 0 and the MQRC was 0.  
No user action required.  
.
```

Optional traces

```
.  
.  
Threads encountered in this trace: 335 388
```

The above example shows the formatted output of a 'normal' level trace file. For comparison, the following example shows the equivalent portion of a similar 'debug' level trace:

Timestamps are formatted in local time, local time is GMT.

```
.  
.  
2001-02-12 13:01:24.854999 440 UserTrace BIP2638E:  
MQPUT to queue 'SYSTEM.BROKER.EXECUTIONGROUP.REPLY' on queue manager 'MQSI_SAMPLE_QM':  
MQCC=0, MQRC=0; node 'ConfigurationMessageFlow.outputNode'. The node  
'ConfigurationMessageFlow.outputNode' attempted to write a message to the specified  
queue 'SYSTEM.BROKER.EXECUTIONGROUP.REPLY' connected to queue manager 'MQSI_SAMPLE_QM'.  
The MQCC was 0 and the MQRC was 0.  
No user action required.  
.  
.  
2001-02-12 13:02:07.605999 444 UserTrace BIP6061I:  
Parser type 'MRM' created on behalf of node 'Postcard.Receive Message' to handle portion  
of incoming message of length '400' bytes beginning at offset '364'. Parser type  
selected based on value 'MRM' from previous parser. The message broker has created a  
parser of type 'MRM' on behalf of node 'Postcard.Receive Message' to handle a subsequent  
part of an incoming message. This parser has been given the portion of the message  
starting at offset '364' and '400' bytes long. This parser type was selected as the best  
match given the value 'MRM' from the previous parser.  
No user action required.  
2001-02-12 13:02:07.625999 444 UserTrace BIP2537I:  
Node 'Postcard.AddCountry': Executing statement 'SET OutputRoot = InputRoot;' at (1, 1).  
The statement being executed was 'SET OutputRoot = InputRoot;'.  
No user action required.  
2001-02-12 13:02:07.625999 444 UserTrace BIP2538I:  
Node 'Postcard.AddCountry': Evaluating expression 'InputRoot' at (1, 18).  
The expression being evaluated was 'InputRoot'.  
No user action required.  
2001-02-12 13:02:07.625999 444 UserTrace BIP2542I:  
Node 'Postcard.AddCountry': (1, 5) : Navigating path element.  
No user action required.  
2001-02-12 13:02:07.625999 444 UserTrace BIP2568I:  
Node 'Postcard.AddCountry': Performing tree copy of 'InputRoot' to 'OutputRoot'.  
The parsed message tree identified by path 'InputRoot' is being copied to 'OutputRoot'.  
No user action required.  
2001-02-12 13:02:07.625999 444 UserTrace BIP2537I:  
Node 'Postcard.AddCountry': Executing statement 'SET OutputRoot.MRM.Country = CASE  
InputRoot.MRM.Location WHEN 'Adelaide' THEN 'Australia' WHEN 'Buenos Aires'  
THEN 'Argentina' WHEN 'Florence' THEN 'Italy' WHEN 'London' THEN 'United Kingdom'  
WHEN 'New Delhi' THEN 'India' WHEN 'Paris' THEN 'France' WHEN 'Tokyo' THEN 'Japan'  
WHEN 'Toronto' THEN 'Canada' ELSE NULL END;' at (2, 1). The statement being executed  
was 'SET OutputRoot.MRM.Country = CASE InputRoot.MRM.Location WHEN 'Adelaide'  
THEN 'Australia' WHEN 'Buenos Aires' THEN 'Argentina' WHEN 'Florence' THEN 'Italy'  
WHEN 'London' THEN 'United Kingdom' WHEN 'New Delhi' THEN 'India' WHEN 'Paris'  
THEN 'France' WHEN 'Tokyo' THEN 'Japan' WHEN 'Toronto' THEN 'Canada' ELSE NULL END;'.  
No user action required.  
.  
.  
2001-02-12 13:02:07.727000 444 UserTrace BIP4121I:  
Message propagated to out terminal from compute node 'Postcard.AddCountry'. The compute
```


Optional traces

```
node 'Postcard.AddCountry' has received a message and is propagating it to any nodes
connected to its out terminal.
No user action required.
```

```
2001-02-12 13:02:07.727000 444 UserTrace BIP2638E:
MQPUT to queue 'MQSI_POSTCARD_OUTPUT_QUEUE' on queue manager '': MQCC=0, MQRC=0; node
'Postcard.Send Message'. The node 'Postcard.Send Message' attempted to write a message
to the specified queue 'MQSI_POSTCARD_OUTPUT_QUEUE' connected to queue manager ''.
The MQCC was 0 and the MQRC was 0.
No user action required.
.
.
.
Threads encountered in this trace: 440 444
```

Viewing and interpreting user trace information

A formatted log file, like the one illustrated in the example of “formatted normal level user trace” on page 71, contains a sequence of MQSeries Integrator messages that record the activity in a specific part of the system (the part that you identified when you started the trace). You can use this sequence to understand what is happening, and check that the behavior recorded is what you are expecting.

For example, if you have activated message flow trace, you can see entries that record the path a message takes through the message flow, and why decisions result in this path (where a choice is available).

If you are seeing unexpected behavior in a message flow, or execution group, you can use this trace information to check the actions taken and identify the source of an error or other discrepancy.

The messages contain identifiers for the resources that are being traced, for example the execution groups and message flows. The identifier given is usually the label (the name) you gave the resource when you defined it.

Stopping user trace

You can stop an active trace using the command **mqsichangetrace**. You must specify a trace level of none. This stops the trace activity for the component you specify on this command. It does not affect active traces on other components. For example, if you stop tracing on the execution group *test*, an active trace on another execution group will continue. For execution groups or assigned message flows, you can also use the Control Center to stop an active trace.

You should note that if you redeploy a component from the Control Center, trace for that component will be returned to its default setting; that is, none. A delta deploy will maintain the current settings.

Example of stopping user trace: To stop the trace started by the command shown in “Starting user trace” on page 68, enter the command:

```
mqsichangetrace MQSI_SAMPLE_BROKER -u -e default -l none
```

For a full description of this command, see “mqsichangetrace (Change trace settings)” on page 95.

Controlling Service traces

MQSeries Integrator supports more extensive service tracing in addition to the user level trace. You are recommended to use service traces only when you receive an error message that instructs you to start service trace, or when directed to do so by your IBM Support Center.

Optional traces

In summary:

- Service tracing of the brokers, the Configuration Manager, and the User Name Server are controlled using the trace commands that support and control user tracing.
- The Control Center can be traced by invoking it with a special command, **mqsilcc**, described in “mqsilcc (Start Control Center trace)” on page 133.
When you activate trace for the Control Center, tracing is also activated for the MQSeries Client for Java. For more information about this trace, refer to the *MQSeries Clients* book.
- The commands described in “Chapter 8. Commands” on page 89 (including the trace commands) can themselves be controlled by the existence and setting (on both Windows NT and UNIX platforms) of the environment variables **MQSI_UTILITY_TRACE** (which can be set to normal or debug) and **MQSI_UTILITY_TRACESIZE** (which defines the size of the trace file in KB, to a maximum of 2GB).

Messages

MQSeries Integrator produces messages in response to most requests for information and action. For example, when you invoke one of the commands detailed in “Part 2. Reference” on page 79, you will receive one or more messages indicating the success or failure of the command.

All MQSeries Integrator messages can be identified by the prefix BIP. These three characters are followed by a numeric string of four additional characters, which are a unique message identifier.

MQSeries Integrator Messages contains a detailed description of every message generated by MQSeries Integrator, and provides information about any action required by you in response to each message (for example, to correct an error).

Message flow debugger

The debugger is a robust problem determination tool that can be readily applied to complex message flows. For example, it can track single messages through a message flow one step at a time. Alternatively, by putting a breakpoint on one filter output, you can trap one ‘rogue’ message out of hundreds of ‘normal’ messages thus revealing an unexpected condition.

The debugger is accessed from the Control Center and so you should refer to *MQSeries Integrator Using the Control Center* for details of its use.

You should note that there is a minor administrative task which you might need to perform in order to successfully operate the debugger. You might need to set two system services to the minimum and maximum range of port numbers to be used by the Control Center when sending breakpoint information to the broker runtime for debugging purposes, if the defaults are not acceptable. This is explained fully in *MQSeries Integrator Using the Control Center*.

MQSeries facilities

MQSeries Integrator components depend on MQSeries resources in many ways. You can therefore gain valuable information from the MQSeries logs and events.

MQSeries logs

The MQSeries product logs can be very useful in diagnosing errors that occur in your broker domain. For example, if the Configuration Manager is unable to communicate with a broker, the channels that connect them might be wrongly configured, or experiencing network problems.

MQSeries writes entries into the local error log (that is, the Windows NT Event log or the UNIX syslog) as appropriate. It also creates queue manager logs, normally written to the errors subdirectory of the queue manager's subdirectory. Client logs are created in the errors subdirectory of the client's MQSeries directory. There are additional logs in the errors directory in the MQSeries data path. These logs might contain information about failures in the queue manager.

For more information about using MQSeries logs, see the *MQSeries System Administration* book.

MQSeries events

MQSeries provides information about errors, warnings, and other significant occurrences in a queue managers in the form of instrumentation events messages.

You can activate event activity using the MQSC or PCF interfaces in three areas:

- Queue manager events.
- Performance events.
- Channel events.

When active, these event messages are sent to event queues that can be monitored or triggered. You might find it appropriate to activate MQSeries events when you are investigating performance, or unexpected behavior, in your MQSeries Integrator broker domain.

For further details about MQSeries events, see *MQSeries Programmable System Management*.

Database logs

The database products used by MQSeries Integrator also record information that might be useful if you have any problems with their access. You should refer to the product documentation for details of logs and other problem determination options.

DB2 logs

DB2 has a number of facilities that assist you with problem diagnosis and recovery. For example, there are the error logs `db2diag.log` and `db2alert.log` that contain error and alert information recorded by various components of the DB2 product.

Refer to the *DB2 Troubleshooting Guide* for comprehensive information on what options are available, how to use them, and how to interpret the information provided.

ODBC tracing

On Windows NT only, you can initiate trace for ODBC activity by using the Tracing tab of the ODBC function available in the Windows NT Control Panel.

Contacting your IBM Support Center

If you are unable to resolve problems that you find when you use MQSeries Integrator, or if you are directed to do so by an error message generated by MQSeries Integrator, you can request assistance from your IBM Support Center.

Before you contact them, use the checklist below to gather key information. Some items might not necessarily be relevant in every situation. But you should provide as much information as possible to enable the IBM Support Center to recreate your problem.

- For MQSeries Integrator:
 - CSDs applied.
 - E-fixes applied.
 - All current trace and error logs, including relevant Windows NT Event log or UNIX platform syslog entries. User trace log files at debug level should be obtained for all relevant message flows and should preferably be formatted. (See “Optional traces” on page 67 for details of how to obtain and format these.)
 - A list of the components installed. This should include details of the number of machines and their operating systems, the number of brokers and the machine on which they are running, and the existence and details of any User Name Servers.
 - The file obtained by exporting your workspace or appropriate message flows. This action is performed from the Control Center; see *MQSeries Integrator Using the Control Center* for details of how to do this.
 - The files obtained by exporting all relevant message sets. This action is performed for each message set by using the `mqsiiimpexpmsgset` command with the `-e` flag set. See “`mqsiiimpexpmsgset` (Import/Export message set)” on page 129 for further details.
 - Details of the operation you were performing, the results that occurred, and the results you were expecting.
 - A sample of the messages being used when the problem arose.
 - If relevant, the report file from the C or COBOL importer. This is located in the directory from which the file import was attempted.
- For MQSeries:
 - CSDs applied.
 - E-fixes applied.
 - All current trace and error logs, including relevant Windows NT Event log or UNIX platform syslog entries and First Failure Support Technology™ (FFST) output files. You can find these files, which have the extension FDC, in the errors subdirectory within the MQSeries home directory.
 - Details of MQSeries client software, if appropriate.
- For each database you are using:
 - Product and release level (for example, DB2 7.1).
 - CSDs applied.
 - -- E-fixes applied.
 - -- All current trace and error logs, including relevant Windows NT Event log or UNIX platform syslog entries and First Failure Support Technology (FFST) output files. Check database product documentation for where to find these files.
- For Windows NT:

- Version.
- Service Pack level.
- The version of the system files `msvcrt.dll`, `msvcp60.dll`, `msvcirt.dll`, and `mfc42.dll`. You can find these files in the `WINNT\SYSTEM32` directory. Use the Windows NT Explorer file properties to display the versions.
- For Sun Solaris and HP-UX:
 - Version. You can find the version of Sun Solaris or HP-UX installed by using the `uname -a` command.
 - Service level applied.
- For AIX:
 - Version. You can find the version of AIX installed by using the `uname -a` command. More detailed information is available with the `lspp -l bos.rte` command.
 - Service level applied.

Contacting IBM

Part 2. Reference

Chapter 6. Using MQSeries Integrator commands	81	<code>mqsiclearmqpubsub</code> (Remove MQSeries Publish/Subscribe broker as a neighbor)	101
Rules for using MQSeries Integrator commands	81	Purpose	101
Rules for naming resources	82	Syntax	101
Responses to commands	83	Required parameters	101
Command syntax help	83	Authorization	101
How to read syntax diagrams	83	Responses	101
		Examples	102
Chapter 7. Using the MQSeries Integrator Command Assistant	87	Related commands	102
Overview	87	<code>mqsicopymsgset</code> (Copy message set)	103
Invocation	87	Purpose	103
Navigation	88	Syntax	103
Command processing	88	Required parameters	103
		Authorization	104
Chapter 8. Commands	89	Examples	104
<code>mqsichangebroker</code> (Change broker)	90	Related commands	104
Purpose	90	<code>mqsicreatebroker</code> (Create broker)	105
Syntax	90	Purpose	105
Required parameters	90	Syntax	106
Optional parameters	90	Required parameters	106
Authorization	91	Optional parameters	107
Responses	91	Authorization	108
Examples	92	MQSeries queues created	108
Related commands	92	Database tables created	108
<code>mqsichangeconfigmgr</code> (Change Configuration Manager)	93	Responses	109
Purpose	93	Examples	110
Syntax	93	Related commands	110
Optional parameters	93	<code>mqsicreateconfigmgr</code> (Create Configuration Manager)	111
Authorization	94	Purpose	111
Responses	94	Syntax	112
Examples	94	Required parameters	112
Related commands	94	Optional parameters	113
<code>mqsichangetrace</code> (Change trace settings)	95	Authorization	114
Purpose	95	MQSeries queues created	114
Syntax	96	MQSeries channels created	114
User trace	96	Database tables created	114
Service trace	96	Responses	115
Required parameters	96	Examples	116
Optional parameters	96	Related commands	116
Additional parameters exclusive to service trace	98	<code>mqsicreateusername</code> (Create User Name Server)	117
Authorization	98	Purpose	117
Responses	98	Syntax	117
Examples	98	Required parameters	117
Related commands	98	Optional parameters	118
<code>mqsichangeusername</code> (Change User Name Server)	99	Authorization	118
Purpose	99	MQSeries queues created	118
Syntax	99	Responses	119
Optional parameters	99	Examples	119
Authorization	100	Related commands	119
Responses	100	<code>mqsideletebroker</code> (Delete broker)	120
Examples	100	Purpose	120
Related commands	100	Syntax	121
		Required parameters	121
		Optional parameters	121

Authorization	121	Examples	136
Responses	121	mqsilistmqpubsub (List MQSeries	
Examples	121	Publish/Subscribe neighbor broker status).	137
Related commands	121	Purpose	137
mqsideleteconfigmgr (Delete Configuration		Syntax	137
Manager)	122	Required parameters	137
Purpose	122	Authorization	137
Syntax	122	Responses	137
Optional parameters	122	Examples	138
Authorization	123	Related commands	139
Responses	123	mqsinrfreload (Reload NEON messages)	140
Examples	124	Purpose	140
Related commands	124	Syntax	140
mqsideleteusername (Delete User Name		Required parameters	140
Server)	125	Authorization	140
Purpose	125	Responses	140
Syntax	125	Examples	141
Optional parameters	125	Related commands	141
Authorization	125	mqsireadlog (Read log)	142
Responses	125	Purpose	142
Examples	126	Syntax - user trace	142
Related commands	126	Syntax - service trace	142
mqsiformatlog (Format log)	127	Required parameters	142
Purpose	127	Optional parameters	143
Syntax	127	Additional parameters exclusive to service	
Required parameters	127	trace	143
Optional parameters	127	Authorization	144
Authorization	127	Responses	144
Responses	127	Examples	145
Examples	127	Related commands	145
Related commands	128	mqsireporttrace (Report trace settings)	146
mqsiimpexpmsgset (Import/Export message set)	129	Purpose	146
Purpose	129	Syntax	146
Syntax	129	User trace	146
Required parameters	129	Service trace	146
Authorization	130	Required parameters	146
Examples	130	Optional parameters	146
Related commands	130	Additional parameters exclusive to service	
mqsijoinmqpubsub (Join broker to MQSeries		trace	147
Publish/Subscribe parent broker)	131	Authorization	147
Purpose	131	Responses	147
Syntax	131	Examples	147
Required parameters	131	Related commands	147
Authorization	131	mqsisstart (Start component)	148
Responses	131	Purpose	148
Examples	132	Syntax	148
Related commands	132	Required parameters	148
mqsilcc (Start Control Center trace)	133	Authorization	148
Purpose	133	Responses	149
Syntax	133	Examples	149
Optional parameters	133	Related commands	149
Authorization	134	mqsisstop (Stop component)	150
Responses	134	Purpose	150
Examples	134	Syntax	150
mqsilist (List resources)	135	Required parameters	150
Purpose	135	Optional parameters	150
Syntax	135	Authorization	150
Optional parameters	135	Responses	151
Authorization	135	Examples	151
Responses	136	Related commands	151

Chapter 6. Using MQSeries Integrator commands

This chapter includes the following general information:

- “Rules for using MQSeries Integrator commands”.
- “Rules for naming resources” on page 82.
- “How to read syntax diagrams” on page 83.

Note: If you cut and paste examples of commands from the PDF version of this book to a command line for execution, you must check the content is correct before you press Enter. Some characters might be corrupted by local system and font settings.

Rules for using MQSeries Integrator commands

You should observe the following rules when using the MQSeries Integrator commands:

- Each command must be issued on the system on which the resource it relates to is defined (or is to be created).
- Each command starts with a primary keyword (the executable command name) followed by one or more blanks.
- Following the primary keyword, flags (parameters) can occur in any order. There is one exception to this; the command `mqsilcc` have positional parameters that must be specified in the order given.

- Flags are shown in this book in the form `-t`, for example. In all cases, the character `/` can be substituted for the `-` character.
- If a flag has a corresponding value, its value must follow the flag to which it relates. A flag can be followed by its value directly or can be separated by any number of blanks.

- Flags can be concatenated if they do not have corresponding values, although the last flag in a concatenated group *can* have a value associated with it. For example, the command:

```
mqsireadlog MQSI_SAMPLE_BROKER -u -e default -o trace.xml -f
```

could be entered as:

```
mqsireadlog MQSI_SAMPLE_BROKER -ufedefault -o trace.xml
```

where the name of the execution group, `default`, relates to the `-e` flag. For clarity, all examples given in this book are shown with separate flags and with a space before any associated value.

- Repeated flags are not allowed.
- Strings that contain blanks or special characters must be enclosed in double quotation marks. For example, you can specify a broker with the name "My Broker". Additionally, you can specify a 'null', or empty, string with a pair of double quotes with nothing between: `""`.
- The case sensitivity of primary keywords and parameters depends on the underlying operating system. On Windows NT, keywords are not case sensitive; `mqsistart`, `MQSISTART` and `MQSISTART` are all acceptable. On UNIX platforms, you should use lower case; only `mqsistart` is acceptable.

Rules for using commands

All MQSeries Integrator commands have dependencies on MQSeries function. You must ensure that MQSeries is available before issuing these commands.

Rules for naming resources

There are a few rules you must adhere to when you provide names or identifiers for the components and resources in your broker domain.

The components of the broker domain are:

- Brokers.
- The Configuration Manager.
- The User Name Server (this component is optional).

The broker resources are:

- Execution groups.
- Message sets and messages.
- message flows.
- Topics.

The character set that can be used for naming brokers, execution groups and message identifiers is as follows:

- Uppercase A-Z
- Lowercase a-z
- Numerics 0-9
- Any special characters supported by the underlying file system. The following are accepted on Windows NT:

\$	%	' (apostrophe)	' (quote)
- (dash)	_ (underscore)	@	~ (tilde)
!	()	{
}	[]	^
#	&	+	, (comma)
;	=		

and the following characters are accepted on UNIX platforms:

. (dot)	%	- (dash)	_ (underscore)
@	~ (tilde)	!	{
}	[]	^
#	, (comma)	=	

You can also use the space character, and any Unicode character with a decimal value greater than 127 (hexadecimal X'7F').

For all other resources, any characters that are supported by the database configuration are supported.

Broker names and fixed names (ConfigMgr and UserNameServer) are not case sensitive on the Windows NT platform. For example, broker names Broker1 and BROKER1 refer to the same broker. On UNIX platforms, they are case sensitive and the previous examples would refer to different brokers. You should use UserNameServer as shown. (In this context, ConfigMgr is not relevant.) A broker name specified in the Control Center must always match the case specified when the broker was created.

Note: These rules are specific to using the commands, the Command Assistant and the Control Center. There are additional rules for naming message service folders within the MQRFH2 header: these are described in the *MQSeries Integrator Programming Guide*.

Responses to commands

Responses are issued to the commands as messages. If a command is successful, it returns a return code of zero, and a message with the number BIP8071I (Command successful).

Warning and error responses are listed in the command descriptions. If the command is unsuccessful and returns, for example, the message BIP8083, it would have an exit code, in this case, of 83. You can check the full text of the message, and the explanation and action, in the *MQSeries Integrator Messages* book.

The following responses are returned by all the commands, and are not listed with each individual command:

- BIP8001 Unknown flag selected
- BIP8002 Selected flags incompatible
- BIP8003 Duplicate flag
- BIP8004 Invalid flags or arguments
- BIP8005 Flag or argument missing
- BIP8006 Mandatory flag missing
- BIP8007 Mandatory argument missing
- BIP8009 Program name invalid
- BIP8083 Invalid component name

Command syntax help

You can enter the characters “-?” or “/?” after every command to see the syntax it requires. This identifies optional and mandatory parameters. For example:

```
mqsicreateusernameserver /?
```

BIP8107W: Creates the User Name Server.

Syntax:

```
mqsicreateusernameserver -i ServiceUserId -a ServicePassword  
-q QueueManagerName [ -d SecurityDomainName]  
[ -r RefreshInterval ] [ -w WorkPath]
```

Command Options

i indicates the userid that the User Name Server should run under.

a the password for the User Name Server userid.

q indicates the MQSeries Queue Manager that the User Name Server should use. This is created if it does not exist.

d indicates the Security Domain that the User Name Server will use.

r number of seconds between each refresh of the User Name Server internal cache.

w indicates the directory into which trace logs are placed.

BIP8071I: Successful command completion.

How to read syntax diagrams

This book contains syntax diagrams (sometimes referred to as “railroad” diagrams).

Syntax 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 4. How to read syntax diagrams

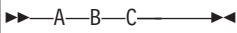
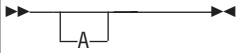
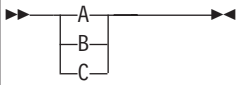
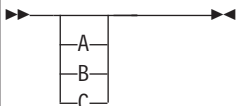
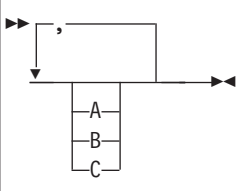
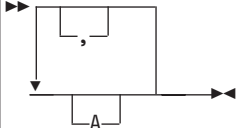
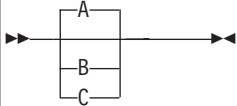
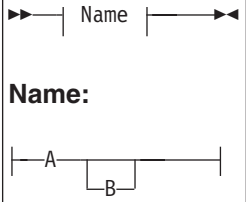
Convention	Meaning
	You must specify values A, B, and C. Required values are shown on the main line of a syntax diagram.
	You may specify value A. Optional values are shown below the main line of a syntax diagram.
	Values A, B, and C are alternatives, one of which you must specify.
	Values A, B, and 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.

Table 4. How to read syntax diagrams (continued)

Convention	Meaning
 <p>The diagram shows a horizontal line with vertical bars at each end. The word "Name" is centered on this line. From the left vertical bar, an arrow points left. From the right vertical bar, an arrow points right. Below the line, the letter "A" is positioned at the left end, and the letter "B" is positioned further to the right. A horizontal line connects "A" and "B", with a vertical line extending downwards from the center of this line to a horizontal bar that spans the distance between "A" and "B".</p>	<p>The syntax fragment Name is shown separately from the main syntax diagram.</p>
<p>Punctuation and uppercase values</p>	<p>Specify exactly as shown.</p>
<p>Lowercase values (for example, <i>name</i>)</p>	<p>Supply your own text in place of the <i>name</i> variable.</p>

Syntax diagrams

Chapter 7. Using the MQSeries Integrator Command Assistant

This chapter introduces the MQSeries Integrator Command Assistant, the preferred interface for configuring your MQSeries Integrator resources.

Overview

The Command Assistant is a graphical interface that supports a subset of the command line commands described in “Chapter 8. Commands” on page 89:

- **mqsicreatebroker**: create a broker.
- **mqsichangebroker**: change the attributes of a broker.
- **mqsideletebroker**: delete a broker.
- **mqsicreateconfigmgr**: create the Configuration Manager.
- **mqsichangeconfigmgr**: change the attributes of the Configuration Manager.
- **mqsideleteconfigmgr**: delete the Configuration Manager.
- **mqsicreateusernameserver**: create a User Name Server.
- **mqsichangeusernameserver**: change the attributes of a User Name Server.
- **mqsideleteusernameserver**: delete a User Name Server.

You should note that commands relating to the Configuration Manager are only available on Windows NT.

The Command Assistant provides a series of easy-to-use windows that significantly simplify the task of creating and changing components. The create commands, in particular, have a large number of parameters. The Command Assistant displays all the parameters with meaningful labels and provides integrated, context-sensitive help information, and indicates whether each parameter is mandatory or optional.

The Command Assistant does not change or enhance the function of the equivalent command in any way.

The Command Assistant does not provide support for starting and stopping these components. You must issue **mqsistart** and **mqsistop** at the command line.

Invocation

On Windows NT, you can invoke the Command Assistant from an icon in the MQSeries Integrator program folder. You can also access it from the Start menu (*Start* → *Programs* → *MQSeries Integrator 2.0.2* → *Command Assistant*). This presents a submenu from which you can choose the specific command you want to issue.

| On Solaris and HP-UX, first change to the subdirectory `/opt/mqsi/CmdAsst/`.

| On AIX, change to the subdirectory `/usr/opt/mqsi/CmdAsst/`.

Then issue one of the commands **create_**, **change_** or **delete_** with either **broker** or **uns** appended (for example, **create_broker** or **change_uns**).

Using the Command Assistant

Navigation

You can navigate both backwards and forwards through the Command Assistant using the **Next** and **Back** buttons. On Windows NT, you can also invoke help at any time by clicking the **Help** button. Help is not available on UNIX platforms.

When you have completed entering the parameters for the command you want to invoke, click **Finish**. If the command is successful, the Command Assistant terminates. If the command returns any error information, this is displayed in the **Error Log** pane.

For assistance in responding to the errors shown, see the *MQSeries Integrator Messages* book. If you can correct the error by changing the parameters of this command, you can return to the window on which you must make the change by clicking **Back**.

Command processing

Using the Command Assistant helps you specify the parameters you need, and those you choose to use, for the commands it supports. It provides help for every parameter to assist you in specifying correct values.

When you have specified the values, the Command Assistant builds a command string equivalent to the one that you can enter at the command line. It displays this command before it executes it, so you can check it and change it (by clicking **Back** and returning to the previous screens) if you want. You can also save this command string to a file if you want, by using cut and paste.

If you believe the command string shown is correct, click **Finish**. The command is executed, and the response to the command is displayed in the lower pane of the final window. If the command is successful, the message BIP8071I is displayed. If the command is not successful, one or more error messages are displayed. Once you have corrected the error, click **Back** to reenter the parameter values, and reissue the command.

The commands and parameters are shown in detail in “Chapter 8. Commands” on page 89.

Chapter 8. Commands

This chapter describes the commands provided by MQSeries Integrator. They are listed below, grouped by function, with an indication of the computer operating system on which they are available:

Command	Available on:	
	Windows NT	UNIX platforms
Broker commands		
mqscreatebroker	page 105	✓
mqschangebroker	page 90	✓
mqsdeletebroker	page 120	✓
mqsimigratebroker	See "Upgrading your Version 2.0 brokers" on page 156	
Message set commands		
mqsicopymsgset	page 103	✓
mqsiiimpexpmsgset	page 129	✓
mqsinrfreload	page 140	✓
Configuration Manager commands		
mqscreateconfigmgr	page 111	✓
mqschangeconfigmgr	page 93	✓
mqsdeleteconfigmgr	page 122	✓
mqsimigrateconfigmgr	See "Upgrading your Version 2.0 Configuration Manager" on page 157	
User Name Server commands		
mqscreateusernameserver	page 117	✓
mqschangeusernameserver	page 99	✓
mqsdeleteusernameserver	page 125	✓
Start and stop commands		
mqsistart	page 148	✓
mqsistop	page 150	✓
List and trace commands		
mqsilist	page 135	✓
mqsichangetrace	page 95	✓
mqsiformatlog	page 127	✓
mqsilcc	page 133	✓
mqsireadlog	page 142	✓
mqsireporttrace	page 146	✓
MQSeries Publish/Subscribe interoperability commands		
mqsiclearmqpubsub	page 101	✓
mqsijoinmqpubsub	page 131	✓
mqsilistmqpubsub	page 137	✓

mqsichangebroker (Change broker)

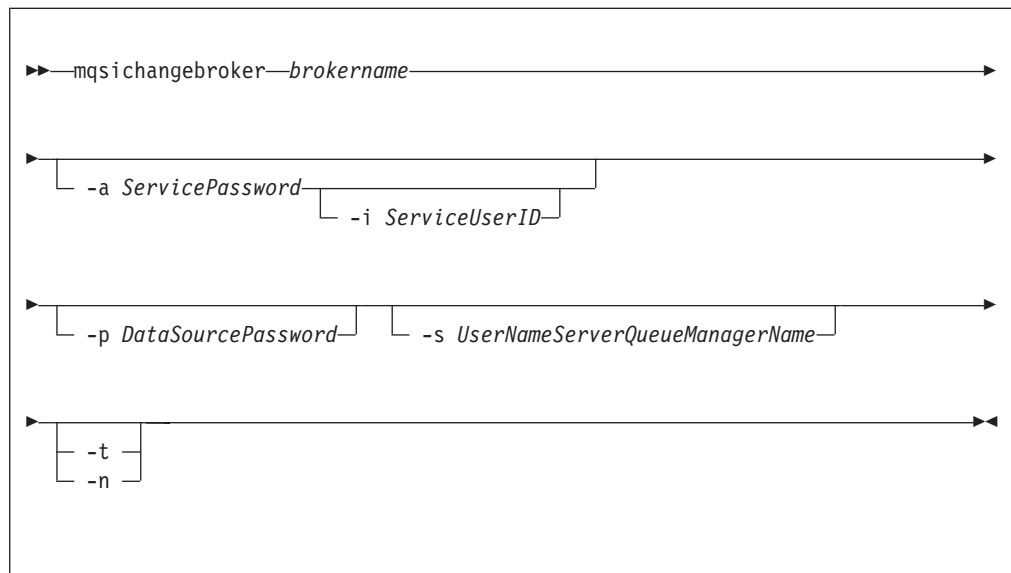
Purpose

Use the **mqsichangebroker** command to change some of the properties of a broker.

You must stop the broker, using **mqsistop**, before you can issue this command. When you restart the broker, using **mqsistart**, it uses the changed parameters.

You can also use the Command Assistant to issue this command.

Syntax



Required parameters

brokername

This must be the first parameter. Specify the name of the broker you want to modify.

Optional parameters

-a ServicePassword

The password for the ServiceUserID. On UNIX platforms -a is required for Windows NT compatibility, but is not used in relation to ServiceUserID; it is only used as a default if -p is not specified. (See notes about the -p parameter for further details.)

-i ServiceUserID

The user ID under which the broker will run. You can only change this value if you also change the password.

This can be specified in any valid username syntax. On Windows NT, these are:

- domain\username
- \\server\username
- .\username
- username

You should note that on UNIX systems, only the last format, username, is valid.

The ServiceUserID specified must be a member of the local group **mqbrkrs**. On Windows NT, it can be an indirect or direct member of the group. The ServiceUserID must also be authorized to access the home directory (where MQSeries Integrator has been installed), and the working directory (if specified by the `mqsiccreatebroker -w` flag). This ID must also be a member (either direct or indirect) of the local group **mqm**.

The security requirements for the ServiceUserID are detailed in Table 2 on page 54 for Windows NT and in Table 3 on page 62 for UNIX platforms.

Note: For Windows NT: If you use the unqualified form for this user ID (username), the operating system searches for the user ID throughout its domain, starting with the local system. This search might take some time to complete.

-p *DataSourcePassword*

The password of the user ID with which the database containing the broker tables is to be accessed. If not specified, this defaults to the ServicePassword specified by `-a`. For DB2 on UNIX platforms, `-u` and `-p` can be specified as empty strings (""). In this case, DB2 grants MQSeries Integrator the privileges of the MQSeries Integrator ServiceUserID which results in a database connection as "already verified". If you specify `-a` as an empty string as well as `-u` and `-p`, then no passwords are stored by MQSeries Integrator, creating the most secure configuration.

-s *UserNameServerQueueManagerName*

The name of the MQSeries queue manager that is associated with the User Name Server. If you want to remove topic-based security, specify "".

-t Requests that the broker runs as an MQSeries trusted application.

For more details about using MQSeries trusted applications, see "Using MQSeries trusted applications" on page 49 and *MQSeries Intercommunication*.

-n Requests that the broker ceases to run as an MQSeries trusted application.

For more details about using MQSeries trusted applications, see "Using MQSeries trusted applications" on page 49 and *MQSeries Intercommunication*.

If you want to change other broker properties, you must delete and recreate the broker, then use the Control Center to redeploy the broker's configuration. If you want to change the user ID used for database access, see "Managing databases" on page 36.

Authorization

On Windows NT, the user ID used to invoke this command must have **Administrator** authority on the local system.

On UNIX platforms, the user ID used to invoke this command must either be **root** or must be the same as that specified in the `-i` parameter. It must also be a member of the **mqbrkrs** group.

Responses

This command returns the following responses:

- BIP8012 Unable to connect to system components
- BIP8013 Component does not exist

mqsichangebroker

- BIP8018 Component running
- BIP8021 User ID/password incorrect
- BIP8022 Invalid user ID/password
- BIP8023 Password required
- BIP8030 Unable to modify user ID privileges
- BIP8073 Invalid broker name

Examples

```
mqsichangebroker MQSI_SAMPLE_BROKER -s MQSI_SAMPLE_UNQ_QM
```

Related commands

“mqsicreatebroker (Create broker)” on page 105

“mqsdeletebroker (Delete broker)” on page 120

mqsichangeconfigmgr (Change Configuration Manager)

Purpose

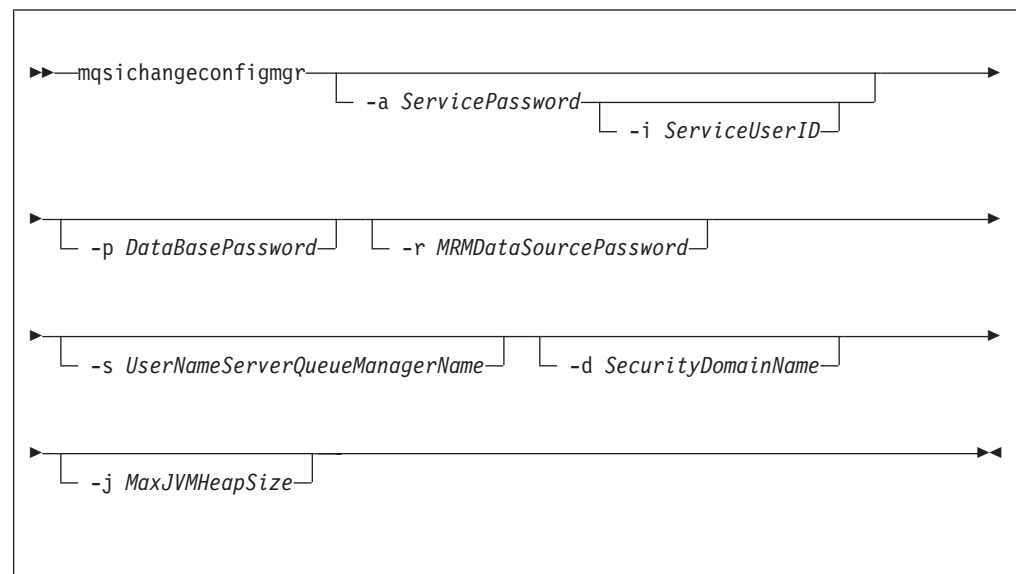
Use the **mqsichangeconfigmgr** command to change some of the properties of the Configuration Manager.

You should only use the Configuration Manager on a Windows NT platform. This command is therefore only applicable when using that operating system.

You must stop the Configuration Manager, using **mqsistop**, before you can issue this command. When you restart the Configuration Manager, using **mqsistart**, it uses the changed parameters.

You can also use the Command Assistant to issue this command.

Syntax



Optional parameters

-a ServicePassword

The password for the ServiceUserID.

-i ServiceUserID

The user ID under which the Windows NT service must run. You can only change this value if you also change the password.

This can be specified in any valid Windows NT username syntax:

- domain\username
- \\server\username
- .\username
- username

You should note that on UNIX systems, only the last format, username, is valid.

The ServiceUserID specified must be a member of the local group **mqbrkrs**. It can be an indirect or direct member of the group. The ServiceUserID must also be authorized to access the home directory (where MQSeries Integrator has

mqsichangeconfigmgr

been installed), and the working directory (if specified by the mqsicreateconfigmgr **-w** flag). This ID must also be a member (either direct or indirect) of the local group **mqm**.

The security requirements for the ServiceUserID are detailed in Table 2 on page 54.

Note: For Windows NT: If you use the unqualified form for this user ID (username), the operating system searches for the user ID throughout its domain, starting with the local system. This search might take some time to complete.

-p *DataBasePassword*

The password for the user ID with which the configuration repository database is to be accessed.

-r *MRMDataSourcePassword*

The password for the user ID with which the message repository database is to be accessed.

-s *UserNameServerQueueManagerName*

The name of the MQSeries queue manager that is associated with the User Name Server. If you want to remove topic security, specify "".

-d *SecurityDomainName*

The name of the Windows NT security domain. For details about implementation of security, see "Chapter 4. Setting up security" on page 51.

-j *MaxJVMHeapSize*

The maximum Java virtual machine (JVM) heap size, in megabytes. The smallest value that can be set is 64. If this parameter is not set, the default size of 128 MB is used.

If you want to change other properties, you must delete and recreate the Configuration Manager. If you want to change the DataBaseUserID or the MRMDataSourceUserID, see "Managing databases" on page 36 for instructions.

Authorization

The user ID under which the command is invoked must have Windows NT **Administrator** authority on this local system.

Responses

This command returns the following responses:

- BIP8012 Unable to connect to system components
- BIP8013 Component does not exist
- BIP8018 Component running
- BIP8021 User ID/password incorrect
- BIP8022 Invalid user ID/password
- BIP8023 Password required
- BIP8030 Unable to modify user ID privileges

Examples

```
mqsichangeconfigmgr -d MQSI_DOMAIN
```

Related commands

"mqsicreateconfigmgr (Create Configuration Manager)" on page 111

"mqsideleteconfigmgr (Delete Configuration Manager)" on page 122

mqsichangetrace (Change trace settings)

Purpose

Use the **mqsichangetrace** command to set the tracing characteristics for a component. This command is valid for:

- User trace. Specify the **-u** option.
- Service trace. Specify the **-t** option. You are recommended to use this option only if directed to do so by the action described in a BIPxxxx message, or by your IBM Support Center.

You can initiate, modify, or terminate user tracing for a broker, or initiate, modify, or terminate service tracing for a broker, the Configuration Manager, or the User Name Server (identified by component name). You cannot use this command to initiate service tracing for the Control Center: you must use the **mqsilcc** command.

You can also start and stop tracing activity for execution groups and message flows using the facilities of the Control Center. See *MQSeries Integrator Version 2.0.2 Using the Control Center* for more information.

If you specify a broker, or any of its resources, (execution group or message flow), you must have deployed them before you can start trace.

The trace output generated by these commands is written to trace files in the log subdirectory. When you have completed the work you want to trace, you can use **mqsireadlog** to retrieve the log as an XML format file. You can use either **mqsiformatlog** (to produce a formatted file) or an XML browser to view the XML records.

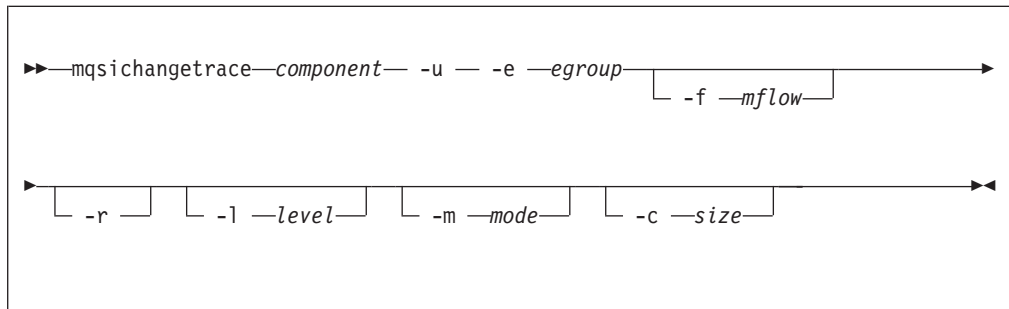
When you set tracing on, you are causing additional processing to be executed for every activity in the component you are tracing. You must therefore expect to see some impact on performance when trace is active.

For more information on using this command, and examples of its use, see "Chapter 5. Problem determination" on page 65.

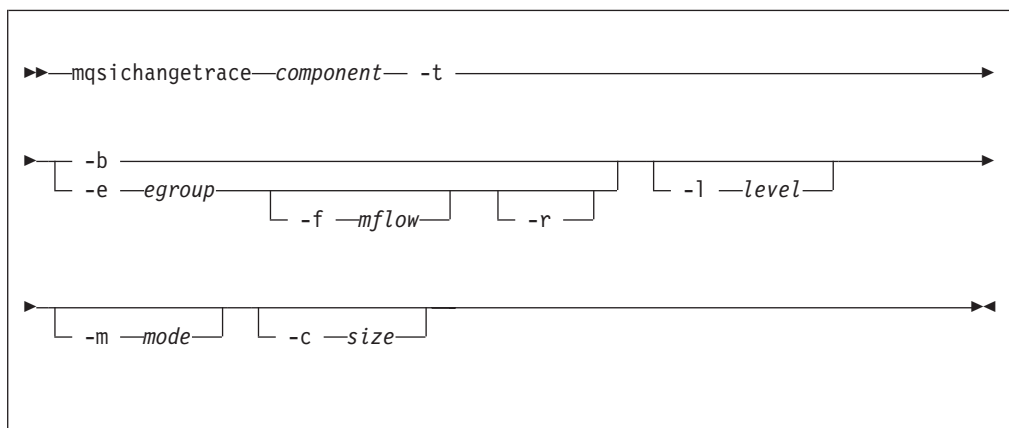
You cannot start tracing for the commands described in this chapter using this command. For tracing these command executables, you must use the environment variables **MQSI_UTILITY_TRACE** and **MQSI_UTILITY_TRACESIZE** described in "Controlling Service traces" on page 73.

Syntax

User trace



Service trace



Required parameters

component

The name of the component for which trace parameters are to be changed. This can either be the name of a broker, or the fixed values ConfigMgr or UserNameServer (all are case sensitive on UNIX platforms).

Optional parameters

-u Specifies that user trace options are to be modified. This option is only valid if you have specified a broker name as the component name.

-e egroup

Identifies the execution group for which trace options are to be modified (for example, started or stopped). This option is only valid if you have specified a broker name as the component name.

-f mflow

Identifies the message flow for which trace options are to be modified. This option is only valid if you have specified an execution group (flag -e).

-r This option requests that the trace log is reset: that is, all current records are discarded. You can use this option when you start a new trace to ensure that all records in the log are unique to the new trace.

This option is only valid if you have specified an execution group (flag -e).

-l level

Set the level of the trace. This must be one of:
 normal. This provides a basic level of trace information.
 none. This sets tracing off.
 debug. This provides a more comprehensive trace.

Each component is created with a default value of none. If you do not specify this parameter, the current value is unchanged. Once you successfully change this value, it is persistent.

This is valid for all components.

-m mode

Indicate the way trace information is to be buffered:

safe. This mode causes trace entries to be written to file when they are generated.

fast. This mode causes trace entries to be buffered, and only written to file in batches.

Each component starts with a default value of safe. If you do not specify this parameter, the current value is unchanged.

This option is only valid if the component you have specified is:

- A broker. If you change this value, it affects tracing for the execution group (if you have specified one), or for the agent component (if you have not specified an execution group).
- The User Name Server. If you change this value, it affects tracing for the entire component. (This is only valid for service trace.) Once you successfully change this value, it is persistent.

-c size

The size of the trace file in KB (kilobytes). If you do not specify this parameter, the current value is left unchanged. Each component starts with a default value of 1024KB. You can specify this option to reset the value. The maximum value you can specify depends on how you subsequently intend to read the log, using the **mqsireadlog** command:

- If you use **mqsireadlog** with the **-f** option set, the log file is read directly from the file system. In this case, the maximum value that can be specified here is 2097151, which will allow a trace file up to 2GB (gigabyte) to be created.
- If you use **mqsireadlog** without setting the **-f** option, an MQ Series message is sent to the broker to retrieve the log. In this case, the trace file size should not be allowed to exceed about 70MB (megabytes). The maximum value that you can set here, should not be appreciably more than 70000.

However you intend to retrieve the trace file, you are recommended to keep its size as small as possible, either by using a low value for this parameter or by using the reset (**-r**) option on this command to clear the trace log. The benefit of adopting this approach is that the formatting process (**mqsiformatlog**) will consequently be much faster and require less resource to carry out its task.

This option is only valid if the component you have specified is:

- A broker. If you change this value, it affects tracing for the execution group (if you have specified one), or for the agent component (if you have not specified an execution group).

mqsichangetrace

- The User Name Server. If you change this value, it affects tracing for the entire component. (This is only valid for service trace.)

If you change the trace size, the new value is persistent over a restart of the broker or User Name Server.

Additional parameters exclusive to service trace

You are recommended to use these options only when directed to do so by your IBM Support Center or by a BIPxxxx message.

- t Specifies that service trace options are to be modified.
- b Specifies that service trace options for the agent subcomponent of the component specified are to be modified (for example, started or stopped). This flag can only be specified if -t is also specified.

Authorization

The user ID used to issue the command must have **mqbrkrs** authority.

Responses

This command returns the following responses:

- BIP8013 Component does not exist
- BIP8020 Unable to access database
- BIP8029 Broker not configured
- BIP8031 Invalid flag supplied
- BIP8032 Unable to connect to queue
- BIP8033 Message send failure
- BIP8035 Response not received before timeout
- BIP8036 Negative response received
- BIP8037 Unsupported flag
- BIP8039 Execution group not available
- BIP8040 Unable to connect to database
- BIP8045 Message flow not found
- BIP8068 Integer argument required

Examples

```
mqsichangetrace MQSI_SAMPLE_BROKER -u -e default -l normal -c 5000
```

```
mqsichangetrace MQSI_SAMPLE_BROKER -u -e "exg1" -m fast
```

```
mqsichangetrace UserNameServer -t -b -l normal
```

Related commands

“mqsiformatlog (Format log)” on page 127

“mqsilcc (Start Control Center trace)” on page 133

“mqsireadlog (Read log)” on page 142

“mqsireporttrace (Report trace settings)” on page 146

mqsicchangeusernameserver (Change User Name Server)

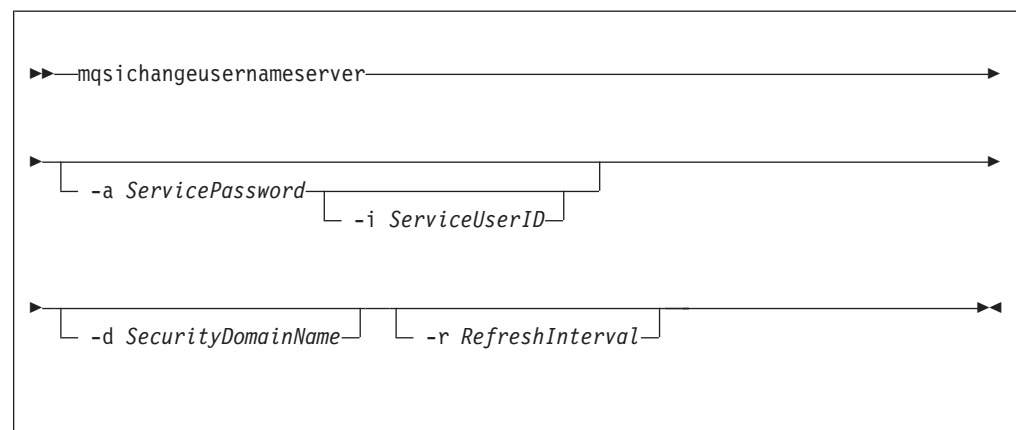
Purpose

Use the **mqsicchangeusernameserver** command to change some of the properties of the User Name Server.

You must stop the User Name Server, using **mqsisstop**, before you can issue this command. When you restart the User Name Server, using **mqsisstart**, it uses the changed parameters.

You can also use the Command Assistant to issue this command.

Syntax



Optional parameters

-a *ServicePassword*

The password for the ServiceUserID.

-i *ServiceUserID*

The user ID under which the broker will run. You can only change this value if you also change the password.

This can be specified in any valid username syntax. On Windows NT, these are:

- domain\username
- \\server\username
- .\username
- username

You should note that on UNIX systems, only the last format, username, is valid.

The ServiceUserID specified must be a member of the local group **mqbrkrs**. On Windows NT, it can be an indirect or direct member of the group. The ServiceUserID must also be authorized to access the home directory (where MQSeries Integrator has been installed), and the working directory (if specified by the mqsiccreateusernameserver **-w** flag). This ID must also be a member (either direct or indirect) of the local group **mqm**.

mqschangeusernameserver

The security requirements for the ServiceUserID are detailed in Table 2 on page 54 for Windows NT platforms and in Table 3 on page 62 for UNIX platforms.

Note: For Windows NT: If you use the unqualified form for this user ID (username), the operating system searches for the user ID throughout its domain, starting with the local system. This search might take some time to complete.

-d *SecurityDomainName*

The name of the Windows NT security domain. For details about implementation of security, see "Chapter 4. Setting up security" on page 51.

-r *RefreshInterval*

The interval, in seconds, at which the User Name Server interrogates the security subsystem for changes to user or group attributes.

Authorization

On Windows NT, the user ID used to invoke this command must have **Administrator** authority on the local system.

On UNIX platforms, the user ID used to invoke this command must either be **root** or must be the same as that specified in the **-i** parameter. It must also be a member of the **mqbrkrs** group.

Responses

This command returns the following responses:

- BIP8012 Unable to connect to system components
- BIP8013 Component does not exist
- BIP8018 Component running
- BIP8021 User ID/password incorrect
- BIP8022 Invalid user ID/password
- BIP8023 Password required
- BIP8030 Unable to modify user ID privileges
- BIP8068 Integer argument required

Examples

```
mqschangeusernameserver -r 2000
```

Related commands

"mqscreateusernameserver (Create User Name Server)" on page 117

"mqsdeleteusernameserver (Delete User Name Server)" on page 125

mqsiclearmqpubsub (Remove MQSeries Publish/Subscribe broker as a neighbor)

Purpose

Use the **mqsiclearmqpubsub** command to remove an MQSeries Publish/Subscribe broker as a neighbor of this MQSeries Integrator broker.

This command removes knowledge of the MQSeries Publish/Subscribe broker from the MQSeries Integrator broker identified on this command. To complete this action you must also issue the MQSeries Publish/Subscribe command **clrmqbrk** against the MQSeries Publish/Subscribe broker. When both clear commands have completed, all publish/subscribe traffic between the two brokers ceases.

Only use this command if you are integrating this MQSeries Integrator broker with an MQSeries Publish/Subscribe broker network. Before you issue this command you must ensure that the MQSeries Integrator broker is ready to receive and process messages on queue SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS (that is, you must have restarted the broker after creating this queue. See “Creating and operating a heterogeneous network” on page 163 for more details.)

Syntax

```
mqsiclearmqpubsub—brokername— -n NeighborQueueManagerName
```

Required parameters

brokername

The name of the broker from which knowledge of an MQSeries Publish/Subscribe neighbor broker is to be removed.

-n *NeighborQueueManagerName*

The name of the queue manager that hosts the MQSeries Publish/Subscribe broker for which the association as a neighbor is being removed.

Authorization

The user ID used to invoke this command must have put and inq authority to the queue SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS.

Responses

This command returns the following responses:

- BIP8013 Component does not exist
- BIP8056 Unknown queue manager
- BIP8057 Queue manager error
- BIP8059 Queue manager not available
- BIP8060 Queue error
- BIP8061 No reply received
- BIP8064 Internal broker error
- BIP8066 Invalid broker name

mqsiclearmqpubsub

- BIP8070 Database exception
- BIP8072 Database exception

Examples

```
mqsiclearmqpubsub MQSI_SAMPLE_BROKER -n MQBroker1
```

Related commands

“mqsijoinmqpubsub (Join broker to MQSeries Publish/Subscribe parent broker)” on page 131

“mqsilistmqpubsub (List MQSeries Publish/Subscribe neighbor broker status)” on page 137

mqsicopymsgset (Copy message set)

Purpose

Use the **mqsicopymsgset** command to create a copy of a complete message within the same message repository. This allows quick and easy creation of message sets similar in structure.

When you have successfully copied a message set, you must stop the Configuration Manager and restart it. Your new message set will then be available to add to your workspace.

When you copy a message set, only those resources previously checked into the repository are copied to the new message set. All resources are created in the new message set as exact copies of the resources in the source message with the exception of the source message set context information. That is, message set state, resource locks and resource states are not copied from the source message set.

This command is only applicable when using the Windows NT operating system.

Note: The command '**mqsimrmcopymsgset**' is deprecated.

The **mqsicopymsgset** command replaces **mqsimrmcopymsgset**, used in previous versions of MQSeries Integrator. You should use this new command in preference to the old version, which is no longer supported. Unlike the old command, parameters now follow the normal convention and are *not* positional.

Syntax

```

▶▶—mqsicopymsgset— -n MRMDataSourceName— -u MRMDataSourceUserID—————▶
▶ -p MRMDataSourcePassword— -s SourceMessageSetName—————▶
▶ -l SourceMessageSetLevel— -t TargetMessageSetName—————▶
▶ -k TargetMessageSetLevel—————▶▶

```

Required parameters

-n *MRMDataSourceName*

The ODBC data source name (DSN) of the database created to hold the message repository tables.

This database must already exist. You must create a System DSN ODBC connection for this DSN, if you have not already done so.

The *MQSeries Integrator Installation Guide* for your computer platform contains details of how to create a database and an ODBC connection for all databases required by MQSeries Integrator.

mqsicopymsgset

-u *MRMDataSourceUserID*

The user ID with which the message repository database is to be accessed. This user ID must have the authority to read from (export) and write to (import) the database identified by *MRMDataSourceName*.

-p *MRMDataSourcePassword*

The password for the user ID with which the message repository database is to be accessed.

-s *SourceMessageSetName*

The name (not the identifier) of the message set to be copied.

-l *SourceMessageSetLevel*

The level of the message set to be copied.

-t *TargetMessageSetName*

The name (not the identifier) of the message set to be created.

-k *TargetMessageSetLevel*

The level of the message set to be created.

Authorization

None.

Examples

```
mqsicopymsgset -n MQSIMRDB -u mqsuid -p mqsipw  
-s "Accounts Payable" -l 1 -t "Accounts Payable Copy" -k 1
```

Related commands

“*mqsiiimpexpmsgset* (Import/Export message set)” on page 129

“*mqsinrfreload* (Reload NEON messages)” on page 140

mqsicreatebroker (Create broker)

Purpose

Use the **mqsicreatebroker** command to create a new broker.

This command allows you to specify all the properties required to create a new broker.

This command performs the following actions:

- Creates an MQSeries queue manager, if one does not already exist.

Note: If this command creates an MQSeries queue manager, it will also enable the default dead-letter queue (DLQ) provided by MQSeries (SYSTEM.DEAD.LETTER.QUEUE). The security settings are the same as those of other broker-specific MQSeries queues.

If a message in either a user-defined message flow or in the publish subscribe model cannot be processed, it will be routed to this DLQ as a last resort. If instead, you would prefer the message to be backed out onto the input queue, effectively halting the message flow until the problem is resolved, you should disable the DLQ.

The **mqsideletebroker** command will not delete this queue (unless the Queue Manager is deleted).

- Starts the MQSeries queue manager, if this is not already running.

Note: If the queue manager is created by this command on Windows NT, it is not started as a service; it will therefore stop if you log off. To avoid this happening, you must either remain logged on, or you must change the start up status of the queue manager service (described in “Starting MQSeries queue managers as a Windows NT service” on page 24). (If you lock your workstation, the MQSeries queue manager does not stop).

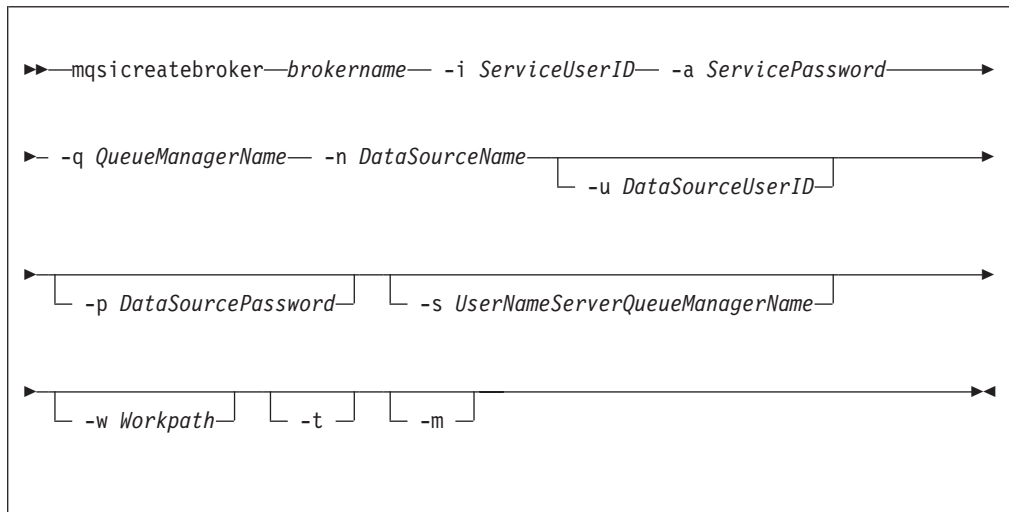
- Creates the broker-specific MQSeries queues, if these do not already exist.
- Creates database tables for the broker, if they do not already exist.
- On Windows NT, installs a service under which the broker will run.
- Creates a record for the component in the broker registry.

When you have created the broker, you must register it as a broker in the broker domain. You must do this by defining and deploying the broker using the *Topology* view in the Control Center. You must use the same broker name and the same queue manager name for both the create command and the Control Center definition.

You can also use the Command Assistant to issue this command.

mqsicreatebroker

Syntax



Required parameters

brokername

The name of the broker you want to create. This must be the first parameter. The broker name can be up to 242 characters in length. It is case sensitive on UNIX platforms. For restrictions on the character set that can be used, see “Rules for naming resources” on page 82.

-i ServiceUserID

The user ID under which the broker will run.

This can be specified in any valid username syntax. On Windows NT, these are:

- domain\username
- \\server\username
- .\username
- username

You should note that on UNIX systems, only the last format, username, is valid.

The ServiceUserID specified must be a member of the local group **mqbrkrs**. On Windows NT, it can be a direct or indirect member of the group. The ServiceUserID must also be authorized to access the home directory (where MQSeries Integrator has been installed), and the working directory (if specified by the -w flag).

If you specify, on Windows NT, that the broker is to run as an MQSeries trusted application (flag -t), you must also add this user ID to the group **mqm**. On UNIX platforms, the ServiceUserID itself must be specified as mqm if the -t flag is set.

The security requirements for the ServiceUserID are detailed in Table 2 on page 54.

Note: For Windows NT: If you use the unqualified form for this user ID (username), the operating system searches for the user ID throughout its domain, starting with the local system. This search might take some time to complete.

If you use this user ID for database access (that is, you do not specify a different user ID with the `-u` flag) and you are using SQL Server for your database, you must create this user ID as a SQL Server login ID and give it the correct access before you create the broker (see “Authorizing internal database access” on page 19 for further details). If your broker database exists in DB2, and this user ID is not known to DB2, DB2 automatically creates it for you.

-a *ServicePassword*

The password for the ServiceUserID. On UNIX platforms `-a` is required for Windows NT compatibility, but is not used in relation to ServiceUserID; it is only used as a default if `-p` is not specified. (See notes about the `-p` parameter for further details.)

-q *QueueManagerName*

The name of the queue manager associated with this broker. You are recommended to use the same name for your broker and the queue manager to simplify the organization and administration of your network. Note, however, that queue manager names are limited to 48 characters in length and are case sensitive.

If the queue manager does not already exist, it is created by this command. It is not created as the default queue manager: if you want this queue manager to be the default queue manager on this system, you must either create the queue manager before you issue this command, or use MQSeries Services to change the settings of this queue manager to make it the default.

The queue manager attribute MAXMSGL (maximum length of messages that can be put to queues) is updated to 100MB. This is done whether or not the queue manager is created by this command.

-n *DataSourceName*

The ODBC data source name (DSN) of the database in which the broker tables will be created. This must be the DSN, not the name of the database, if you have not used the same name for both.

This database must already exist. You must create a System DSN ODBC connection for this DSN, if you have not already done so. This task is described in “Defining internal MQSeries Integrator database connections (Windows NT only)” on page 18.

Optional parameters

-u *DataSourceUserID*

The user ID with which to access the broker database. If this is not specified, it defaults to the value specified by `-i`.

This user ID must have the authority to create tables within this database, and read from and write to those tables.

On Windows NT, if your broker database exists in DB2, and this user ID is not known to DB2, it is created for you within DB2. On UNIX platforms, the service user must have previously been granted the correct privilege. If your database is SQL Server, you must create this user ID as a SQL Server login ID and give it the correct access before you create the broker (see “Authorizing internal database access” on page 19 for further details).

If you have an application database in DB2 which was created by this user ID or to which this user ID has appropriate read, write or create authority, message flows executing in this broker will be able to access and manipulate the application data held within it without having to specify explicitly schema names.

mqsicreatebroker

-p *DataSourcePassword*

The password of the user ID with which the database is to be accessed. If not specified, this defaults to the ServicePassword specified by -a. For DB2 on UNIX platforms, -u and -p can be specified as empty strings (""). In this case, DB2 grants MQSeries Integrator the privileges of the MQSeries Integrator ServiceUserID which results in a database connection as "already verified". If you specify -a as an empty string as well as -u and -p, then no passwords are stored by MQSeries Integrator, creating the most secure configuration.

-s *UserNameServerQueueManagerName*

The name of the MQSeries queue manager that is associated with the User Name Server. If this is not specified, the broker assumes there is no User Name Server defined.

-w *WorkPath*

The directory in which working files for this broker are stored. If not specified, files are stored in the default workpath, specified when the product was installed.

-t Requests that the broker is configured to run as an MQSeries trusted application.

If you specify this option on Windows NT, you must add the service user ID (identified by flag -i) to the group **mqm**. On UNIX platforms, the ServiceUserID itself must be specified as **mqm** if this flag is set. For more details about using MQSeries trusted applications, see "Using MQSeries trusted applications" on page 49 and *MQSeries Intercommunication*.

-m Request migration of an existing MQSeries Publish/Subscribe broker. If you specify this option, the queue manager identified by -q must be the queue manager being used by the MQSeries Publish/Subscribe broker.

Authorization

On Windows NT, the user ID used to invoke this command must have **Administrator** authority on the local system.

On UNIX platforms, the user ID used to invoke this command must either be **root** or must be the same as that specified in the -i parameter. It must also be a member of the **mqbrkrs** group.

MQSeries queues created

SYSTEM.BROKER.ADMIN.QUEUE
SYSTEM.BROKER.CONTROL.QUEUE
SYSTEM.BROKER.EXECUTIONGROUP.QUEUE
SYSTEM.BROKER.EXECUTIONGROUP.REPLY
SYSTEM.BROKER.INTERBROKER.QUEUE
SYSTEM.BROKER.MODEL.QUEUE

Access authority is granted for the MQSeries Integrator group **mqbrkrs** to all these queues. If the DLQ is enabled, it will also have the same authority.

Database tables created

BACLENTRIES
BCLIENTUSER
BGROUPNAME
BLOGICALTOPHYSNAME
BMQEPUBDEST

```

|      BMQEPUBMSGIN
|      BMQEPUBMSGOUT
|      BMQESTDMSGIN
|      BMQESTDMSGOUT
|      BMQPSTOPOLOGY
|      BNBRCONNECTIONS
|      BPHYSICALFILE
|      BPUBLISHERS
|      BRETAINEDPUBS
|      BRMCONFIG
|      BROKERA
|      BROKERAEG
|      BROKERRESOURCES
|      BSCADADEST
|      BSCADAMSGIN
|      BSCADAMSGOUT
|      BSUBSCRIPTIONS
|      BTOPOLOGY
|      BUSERCONTEXT
|      BUSERMEMBERSHIP
|      BUSERNAME
|      BWFFRELATIONSHIP

```

Responses

This command returns the following responses:

- BIP8011 Unable to create configuration data
- BIP8012 Unable to connect to system components
- BIP8014 Component cannot be created
- BIP8022 Invalid user ID/password
- BIP8030 Unable to modify user ID privileges
- BIP8040 Unable to connect to database
- BIP8048 Unable to start queue manager
- BIP8050 Unable to create queue manager
- BIP8051 Unable to create queue
- BIP8053 Unable to set security for queue manager
- BIP8054 Unable to set security for queue
- BIP8056 Unknown queue manager
- BIP8070 Database exception
- BIP8072 Database exception
- BIP8073 Invalid broker name
- BIP8084 Unable to create directory
- BIP8086 Queue manager in use
- BIP8087 Component already exists
- BIP8093 Queue manager being created
- BIP8094 Queue manager stopping

Note: In some circumstances, you might see the following error message issued by DB2:

```

(51002) [IBM] [CLI Driver] [DB2/NT] SQL0805N
Package "NULLID.SQLLF000" was not found.  SQLSTATE=51002.

```

This error occurs when the bind to the database is not successful.

mqsicreatebroker

On Windows NT, binding is not needed for broker databases, but is required for user databases. For the database MYDB, for example, you can effect this by entering the following commands at the command prompt:

```
db2 connect to MYDB user db2admin using db2admin
db2 bind X:\sql1lib\bnd\@db2cli.lst grant public
db2 connect reset
```

where X: is the drive on which DB2 is installed.

On UNIX platforms, binding is necessary for all databases. For the database MQSIBKDB, for example, you can effect this by entering the following commands at the command prompt:

```
db2 connect to MQSIBKDB user db2admin using db2admin
db2 bind ~/sql1lib/bnd/@db2cli.lst grant public CLIPKG 5
db2 connect reset
```

Note: If you are not using the default DB2 user ID and password (db2admin) you must replace these values in the db2 connect command with the correct values.

Further information is provided at “Configuring databases for internal data” on page 15.

If you execute the **mqsicreatebroker** command for a second time because of a failure the first time, this will result in the production of a series of messages. These will indicate any items which cannot be created. There should not be any detrimental effects as a result of this. For example, as long as the reason for the first failure has been resolved, attempting to create a broker which failed the first time should result in a properly created broker.

Examples

```
mqsicreatebroker MQSI_SAMPLE_BROKER -i mqsuid -a mqsipw
-q MQSI_SAMPLE_QM -s MQSI_SAMPLE_UNQ_QM -n MQSIBKDB
mqsicreatebroker BROKERA -i mqsuid -a mqsipw -q BROKERA -n BRKA_DB -t
```

Related commands

“mqsichangebroker (Change broker)” on page 90

“mqsdeletebroker (Delete broker)” on page 120

mqsicreateconfigmgr (Create Configuration Manager)

Purpose

Use the **mqsicreateconfigmgr** command to create the Configuration Manager.

You should only use the Configuration Manager on a Windows NT platform. This command is therefore only applicable when using that operating system.

This command allows you to specify all the properties required to create the Configuration Manager.

This command performs the following actions:

- Creates an MQSeries queue manager, if one does not already exist.

Note: If this command creates an MQSeries queue manager, it will also enable the default dead-letter queue (DLQ) provided by MQSeries (SYSTEM.DEAD.LETTER.QUEUE). The security settings are the same as those of other Configuration Manager-specific MQSeries queues.

The **mqsideleteconfigmgr** command will not delete this queue (unless the Queue Manager is deleted).

- Starts the MQSeries queue manager, if this is not already running. The Configuration Manager always runs as an MQSeries trusted application.

Note: If the queue manager is created by this command, it is not started as a Windows NT service; it will therefore stop if you log off. To avoid this happening, you must either remain logged on, or you must change the start up status of the queue manager service (described in “Starting MQSeries queue managers as a Windows NT service” on page 24). (If you lock your workstation, the MQSeries queue manager does not stop).
- Creates the Configuration Manager-specific MQSeries queues and channel, if they do not already exist.
- Creates database tables for the Configuration Manager in the message repository, if they do not already exist.
- Creates database tables for the Configuration Manager in the configuration repository, if they do not already exist.
- Installs a Windows NT service, under which the Configuration Manager will run.
- Creates a record for the component in the broker registry.

This command does not start the listener on the queue manager. You must start this before you can use the Control Center. See “Connecting Control Center clients to the Configuration Manager (Windows NT only)” on page 21 for instructions on how you can start the listener.

You can also use the Command Assistant to issue this command.

Note: If you have installed VisualAge® for Java, and selected the MQSeries Connector as part of that installation, you must ensure that the CLASSPATH entry for VisualAge for Java appears *after* the CLASSPATH entries for MQSeries for Windows NT (server or Java client). This is to ensure that the Configuration Manager accesses the correct MQSeries classes not the VisualAge classes when it is started (by the **mqsisstart** command). If the

mqsicreateconfigmgr

Configuration Manager detects an error in this area, it will write message BIP1004 to the Windows NT Event log or UNIX syslog.

Syntax



Required parameters

`-i ServiceUserID`

The user ID under which the Windows NT service must run.

This can be specified in any valid Windows NT username syntax:

- domain\username
- \\server\username
- .\username
- username

The ServiceUserID specified must be a member (either direct or indirect) of the local group **mqbrkrs**, and must be authorized to access the home directory (where MQSeries Integrator has been installed), and the working directory (if specified by the `-w` flag). This ID must also be a member (either direct or indirect) of the local group **mqm** or of the local Windows NT **Administrators** group.

The security requirements for the ServiceUserID are detailed in Table 2 on page 54.

Note: If you use the unqualified form for this user ID (username), the operating system searches for the user ID throughout its domain, starting with the local system. This search might take some time to complete.

-a *ServicePassword*

The password for the ServiceUserID.

-q *QueueManagerName*

The name of the queue manager associated with the Configuration Manager.

If the queue manager does not already exist, it is created by this command. It is not created as the default queue manager: if you want this queue manager to be the default queue manager on this system, you must create the queue manager before you issue this command.

The queue manager attribute MAXMSGL (maximum length of messages that can be put to queues) is updated to 100MB. This is done whether or not the queue manager is created by this command.

-n *DataBaseName*

The name of the database you created to hold the configuration repository tables. This database is the configuration repository for the whole broker domain, and contains configuration information for all resources, as well as data internal to the Configuration Manager itself.

This database must already exist. You do not need to create an ODBC connection for this database because access is provided by JDBC.

The *MQSeries Integrator for Windows NT Installation Guide* contains details of how to create a database for all databases required by MQSeries Integrator.

-m *MRMDataSourceName*

The ODBC DSN name of the database created to hold the message repository tables. This must be the DSN name, not the name of the database, if you have not used the same name.

This database must already exist. You must create a System DSN ODBC connection for this data source name (DSN), if you have not already done so (see "Defining internal MQSeries Integrator database connections (Windows NT only)" on page 18 for a description of this task).

Optional parameters

-u *DataBaseUserID*

The user ID with which the configuration repository database is to be accessed. If this is not specified, the value set in ServiceUserID is used.

This user ID must have the authority to create tables in the database identified by the DataBaseName, and to read from and write to that database.

-p *DataSourcePassword*

The password of the user ID with which the configuration repository database is to be accessed. If not specified, this defaults to the ServicePassword specified by -a.

-e *MRMDataSourceUserID*

The user ID with which the message repository database is to be accessed. If this is not specified, the value set in ServiceUserID is used.

This user ID must have the authority to create tables in the database identified by the MRMDataSourceName, and to read from and write to that database.

-r *MRMDataSourcePassword*

The password for the user ID with which the message repository database is to be accessed. If this is not specified, the value set in ServicePassword is used.

mqsicreateconfigmgr

-d *SecurityDomainName*

The name of the Windows NT security domain. If this is not specified, it defaults to the system's local security domain. See "Chapter 4. Setting up security" on page 51 for more details about using security domains with MQSeries Integrator.

-s *UserNameServerQueueManagerName*

The name of the MQSeries queue manager that is associated with the User Name Server. If this is not specified, the Configuration Manager assumes there is no User Name Server defined, and will not attempt to communicate with one.

-w *WorkPath*

The directory in which working files for the Configuration Manager are stored. If not specified, the default directory specified when the product was installed is used.

Authorization

This command changes security privileges for the ServiceUserID; the user ID used to invoke this command must be a member of the Windows NT **Administrators** group on this local system.

MQSeries queues created

SYSTEM.BROKER.CONFIG.QUEUE
SYSTEM.BROKER.CONFIG.REPLY
SYSTEM.BROKER.ADMIN.REPLY
SYSTEM.BROKER.SECURITY.REPLY
SYSTEM.BROKER.MODEL.QUEUE

Access authority is granted for the MQSeries Integrator group **mqbrkrs** to all these queues. If the DLQ is enabled, it will also have the same authority.

Access authority is granted for the MQSeries Integrator groups **mqbrdevt**, **mqbrasgn**, **mqbrops**, and **mqbrtpic** to the queues SYSTEM.BROKER.CONFIG.QUEUE and SYSTEM.BROKER.CONFIG.REPLY.

MQSeries channels created

SYSTEM.BKR.CONFIG

Database tables created

The following tables are created in the configuration repository:

CBROKER
CBROKERCEG
CCICSADAPTER
CCMDFCTYPE
CCOLLECTIVE
CCOLLECTIVECBROKER
CDATACTXTFCTYPE
CDECFTYPE
CDELETED
CEG
CEGCMGFLOW
CEGCMGPROJECT
CITERFCTYPE
CLOG

CMSGFLOW
 CMSGPROJECT
 CNAVTYPE
 CNEIGHBOURS
 COUTSTANDING
 CPRIMITIVES
 CPROPERTIES
 CSRCADAPTER
 CSUBSCRIBE
 CTGTADAPTER
 CTOPIC
 CTOPICCTOPIC
 CTOPOLOGY
 CTRACE
 CUUIDLOCKS

The following tables are created in the message repository:

CATEGORY_MEMBER
 M_ATTRIBUTE
 M_CATEGORY
 M_CONST_DEF
 M_CONTEXT_TAG
 M_ELEMENT
 M_LENGTH
 M_MEMBER_ATTRIBUTE
 M_MESSAGE
 M_TRANSACTION
 M_TYPE
 M_VALID_VALUE
 MRM_USER
 MSG_CONTEXTTAG_MBR
 PLUGIN
 PROJECT
 PROJECT_DEP_MEMBER
 REG_PLUGIN_MEMBER
 REPOSITORY
 TRANSACTION_MEMBER
 TYPE_MEMBER
 USER_MEMBER
 VALID_VALUE_MEMBER

Responses

This command returns the following responses:

- BIP8011 Unable to create configuration data
- BIP8012 Unable to connect to system components
- BIP8014 Component cannot be created
- BIP8022 Invalid user ID/password
- BIP8030 Unable to modify user ID privileges
- BIP8048 Unable to start queue manager
- BIP8050 Unable to create queue manager
- BIP8051 Unable to create queue
- BIP8053 Unable to set security for queue manager
- BIP8054 Unable to set security for queue
- BIP8055 Unable to load Java class
- BIP8056 Unknown queue manager

mqsicreateconfigmgr

- BIP8074 Unable to create JVM
- BIP8075 Java exception
- BIP8076 Unable to set current directory
- BIP8077 Error initializing configuration data
- BIP8078 Error initializing configuration data
- BIP8084 Unable to create directory
- BIP8087 Component already exists
- BIP8093 Queue manager being created
- BIP8094 Queue manager stopping
- BIP8097 Unable to create Java object

Examples

```
mqsicreateconfigmgr -i mqsiuid -a mqsipw -q MQSI_SAMPLE_CONFIG_QM  
-n MQSICMDB -m MQSIMRDB
```

Related commands

“[mqsicchangeconfigmgr \(Change Configuration Manager\)](#)” on page 93

“[mqsideleteconfigmgr \(Delete Configuration Manager\)](#)” on page 122

mqsicreateusernameserver (Create User Name Server)

Purpose

Use the `mqsicreateusernameserver` command to create a new User Name Server.

This command allows you to specify all the properties required to create a User Name Server.

This command performs the following actions:

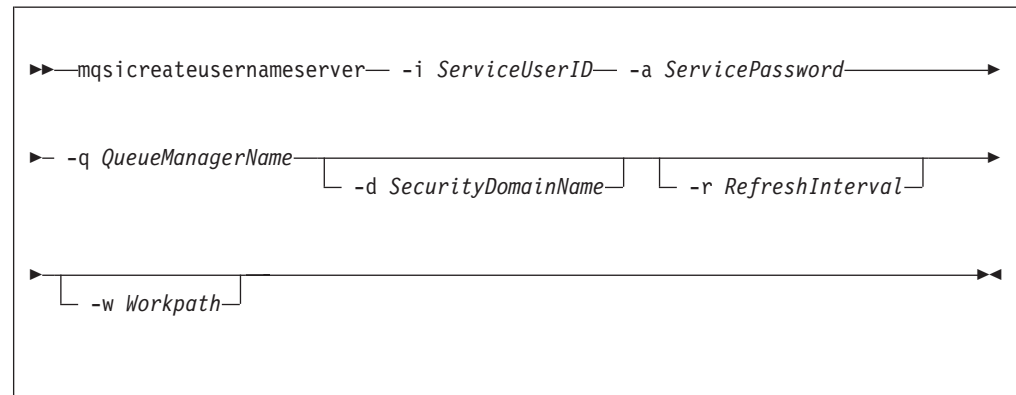
- Creates an MQSeries queue manager, if one does not already exist.
- Starts the MQSeries queue manager, if this is not already running.

Note: If the queue manager is started by this command on Windows NT, it is not started as a service. It will therefore stop if you log off. To avoid this happening, you must either remain logged on, or you must change the start up status of the queue manager service (described in “Starting MQSeries queue managers as a Windows NT service” on page 24). (If you lock your workstation, the queue manager does not stop).

- Creates the User Name Server-specific MQSeries queues, if these do not already exist.
- On Windows NT, installs a service under which the User Name Server will run.
- Creates a record for the component in the broker registry.

You can also use the Command Assistant to issue this command.

Syntax



Required parameters

-i *ServiceUserID*

The user ID under which the broker will run.

This can be specified in any valid username syntax. On Windows NT, these are:

- domain\username
- \\server\username
- .\username
- username

You should note that on UNIX systems, only the last format, username, is valid.

mqsicreateusernameserver

The ServiceUserID specified must be a member of the local group **mqbrkrs**. On Windows NT, it can be a direct or indirect member of the group. The ServiceUserID must also be authorized to access the home directory (where MQSeries Integrator has been installed).

The security requirements for the ServiceUserID are detailed in Table 2 on page 54 for Windows NT platforms and in Table 3 on page 62 for UNIX platforms.

Note: For Windows NT: If you use the unqualified form for this user ID (username), the operating system searches for the user ID throughout its domain, starting with the local system. This search might take some time to complete.

-a *ServicePassword*

The password for the ServiceUserID.

-q *QueueManagerName*

The name of the queue manager associated with the User Name Server.

If the queue manager does not already exist, it is created by this command. It is not created as the default queue manager: if you want this queue manager to be the default queue manager on this system, you must create the queue manager before you issue this command.

The queue manager attribute MAXMSGL (maximum length of messages that can be put to queues) is updated to 100MB. This is done whether or not the queue manager is created by this command.

Optional parameters

-d *SecurityDomainName*

The name of the Windows NT security domain. If this is not specified, it defaults to the system's local Windows NT security domain. For more details about the implementation of security in MQSeries Integrator, see "Chapter 4. Setting up security" on page 51.

-r *RefreshInterval*

The interval, specified in seconds, at which the User Name Server interrogates the security subsystem for changes to user or group attributes. If it is not specified, the User Name Server's default interval of 60 seconds is used.

-w *WorkPath*

The directory in which working files for the User Name Server are stored. If not specified, the default value specified when the product was installed is used.

Authorization

On Windows NT, the user ID used to invoke this command must have **Administrator** authority on the local system.

On UNIX platforms, the user ID used to invoke this command must either be **root** or must be the same as that specified in the **-i** parameter. It must also be a member of the **mqbrkrs** group.

MQSeries queues created

SYSTEM.BROKER.SECURITY.QUEUE
SYSTEM.BROKER.MODEL.QUEUE

Access authority has been granted for the MQSeries Integrator group **mqbrkrs** to all these queues.

Responses

This command returns the following responses:

- BIP8011 Unable to create configuration data
- BIP8012 Unable to connect to system components
- BIP8014 Component cannot be created
- BIP8022 Invalid user ID/password
- BIP8030 Unable to modify user ID privileges
- BIP8048 Unable to start queue manager
- BIP8050 Unable to create queue manager
- BIP8051 Unable to create queue
- BIP8053 Unable to set security for queue manager
- BIP8054 Unable to set security for queue
- BIP8056 Unknown queue manager
- BIP8068 Integer argument required
- BIP8084 Unable to create directory
- BIP8087 Component already exists
- BIP8093 Queue manager being created
- BIP8094 Queue manager stopping

Examples

```
mqsicreateusernameeserver -i mqsiuid -a mqsipw  
-q MQSI_SAMPLE_QM -r 1000
```

Related commands

- “[mqsichangeusernameeserver \(Change User Name Server\)](#)” on page 99
- “[mqsdeleteusernameeserver \(Delete User Name Server\)](#)” on page 125

mqsdeletebroker (Delete broker)

Purpose

Use the **mqsdeletebroker** command to delete a named broker, the queues on its local queue manager (created when the broker was created), and its data in the broker database. You can also specify that the queue manager is to be deleted.

Note, even if all broker data is deleted and the broker database tables are empty, the broker tables are not removed from the database.

When you delete a broker, you must implement a set of related tasks in a specific order to ensure the integrity of your network. For more details about deleting components from your configuration, see “Deleting components from the broker domain” on page 40. **The following tasks must be completed:**

- Remove the broker from the configuration repository by deleting it from the Control Center **Topology** view.
- Deploy the delta configuration (all types) from the File menu on the **Topology** view. This removes the broker configuration data from the configuration repository.
- Check that the deployment has been successful (use the **Log** view).
- Stop the broker, using **mqsistop**.
- Delete the broker, using **mqsdeletebroker**.

For more details of the tasks invoked through the Control Center, see *MQSeries Integrator Using the Control Center*.

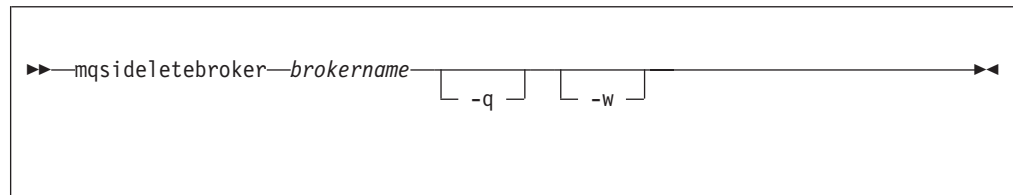
This command performs the following actions:

- On Windows NT, stops the service that runs the broker.
- Deletes the broker’s queues on the queue manager.
- Stops and deletes the broker’s queue manager, if requested.
- Removes the broker’s data from the database.
- Removes the record for the component in the broker registry.

You can also use the Command Assistant to issue this command.

Note: If you delete a broker that has MQSeries Publish/Subscribe broker neighbors, you must also invoke the command **clrmqbrk** at each of these neighbors, specifying the MQSeries Integrator broker that you are deleting with this command. See “Deleting brokers in a heterogeneous network” on page 169 for more details about deleting brokers.

Syntax



Required parameters

brokername

The name of the broker you want to delete.

Optional parameters

-q Specifies that the broker's queue manager will be deleted. (If this option is not specified, only the MQSeries Integrator queues and broker's data are deleted.)

If the queue manager hosts another component (the Configuration Manager, or the User Name Server, or both in addition to this broker) which still exists, this command will fail.

-w Causes all files related to this broker to be deleted from the workpath.

Authorization

On Windows NT, the user ID used to invoke this command must have **Administrator** authority on the local system.

On UNIX platforms, the user ID used to invoke this command must either be **root** or must be the same as that specified in the **-i** parameter when the broker was created. It must also be a member of the **mqbrkrs** group.

Responses

- BIP8012 Unable to connect to system components
- BIP8013 Component does not exist
- BIP8017 Component cannot be deleted
- BIP8018 Component running
- BIP8040 Unable to connect to database
- BIP8048 Unable to start queue manager
- BIP8049 Unable to stop queue manager
- BIP8052 Unable to delete queue
- BIP8073 Invalid broker name
- BIP8082 Unable to delete queue manager
- BIP8093 Queue manager being created
- BIP8094 Queue manager stopping
- BIP8095 Queue manager reserved

Examples

```
mqsideletebroker MQSI_SAMPLE_BROKER -q
```

Related commands

"mqsicreatebroker (Create broker)" on page 105

"mqsichangebroker (Change broker)" on page 90

mqsideleteconfigmgr (Delete Configuration Manager)

Purpose

Use the **mqsideleteconfigmgr** command to delete the Configuration Manager and the queues on its local queue manager (created when the Configuration Manager was created). You can also specify that database tables in the configuration repository and the message repository are deleted.

You should only use the Configuration Manager on a Windows NT platform. This command is therefore only applicable when using that operating system.

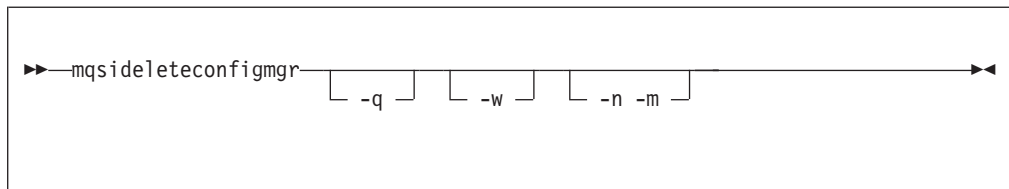
You must stop the Configuration Manager, using **mqsistop**, before you can delete it.

This command performs the following actions:

- Stops the Windows NT service that runs the Configuration Manager.
- Deletes the Configuration Manager's queue manager.
- Stops and deletes the Configuration Manager's queue manager, if requested.
- Removes the tables from the configuration repository and the message repository, if requested.
- Removes the record for the component in the broker registry.

You can also use the Command Assistant to issue this command.

Syntax



Optional parameters

-q Specifies that the Configuration Manager's queue manager will be deleted. (If this option is not specified, only the MQSeries Integrator queues are deleted.)

If the queue manager hosts another component (a broker or the User Name Server, or both) which still exists, this command will fail.

-w Causes all files related to the Configuration Manager to be deleted from the workpath.

-n -m

Causes the configuration repository and the message repository to be deleted.

The two parameters **-n** and **-m** must be specified together. If they are, the tables in the configuration repository and the message repository are deleted.

Warning

You must be very careful if you choose to specify this option. The configuration repository and message repository contain the configuration data for the whole broker domain, not just data internal to the Configuration Manager itself. Deleting these repositories therefore destroys all information pertinent to the broker domain, and requires you to recreate every resource within it to recover the broker domain.

If you want to change the user ID that you used to create the database tables for the configuration repository, or the message repository, or both, you must:

- Stop the Configuration Manager.
- Delete the Configuration Manager, but do not specify these flags. The data is preserved in the existing tables.
- Create the Configuration Manager. Specify the new user ID as the user ID you want to use for database access. You can specify a unique user ID for configuration repository access (flag `-u`), or for the message repository access (flag `-e`), or both, or you can specify the ID as the service user ID (flag `-i`) and allow database access to default to that ID. Check “mqsicreateconfigmgr (Create Configuration Manager)” on page 111 for details.
- Copy the old database tables to the new tables.
- Restart the Configuration Manager.
- When you are confident that the transfer is complete, you can delete the old tables.

If you specify one of these flags, but not the other, the command returns an error. If neither flag is specified, the tables remain intact in the database.

Authorization

The user ID used to invoke this command must have Windows NT **Administrator** authority.

Responses

- BIP8012 Unable to connect to system components
- BIP8013 Component does not exist
- BIP8017 Component cannot be deleted
- BIP8018 Component running
- BIP8038 Unsupported command option
- BIP8048 Unable to start queue manager
- BIP8049 Unable to stop queue manager
- BIP8052 Unable to delete queue
- BIP8055 Unable to load Java class
- BIP8074 Unable to create JVM
- BIP8075 Java exception
- BIP8076 Unable to set current directory
- BIP8077 Error initializing configuration data
- BIP8078 Error initializing configuration data
- BIP8082 Unable to delete queue manager
- BIP8093 Queue manager being created
- BIP8094 Queue manager stopping
- BIP8095 Queue manager reserved
- BIP8097 Unable to create Java object

mqsdeleteconfigmgr

Examples

```
mqsdeleteconfigmgr -q
```

Related commands

“mqscreateconfigmgr (Create Configuration Manager)” on page 111

“mqschangeconfigmgr (Change Configuration Manager)” on page 93

mqsideleteusernameeserver (Delete User Name Server)

Purpose

Use the **mqsideleteusernameeserver** command to delete the User Name Server and the queues on its local queue manager (created when the User Name Server was created).

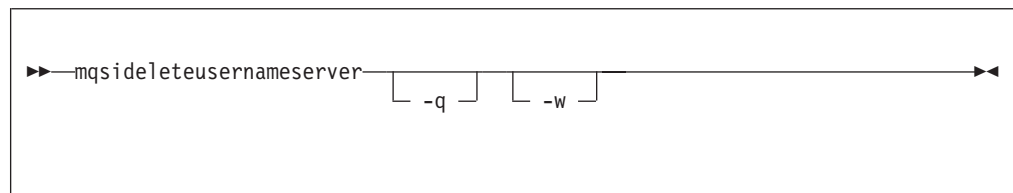
You must stop the User Name Server, using **mqsisstop**, before you can delete it.

This command performs the following actions:

- On Windows NT, stops the service that runs the User Name Server.
- Deletes the User Name Server's queues on the queue manager.
- Stops and deletes the User Name Server's queue manager, if requested.
- Removes the record for the component in the broker registry.

You can also use the Command Assistant to issue this command.

Syntax



Optional parameters

-q Specifies that the User Name Server's queue manager will be deleted when the User Name Server has been deleted. (If this option is not specified, only the MQSeries Integrator queues are deleted.)

If the queue manager hosts another component (a broker or the Configuration Manager, or both) which still exists, this command will fail.

-w Causes all files related to the User Name Server to be deleted from the workpath.

Authorization

On Windows NT, the user ID used to invoke this command must have **Administrator** authority on the local system.

On UNIX platforms, the user ID used to invoke this command must either be **root** or must be the same as that specified in the **-i** parameter when the User Name Server was created. It must also be a member of the **mqbrkrs** group.

Responses

- BIP8012 Unable to connect to system components
- BIP8013 Component does not exist
- BIP8017 Component cannot be deleted
- BIP8018 Component running
- BIP8048 Unable to start queue manager
- BIP8049 Unable to stop queue manager

mqsdeleteusernameserver

- BIP8052 Unable to delete queue
- BIP8082 Unable to delete queue manager
- BIP8093 Queue manager being created
- BIP8094 Queue manager stopping
- BIP8095 Queue manager reserved

Examples

mqsdeleteusernameserver

Related commands

“mqscreateusernameserver (Create User Name Server)” on page 117

“mqschangeusernameserver (Change User Name Server)” on page 99

mqsiformatlog (Format log)

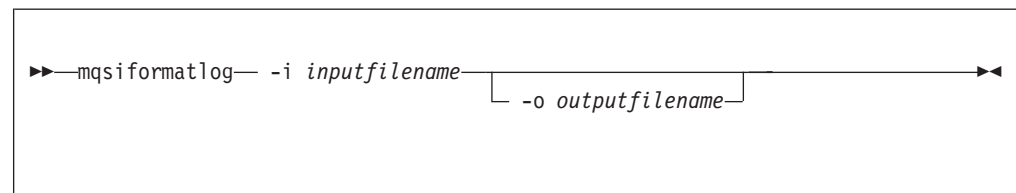
Purpose

Use the **mqsiformatlog** command to process the XML log created by **mqsireadlog** into a formatted content, and retrieve and format any messages that the XML log contains into a form suitable for the locale of the user invoking the command.

You can specify the output to be directed to file, or to the command shell.

For more information on using this command, and examples of its use, see “Chapter 5. Problem determination” on page 65.

Syntax



Required parameters

-i *inputfilename*

The filename of the XML log file that is to be formatted.

Optional parameters

-o *outputfilename*

The filename of the file into which the formatted log output is to be written. If this is not specified, the formatted log data is written to stdout.

Authorization

The user ID used to invoke this command must have read access to the input file, and write access to the output file.

Responses

- BIP8041 Unable to open file
- BIP8042 Insufficient memory
- BIP8043 Invalid trace file
- BIP8046 Unable to initialize XML
- BIP8047 Unable to parse data
- BIP8069 Unable to find message
- BIP8079 Unable to locate XML function
- BIP8080 Unable to load resource
- BIP8081 Error processing command

Examples

```
mqsiformatlog -i trace.xml -o formattrace.log
```

See the example of formatted normal level user trace in “Formatting user trace information” on page 70.

mqsiformatlog

Related commands

- “mqsichangetrace (Change trace settings)” on page 95
- “mqsireadlog (Read log)” on page 142
- “mqsireporttrace (Report trace settings)” on page 146

mqsiimpexpmsgset (Import/Export message set)

Purpose

Use the **mqsiimpexpmsgset** command to export a complete message set from the message repository into an XML-format file, or to import a complete message set from an XML-format file into the message repository. This allows message sets to be exported from one MQSeries Integrator broker domain and imported into a different broker domain.

When you have successfully imported a message set, you must stop the Control Center and the Configuration Manager and restart them. Your new message set will then be available to add to your workspace.

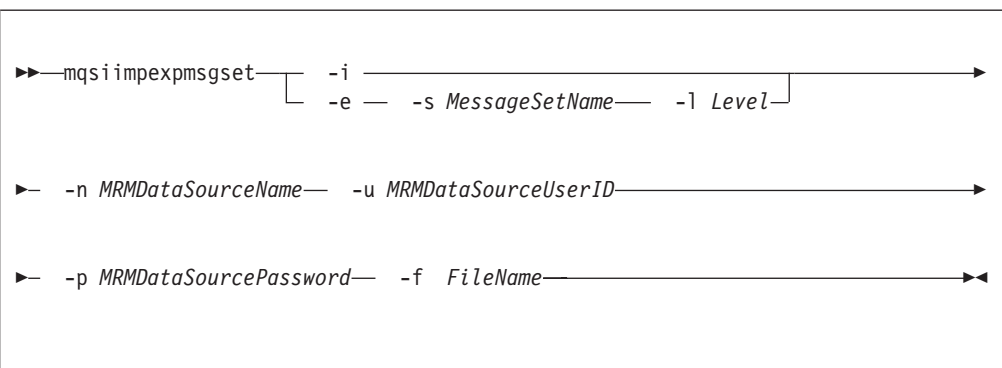
When you import or export a new message set, it is placed in *Frozen* and *Unlocked* state.

This command is only applicable when using the Windows NT operating system.

Note: The command 'mqsimrmimpexp' is deprecated.

The **mqsiimpexpmsgset** command replaces **mqsimrmimpexp**, used in previous versions of MQSeries Integrator. You should use this new command in preference to the old version, which is no longer supported. Unlike the old command, parameters now follow the normal convention and are *not* positional.

Syntax



Required parameters

- i Either specify this flag to initiate the import of message sets, or...
- e specify this flag to initiate the export of message sets. If you use this flag, you must specify both -s and -l:
- s *SourceMessageSetName*
The name (not the identifier) of the message set to be exported. This must be specified if you use -e.
- l *SourceMessageSetLevel*
The level of the message set to be exported. This must be specified if you use -e.

mqsiiimpexpmsgset

-n *MRMDataSourceName*

The ODBC data source name (DSN) of the database created to hold the message repository tables.

This database must already exist. You must create a System DSN ODBC connection for this DSN, if you have not already done so.

The *MQSeries Integrator Installation Guide* for your computer platform contains details of how to create a database and an ODBC connection for all databases required by MQSeries Integrator.

-u *MRMDataSourceUserID*

The user ID with which the message repository database is to be accessed. This user ID must have the authority to read from (export) and write to (import) the database identified by *MRMDataSourceName*.

-p *MRMDataSourcePassword*

The password for the user ID with which the message repository database is to be accessed.

-f *FileName*

The name of the XML file to be used as output for an export, or as input for an import.

Authorization

None.

Examples

Import message set:

```
mqsiiimpexpmsgset -i -n MQSIMRDB -u mqsiiuid -p mqsiiPW -f acctpay.mrp
```

Export message set:

```
mqsiiimpexpmsgset -e -s "Accounts Payable" -l 1 -n MQSIMRDB  
-u mqsiiuid -p mqsiiPW -f acctpay.mrp
```

Related commands

“mqsicopymsgset (Copy message set)” on page 103

“mqsinrfreload (Reload NEON messages)” on page 140

mqsijoinmqpubsub (Join broker to MQSeries Publish/Subscribe parent broker)

Purpose

Use the **mqsijoinmqpubsub** command to join this MQSeries Integrator broker to an MQSeries Publish/Subscribe broker network. The command identifies a specific MQSeries Publish/Subscribe broker that will be the parent of the MQSeries Integrator broker.

This is an asynchronous command. Successful completion of this command indicates that the MQSeries Integrator broker has accepted the request, not that the required action has completed.

Use the **mqsilistmqpubsub** command (“mqsilistmqpubsub (List MQSeries Publish/Subscribe neighbor broker status)” on page 137) to monitor the status of the asynchronous actions that result from this command.

Only use this command if you are integrating this MQSeries Integrator broker with an MQSeries Publish/Subscribe broker network. Before you issue this command, you must ensure that the MQSeries Integrator broker is ready to receive and process messages on queue SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS (that is, you must have restarted the broker after creating this queue. See “Creating and operating a heterogeneous network” on page 163 for more details.)

Syntax

```
►► mqsijoinmqpubsub—brokername— -p ParentQueueManagerName—►►
```

Required parameters

brokername

The name of the broker that is to be joined to an MQSeries Publish/Subscribe broker.

-p *ParentQueueManagerName*

The name of the queue manager that hosts the MQSeries Publish/Subscribe broker to which this MQSeries Integrator broker is to be joined.

Authorization

The user ID used to invoke this command must have put and inq authority to the queue SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS.

Responses

- BIP8013 Component does not exist
- BIP8056 Unknown queue manager
- BIP8057 Queue manager error
- BIP8059 Queue manager not available
- BIP8060 Queue error

mqsijoinmqpubsub

- BIP8061 No reply received
- BIP8064 Internal broker error
- BIP8066 Invalid broker name

Examples

```
mqsijoinmqpubsub MQSI_SAMPLE_BROKER -p MQBroker1
```

Related commands

“mqsiclearmqpubsub (Remove MQSeries Publish/Subscribe broker as a neighbor)” on page 101

“mqsilistmqpubsub (List MQSeries Publish/Subscribe neighbor broker status)” on page 137

mqsilcc (Start Control Center trace)

Purpose

Use the **mqsilcc** command to start the Control Center with service trace active at the specified level. When you invoke the Control Center from the MQSeries Integrator program folder (or the Start menu), it is invoked without trace (that is, with a trace level of 0 (none)). To activate its tracing, you must start the Control Center using this command. This command also activates tracing of the MQSeries Client for Java.

The Control Center is only available on a Windows NT platform. This command is therefore only applicable when using that operating system.

You are recommended to use this command only when you are instructed to do so by an MQSeries Integrator error message, or when directed to do so by your IBM Support Center.

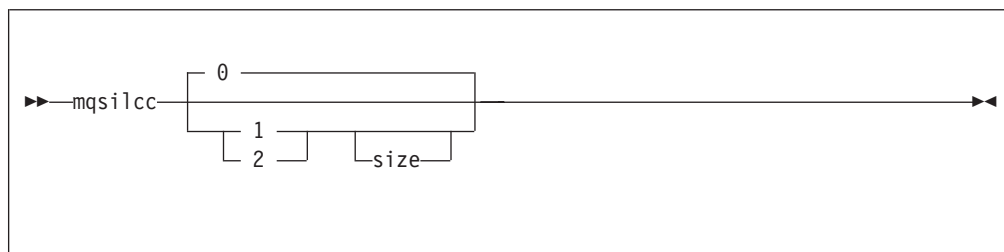
The trace output generated by these commands is written to trace files in the log subdirectory. These files are named Control.Center.trace.bin.x, where x is the level of trace being used (see below). When you have completed the work you want to trace, you can use **mqsireadlog** to retrieve the log as an 'XML format' file. You can use either **mqsiformatlog** (to produce a formatted file) or an XML browser to view the XML records.

When you set tracing on, you are causing additional processing to be executed for every activity in the component you are tracing. You must therefore expect to see an impact on performance when trace is active.

You must invoke this command from the <mqs_i_root>\Tool subdirectory.

The parameters on this command are positional.

Syntax



Optional parameters

0, 1 or 2

The level of trace you want to start. This can be:

- 0 This value requests that no tracing is done (the level is none).
- 1 This value requests a normal level of tracing. Tracing for the MQSeries Client for Java is started at level 2.
- 2 This value requests a debug level of tracing. Tracing for the MQSeries Client for Java is started at level 5.

mqsilcc

For further information about tracing for the MQSeries Client for Java, see the *MQSeries Clients* book.

size

The size of the trace file in KB (kilobytes). This is initially set to 4096 (file size 4MB). The maximum you can specify for size is 2097151 which creates a 2GB (gigabyte) trace file.

If you do not specify this parameter, the size is unchanged.

If the trace file reaches maximum size when new trace records are written, the trace file wraps and the new entries overwrite the oldest existing entries.

Authorization

None. However, the actions you can take in the Control Center depend on the authority of the Control Center user.

Responses

No additional responses are returned.

Examples

To set normal tracing on, with a default trace file size, enter

```
mqsilcc 1
```

To set tracing off, enter

```
mqsilcc
```

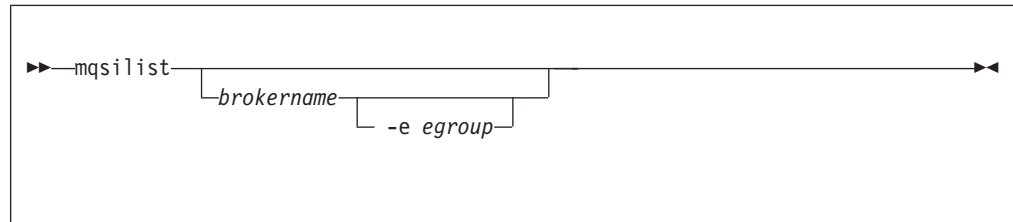
mqsilist (List resources)

Purpose

Use the **mqsilist** command to list all of the components installed on this system, all the execution groups defined to a specific broker, or all the message flows contained in a named execution group on a named broker.

The output is directed to stdout.

Syntax



If you do not specify any parameters when you issue this command, a list of components and queue manager names is displayed for each component created on this system, in the form:

```

BIP8099I: brokername - queuemanagename
BIP8099I: ConfigMgr - queuemanagename
BIP8099I: UserNameServer - queuemanagename
BIP8071I: Successful command completion
  
```

Optional parameters

brokername

The name of the broker you want resources listed for. This must be a deployed broker. A list of execution groups configured on this broker and the process ID (pid) of each is displayed.

-e egroup

Selects an execution group within a broker. You must specify the label of the execution group for which you want to list message flows. The command returns a list of message flows assigned to the specified execution group within the broker.

The broker specified must be active for any message flow information to be returned.

Authorization

If you have specified a broker name and the **-e** flag, the user ID used to invoke this command must have **mqbrkrs** group membership.

mqsilist

Responses

- BIP8013 Component does not exist
- BIP8020 Unable to access database
- BIP8029 Broker not configured
- BIP8038 Unsupported command option
- BIP8039 Execution group not available
- BIP8040 Unable to connect to database

Examples

```
mqsilist MQSI_SAMPLE_BROKER -e DefaultEG
```


mqsilistmqpubsub (List MQSeries Publish/Subscribe neighbor broker status)

Purpose

Use the **mqsilistmqpubsub** command to display the status of the MQSeries Publish/Subscribe neighbor brokers to the specified MQSeries Integrator broker.

This command indicates the status of the activity started by a previous join request (see “mqsjoinmqpubsub (Join broker to MQSeries Publish/Subscribe parent broker)” on page 131). The command reports on the status of each neighbor broker, which can be:

- Active. Broker status is active if the join request has completed successfully.
- Inactive. Broker status is inactive if the join has been initiated, but has not completed.

This command also shows the streams that are recognized by both the MQSeries Integrator broker and its neighbor (on which messages can be published and distributed between the brokers). Stream information is only provided for neighbors with *active* status.

Only use this command if you are integrating with or migrating from an MQSeries Publish/Subscribe broker network.

The output generated by this command is directed to stdout.

Syntax

```
►►mqsilistmqpubsub—brokername—◄◄
```

Required parameters

brokername

The name of the broker for which you want a list of neighbors.

Authorization

None.

Responses

- BIP8013 Component does not exist
- BIP8020 Unable to access database
- BIP8029 Broker not configured
- BIP8040 Unable to connect to database
- BIP8064 Internal broker error
- BIP8070 Database exception
- BIP8072 Database exception

mqsilistmqpubsub

Examples

If there are no MQSeries Publish/Subscribe brokers, and no **mqsijoinmqpubsub** command has been issued, this command returns the following message:

```
BIP8088I: There are no MQSeries Publish/Subscribe neighbors
```

If an **mqsijoinmqpubsub** command has been issued, one of two response messages is displayed:

- For every broker that is an inactive neighbor of MQSI_SAMPLE_BROKER, (that is, an **mqsijoinmqpubsub** command has been successfully initiated either by an **mqsijoinmqpubsub** or an **strmqbrk -p** command, but negotiations for common streams are still in progress), the following message is displayed:

```
BIP8089I: MQSeries Publish/Subscribe neighbor <brokername> is inactive.
```

- For every broker that is an active neighbor of MQSI_SAMPLE_BROKER, (that is, the two brokers are exchanging publications and subscriptions for each of the common streams), the following message is displayed:

```
BIP8090I: MQSeries Publish/Subscribe neighbor <brokername> is active.
```

Additional messages are displayed for active brokers to indicate the common streams for which publications and subscriptions are exchanged, in the following form:

```
BIP8091I: Common stream streamname
```

For example,

```
mqsilistmqpubsub MQSI_SAMPLE_BROKER
```

might return the following responses:

```
BIP8090I: MQSeries Publish/Subscribe neighbor MQPS_BROKER_1 is active.  
BIP8091I: Common stream SYSTEM.BROKER.DEFAULT.STREAM.  
BIP8091I: Common stream STREAM0.  
BIP8090I: MQSeries Publish/Subscribe neighbor MQPS_BROKER_2 is active.  
BIP8091I: Common stream SYSTEM.BROKER.DEFAULT.STREAM.  
BIP8091I: Common stream STREAM150.  
BIP8089I: MQSeries Publish/Subscribe neighbor MQPS_BROKER_3 is inactive.
```

In this example, the MQSeries Integrator broker has three MQSeries Publish/Subscribe neighbors. Two of these neighbors are active and have been successfully joined to the MQSeries Integrator broker. The third is inactive and is in the process of being joined.

The list of streams that are common to the MQSeries Integrator broker and the two active MQSeries Publish/Subscribe brokers are included in the response. For MQPS_BROKER_1, the streams SYSTEM.BROKER.DEFAULT.STREAM and STREAM0 are common. For MQPS_BROKER_2, the streams SYSTEM.BROKER.DEFAULT.STREAM and STREAM150 are common.

If a neighbor is inactive for a long period of time, it is likely that the communication link between the two brokers has been broken. You must ensure that the MQSeries connections between the two brokers (channels and transmission queues) are running, and that the MQSeries Integrator and MQSeries Publish/Subscribe brokers are both active.

Related commands

“mqsiclearmqpubsub (Remove MQSeries Publish/Subscribe broker as a neighbor)” on page 101

“mqsijoinmqpubsub (Join broker to MQSeries Publish/Subscribe parent broker)” on page 131

mqsinrfreload (Reload NEON messages)

Purpose

Note: This command is deprecated.

The **mqsinrfreload** command is deprecated in this version of MQSeries Integrator as it only relates to the old NEON functionality (using the deprecated NEONRules and NEONFormatter nodes, and the NEON domain message parser).

This command does *not* work with the new NEON functionality (NEONRulesEvaluation, NEONMap and NEONTransform nodes, and the new NEONmsg domain message parser). To refresh these, you should stop and restart the broker.

Even if you are using only the old NEON functionality, the recommended MQSeries Integrator Version 2.0.2 procedure for refreshing the rules and formats is to stop and restart the broker.

Use the **mqsinrfreload** command to force the broker to re-access the NEONFormatter and NEONRules database. You must use this command if you make any changes to the database containing your NEONFormatter and NEONRules definitions. You must issue it for each broker that needs access to this database.

Make sure that all message flows that use the NEON parser, or the NEONRules and NEONFormatter nodes are stopped before attempting to use **mqsinrfreload**. This can be done using the Operations view in the Control Center.

Syntax

```
►►mqsinrfreload— -b brokername◄◄
```

Required parameters

-b *brokername*

The name of the broker that must re-access the NEONFormatter and NEONRules database. A message is sent to this broker that instructs the broker to re-access the database.

The name of the broker in this command is case sensitive. It must be specified in the same way it was when created using **mqsicreatebroker**.

Authorization

The user ID used to issue the command must be a member of the group **mqbrkrs**.

Responses

- BIP8013 Component does not exist

Examples

```
mqsinrfreload -b broker1
```

Related commands

“mqsicopymsgset (Copy message set)” on page 103

“mqsimpexpmsgset (Import/Export message set)” on page 129

mqsireadlog (Read log)

Purpose

Use the **mqsireadlog** command to retrieve the trace log for the component specified. This command is valid for:

- User trace. Specify the **-u** option.
- Service trace. Specify the **-t** option. You are recommended to use this option only if directed to do so by the action described in a BIPxxxx message, or by your IBM Support Center.

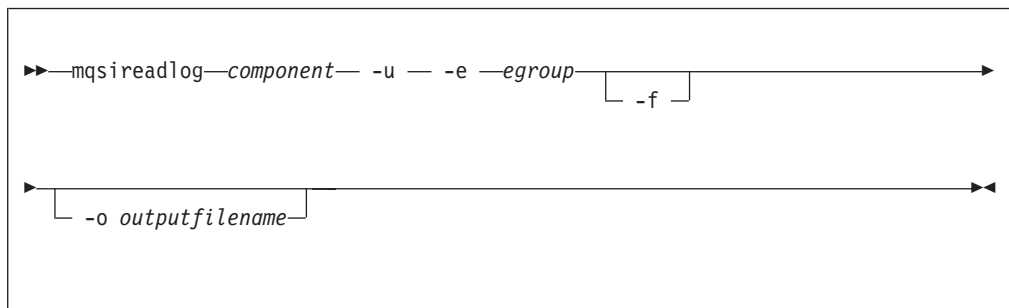
The trace records returned by this command are in XML format and can be browsed with an XML browser or formatted into a flat file using **mqsiformatlog**.

You can specify the output to be directed to file, or to stdout.

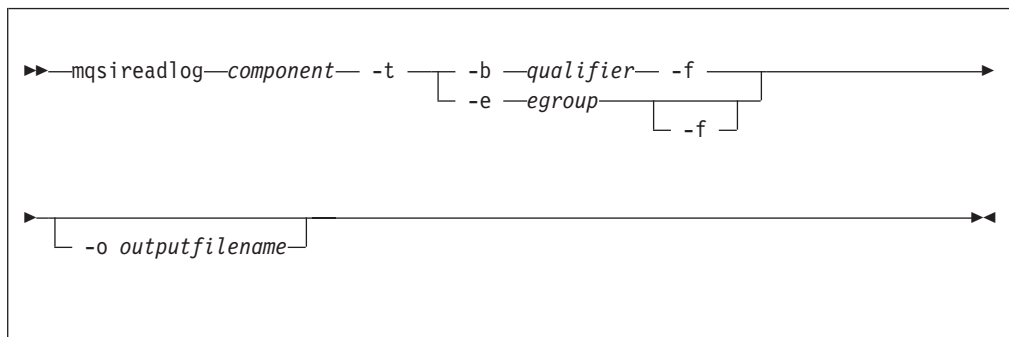
If you specify a broker, or any of its resources, (execution group or message flow), you must have deployed them before you can start trace and read the log files.

For more information on using this command, and examples of its use, see “Chapter 5. Problem determination” on page 65.

Syntax - user trace



Syntax - service trace



Required parameters

component

The name of the component for which the log is to be read. This can be either a broker name, or the fixed values `ConfigMgr`, `UserNameServer`, or `ControlCenter` (all are case sensitive on UNIX platforms).

Optional parameters

- u Read the log contents from the user trace log. This is only valid if you select the broker component.
- e *egroup*
Specify the label of the execution group for which log information is to be read.
- o *filename*
The name of the file into which the log data is to be written. If you specify a full pathname, the file is created in the directory specified. If you specify just the filename, the file is created in the current working directory. You must specify a file name if you want to format the log using **mqsiformatlog**. If you do not specify a filename, the contents of the log are written to stdout. You are recommended to use a file extension of `.xml`.
- f Read the log file directly from the file system. If you do not specify this option, the command sends an XML message to the component to request the log contents. If you have specified `-t` (service trace) then, in general, you must specify this flag as well. Further details are given in “Additional parameters exclusive to service trace”.

If you specify this option, you are recommended to stop tracing (using **mqsichangetrace**) before you use the **mqsireadlog** command. If the log file is in use when you issue this command with this flag specified, partial XML records might be returned. You can reduce the risk of this happening by specifying `-m safe` on the **mqsichangetrace** command. If the component being traced has itself stopped, you do not then need to issue a **mqsichangetrace** command.

If you do not stop tracing before you issue this command, you are recommended to check the contents of the log file created and remove any partial records from the end using a text editor before using the **mqsiformatlog** command, as partial records cannot be read by the format command.

Additional parameters exclusive to service trace

You are recommended to use these options only when directed to do so by your IBM Support Center or by a BIPxxxx message.

- t Read the log contents from the service trace log.
- b *qualifier*
Read the contents of the log for the broker agent, Configuration Manager agent, or User Name Server agent, or for the specified command utility program. This option is only valid if you have specified `-t` (service trace).
Table 5 on page 144 shows the valid combinations of *qualifier* and component for service trace.

You must enter these values exactly as shown.

The agent trace is initiated when you specify the `-b` flag on the **mqsichangetrace** command. You are recommended to do this only when directed to do so by an MQSeries Integrator error message or when instructed to do so by your IBM Support Center.

- f Read the log file directly from the file system. When used with service trace, this flag has the same characteristics as when used with user trace. It remains optional if the `-e` is specified. If the `-b` flag is used, then the `-f` flag must be specified.

mqsireadlog

Table 5. Service trace: qualifiers valid with components

Qualifier (below)	Component= Broker	Component= ConfigMgr	Component= UserNameServer	Component= ControlCenter
mqsichangebroker	✓			
mqsichangeconfigmgr		✓		
mqsichangetrace	✓	✓	✓	
mqsichangeusernameserver			✓	
mqsiclearmqpubsub	✓			
mqsicreatebroker	✓			
mqsicreateconfigmgr		✓		
mqsicreateusernameserver			✓	
mqsideletebroker	✓			
mqsideleteconfigmgr		✓		
mqsideleteusernameserver			✓	
mqsiformatlog ¹	✓	✓	✓	
mqsijoinmqpubsub	✓			
mqsilist ²	✓	✓	✓	
mqsilistmqpubsub	✓			
mqsinrfreload	✓			
mqsireadlog	✓	✓	✓	
mqsireporttrace	✓		✓	
mqsistart	✓	✓	✓	
mqsistop	✓	✓	✓	
agent	✓	✓	✓	
ControlCenter				✓

Note:

1. Because this command does not have a component parameter, trace information is recorded in, and will be retrieved from, the *utility* component trace files. For further details see “Optional traces” on page 67.
2. If this command has been invoked without a component, trace information is recorded in, and will be retrieved from, the *utility* trace files in addition to component specific files. For further details see “Optional traces” on page 67.
3. The commands **mqsilcc**, **mqsiiimpexpmsgset** and **mqsicopymsgset** are not eligible for utility trace.

Authorization

If the **-f** flag is specified, the user ID used to invoke this command must have access to the trace file. If the **-f** flag is not specified, the user ID used to issue the command must have **mqbrkrs** authority.

Responses

- BIP8020 Unable to access database
- BIP8029 Broker not configured
- BIP8032 Unable to connect to queue
- BIP8033 Unable to send XML message
- BIP8035 Response not received within timeout
- BIP8036 Negative response received
- BIP8037 Unsupported flag selected
- BIP8038 Unsupported command option

- BIP8039 Execution group not available
- BIP8040 Unable to connect to database
- BIP8132 Invalid qualifier

Examples

User trace:

```
mqsireadlog MQSI_SAMPLE_BROKER -u -e default -o trace.xml
```

Service trace:

```
mqsireadlog ConfigMgr -t -b mqsicreateconfigmgr -o trace.xml
```

You can format the log file (trace.xml in the above examples) using the command **mqsiformatlog**, or view it using an XML editor or viewer. See “Example of unformatted user trace” viewed with an XML viewer in “Retrieving user trace information” on page 69.

Related commands

“mqsichangetrace (Change trace settings)” on page 95

“mqsiformatlog (Format log)” on page 127

“mqsireporttrace (Report trace settings)” on page 146

mqsiporttrace (Report trace settings)

Purpose

Use the **mqsiporttrace** command to display the trace options currently in effect. This command is valid for:

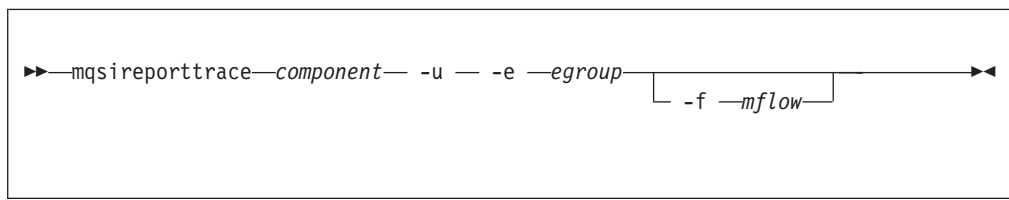
- User trace. Specify the **-u** option.
- Service trace. Specify the **-t** option. You are recommended to use this option only if directed to do so by the action described in a BIPxxxx message, or by your IBM Support Center.

If you specify a broker, or any of its resources, (execution group or message flow), you must have deployed them before you can query trace settings.

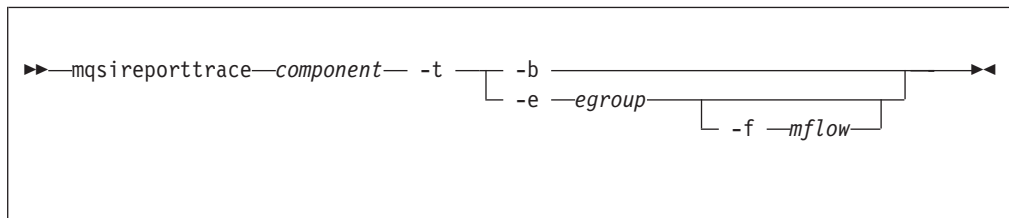
For more information on using this command, and examples of its use, see “Chapter 5. Problem determination” on page 65.

Syntax

User trace



Service trace



Required parameters

component

The name of the broker for which options are reported, or the fixed value UserNameServer. Both values are case sensitive on UNIX platforms.

Optional parameters

-u Derive report information from the user trace.

-e egroup

Specify the label of the execution group for which a report is required. This is only valid if you have specified a broker as the component.

-f mflow

Specify the label of the message flow for which a report is required. This is only valid if you have specified both a broker as the component and an execution group.

Additional parameters exclusive to service trace

You are recommended to use these options only when directed to do so by your IBM Support Center or by a BIPxxxx message.

- t Derive report information from the service trace.
- b Request a report for agent function.

Authorization

The user ID used to issue the command must have **mqbrkrs** authority.

Responses

- BIP8013 Component does not exist
- BIP8020 Unable to access database
- BIP8029 Broker not configured
- BIP8032 Unable to connect to queue
- BIP8033 Unable to send XML message
- BIP8035 Response not received within timeout
- BIP8036 Negative response received
- BIP8037 Unsupported flag selected
- BIP8038 Unsupported command option
- BIP8039 Execution group not available
- BIP8040 Unable to connect to database
- BIP8045 Message flow not available

Examples

```
mqsireporttrace BrokerA -t -e "exgrp1"
```

Related commands

"mqsichangetrace (Change trace settings)" on page 95

"mqsiformatlog (Format log)" on page 127

mqsisstart (Start component)

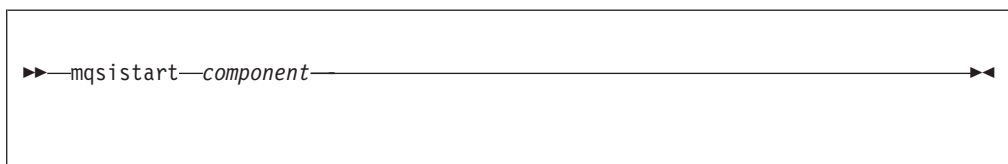
Purpose

Use **mqsisstart** to start an MQSeries Integrator component. If the queue manager associated with this component (defined in the corresponding create command) is not already running, it is also started by this command. However, no additional MQSeries services (listeners, channels, channel initiators) associated with the started component are started. You must therefore use MQSeries Services (or MQSC) to start any additional services required.

Successful completion of this command indicates that the Windows NT service or UNIX daemon has started successfully, and that the component start-up has been initiated. You must check the Windows NT Event log or UNIX syslog to determine if the component has successfully completed start-up, and is in an active state. Any errors detected by the component that have prevented successful start-up are recorded in the log. You should continue to monitor the Windows NT Event log or UNIX syslog.

Note: If the queue manager supporting the component specified on this command is not already running, it is started by this command, but, on Windows NT, is not started as a service. On this system, it will therefore stop if you log off. To avoid this happening, you must either remain logged on, or you must change the start-up status of the queue manager service (described in "Starting MQSeries queue managers as a Windows NT service" on page 24). (If you lock your workstation, the queue manager does not stop).

Syntax



Required parameters

component

This must be either a broker name, or the fixed values ConfigMgr or UserNameServer (all are case sensitive on UNIX platforms).

Authorization

On Windows NT, the user ID used to invoke this command must have **Administrator** authority on the local system.

On UNIX platforms, the user ID used to invoke this command must either be **root** or must be the same as that specified in the **-i** parameter when the component was created. It must also be a member of the **mqrkrs** group.

When the Windows NT service or UNIX daemon is started, it runs under the user ID specified by the **-i** flag on the appropriate **mqsicreatexxxx** command. The component will only start if the ServiceUserID specified is authorized to access the home directory (where MQSeries Integrator has been installed), and the working directory (if specified by the **-w** flag on the **mqsicreatexxxx** command).

The security requirements for using this command are summarized in Table 2 on page 54 .

Responses

- BIP8012 Unable to connect to system components
- BIP8013 Component does not exist
- BIP8015 Component cannot be started
- BIP8018 Component running
- BIP8024 Unable to locate executable
- BIP8025 Component disabled
- BIP8026 Unable to start component
- BIP8027 Unable to start MQSeries
- BIP8028 MQSeries unavailable
- BIP8030 Unable to modify user privileges
- BIP8048 Unable to start queue manager
- BIP8056 Unknown queue manager
- BIP8093 Queue manager being created
- BIP8094 Queue manager stopping

Examples

```
mqsistart MQSI_SAMPLE_BROKER
```

Related commands

“mqsistop (Stop component)” on page 150

Responses

- BIP8012 Unable to connect to system components
- BIP8013 Component does not exist
- BIP8016 Component cannot be stopped
- BIP8019 Component stopped
- BIP8030 Unable to modify user privileges
- BIP8049 Unable to stop queue manager
- BIP8093 Queue manager being created
- BIP8094 Queue manager stopping

Examples

```
mqsistop ConfigMgr -q
```

Related commands

“mqsistart (Start component)” on page 148

mqsistop

Part 3. Migration and integration

Chapter 9. Previous versions of MQSeries

Integrator	155
Migrating from MQSeries Integrator Version 2.0.1	155
Upgrading your Version 2.0.1 Control Center	155
Upgrading your Version 2.0.1 brokers	156
Upgrading your Version 2.0.1 Configuration	
Manager	156
Migrating from MQSeries Integrator Version 2.0	156
Upgrading your Version 2.0 Control Center	156
Upgrading your Version 2.0 brokers	156
Upgrading your Version 2.0 Configuration	
Manager	157
Migrating NEON Rules and Formats from previous	
versions of MQSeries Integrator	158
Using NNFi and NNRI for migration.	158
Adding new Rules and Formats	158
Reloading Rules and Formats	159
User exits	159
Logs and log records	159
The MQSeries Integrator Version 1 Rules	
Daemon	159

Chapter 10. MQSeries Publish/Subscribe 161

Before you start	161
Figures used in this chapter	161
Commands and options	161
Stream queues	163
Running two independent broker networks	163
Creating and operating a heterogeneous network	163
Adding an MQSeries Integrator broker as a leaf	
node	165
Adding an MQSeries Integrator broker as a	
parent node	167
Deleting brokers in a heterogeneous network	169
Migrating MQSeries Publish/Subscribe brokers	170
The Control Center and migration	171
Migrating a single broker	171
Preparing for the migration.	171
Preparing the replacement broker	172
Migrating the MQSeries Publish/Subscribe	
broker.	172
Deploying the stream queues	173
Migrating a broker network	173
Stage 1: migration of the LONDON broker	174
Stage 2: migration of the NEWYORK broker	175
Stage 3: migration of the TOKYO broker	177
A network of migrated brokers	178

Chapter 9. Previous versions of MQSeries Integrator

This chapter describes the tasks you must complete to provide run-time support following a migration from MQSeries Integrator Version 1, 2.0 or 2.0.1 to MQSeries Integrator Version 2.0.2.

For details of other areas impacted by migration:

- For planning information, see the *MQSeries Integrator Introduction and Planning*.
- For message and message flow development and deployment, see *MQSeries Integrator Using the Control Center*.

Migrating from MQSeries Integrator Version 2.0.1

These are the tasks you must complete to migrate from MQSeries Integrator Version 2.0.1 to MQSeries Integrator Version 2.0.2.

Upgrading your Version 2.0.1 Control Center

You are recommended to upgrade all Control Centers to Version 2.0.2 when the Configuration Manager or any broker is upgraded to Version 2.0.2. This enables new messages to be correctly displayed by the Control Center.

Alternatively, you can achieve the same effect by putting the Version 2.0.2 message catalog file `MQSIv202_xx.properties` into the `<mqs_i_root>\Tool` directory of each Version 2.0 or Version 2.0.1 Control Center. Note that there are different language versions of this file; for example, `MQSIv202_en.properties` or `MQSIv202_ja.properties`.

When upgrading your Control Center to Version 2.0.2 you should first check in any message flows that you have newly created or have checked out. When they are checked out again after the upgrade, they will reflect any changes made to IBM supplied nodes.

The following table lists the set of new and deprecated nodes for each release of MQSeries Integrator. This might effect you if you import a workspace into a new version of a Control Center where the workspace was created by exporting from an older version of the Control Center.

Version	New nodes	Deprecated nodes
2.0.1	<ul style="list-style-type: none">• FlowOrder• Label• RouteToLabel	
2.0.2	<ul style="list-style-type: none">• MQeInput• MQeOutput• NEONMap• NEONRulesEvaluation• NEONTransform• SCADAInput• SCADAOutput	<ul style="list-style-type: none">• NEONFormatter• NEONRules

Migrating from Version 2.0.1

If a new node is missing from your workspace, you can add the nodes to the Control Center by using *Message Flow* → *Add to workspace*. The deprecated NEON nodes are still available if required.

Upgrading your Version 2.0.1 brokers

After installing version 2.0.2 on a machine, you must migrate each different database used by brokers on that machine (in other words, each different database specified on the `-n` flag of the `mqsicreatebroker` command when the brokers were created) to the Version 2.0.2 level. For each database thus used, you should:

1. Create a new temporary broker, with any convenient name, specifying the name of the database with the `-n` flag on the `mqsicreatebroker` command.
2. Delete the temporary broker that you have just created using the `mqsdeletebroker` command.

You should only start brokers that use a particular database after you have carried out the above steps for that database.

Upgrading your Version 2.0.1 Configuration Manager

You must delete and recreate the Configuration Manager in order to pick up new or changed IBM supplied nodes.

1. Delete the Configuration Manager.

If you want to preserve your existing configuration data:

Run `mqsdeleteconfigmgr` (without `-n -m`)

If you do not want to preserve your existing configuration data:

Run `mqsdeleteconfigmgr -n -m`

2. Recreate the Configuration Manager: run `mqsicreateconfigmgr` (with appropriate parameters)
3. Start the Configuration Manager and a Control Center.
4. Ensure that the instructions in “Upgrading your Version 2.0.1 brokers” are carried out.
5. Perform a complete deploy from the Control Center Assignments view to all brokers.

Migrating from MQSeries Integrator Version 2.0

These are the tasks you must complete to migrate from MQSeries Integrator Version 2.0 to MQSeries Integrator Version 2.0.2.

Upgrading your Version 2.0 Control Center

In a scenario where there are different versions of MQSeries Integrator Version 2 components, all components from both versions can be used together except that a Version 2.0.2 Control Center will not work with a Version 2.0 Configuration Manager. This combination results in message BIP1360 being issued.

You should proceed by following the instructions in “Upgrading your Version 2.0.1 Control Center” on page 155.

Upgrading your Version 2.0 brokers

Your brokers must be upgraded to pick up modifications to broker database tables. Your brokers must be stopped while this upgrade is performed. The upgrade command, `mqsigratebroker`, operates at the broker database level, so if you

Migrating from Version 2.0

have several brokers sharing the same database then the command need only be issued once. You are advised to back up your broker database before starting.

For each DB2 broker database in your broker domain:

1. Issue db2cmd to open a DB2 command window.
2. From the DB2 command window issue:
`mqsigratebroker db2 <broker_database> <database_user_name> <password>`

For each Microsoft SQL Server broker database in your broker domain:

1. Open a new MS-DOS command window.
2. From the MS-DOS command window issue:
`mqsigratebroker msql <broker_database> <database_user_name> <password>`

Note: You **must** ensure that the **mqsigratebroker** command is run using the same database userid and password that were used on the **mqsicreatebroker** command (-u DataSourceUserId -p DataSourcePassword).

You should proceed by following the instructions in “Upgrading your Version 2.0.1 brokers” on page 156 to create and delete a temporary broker for each broker database.

Upgrading your Version 2.0 Configuration Manager

You must delete and recreate the Configuration Manager in order to pick up new or changed IBM supplied nodes and new IBM supplied message sets, and to incorporate modifications to tables. The new nodes are **RouteToLabel**, **Label** and **FlowOrder**. The new message sets are for the MQSeries link for R/3 SAP bad message header and for the MQSeries PCF message headers.

1. Delete and recreate the Configuration Manager.
If you want to preserve your existing configuration data:
 - a. Run `mqsdeleteconfigmgr` (without -n -m)
 - b. Issue db2cmd to open a DB2 command window.
 - c. From the DB2 command window issue:
 - `mqsigrateconfigmgr <configuration_repository> <user_name> <password>`
 - d. Run `mqsicreateconfigmgr` (with appropriate parameters)
 - e. Before you start the Configuration Manager:
 - Run `mqsiiimpexpmsgset -i` on filename `mqcfh.mrp` in the `mrp` directory
 - Run `mqsiiimpexpmsgset -i` on filename `smqbmh.mrp` in the `mrp` directory

If you do not want to preserve your existing configuration data:

- a. Run `mqsdeleteconfigmgr -n -m`
 - b. Run `mqsicreateconfigmgr` (with appropriate parameters)
- (No **mqsigrateconfigmgr** or **mqsiiimpexpmsgset** steps are needed)
2. Start the Configuration Manager and a Control Center.
 3. Ensure that the instructions in “Upgrading your Version 2.0 brokers” on page 156 are carried out.
 4. Perform a complete deploy from the Control Center Assignments view to all brokers.

Migrating from Version 2.0

Failure to follow these instructions might result in either message BIP1205 or BIP1502 appearing when you deploy, with CMRMUUUID cited as the column name in error.

Note: You **must** ensure that the `mqsimigrateconfigmgr` command is run using the same database userid and password that were used on the `mqsicreateconfigmgr` command (`-u DataBaseUserId -p DataBasePassword`).

Migrating NEON Rules and Formats from previous versions of MQSeries Integrator

In order to use NEON Rules and Formats from previous versions of MQSeries Integrator with MQSeries Integrator Version 2.0.2, some migration steps are required.

Using NNFie and NNRie for migration

The NEONRules and NEONFormatter Support for MQSeries Integrator uses a different database format to previous versions of MQSeries Integrator. You must therefore migrate any existing NEON Rules and Formats. This is achieved with the following steps:

- Step 1. *Before* uninstalling the previous version of MQSeries Integrator, export the Rules and Formats using the version of NNFie and NNRie supplied with that version.
- Step 2. After installing MQSeries Integrator Version 2.0.2, create the new NEON Rules and Formats database by following the instructions in “Appendix B. Using NEON” on page 195.
- Step 3. Import the Rules and Formats into the new database using the version of NNFie and NNRie supplied with MQSeries Integrator Version 2.0.2.
- Step 4. (Optional) Use the consistency checker program supplied with MQSeries Integrator Version 2.0.2 to check the consistency of the Rules and Formats newly imported into the database.

Versions of NNFie and NNRie for all previous versions of MQSeries Integrator are included with MQSeries Integrator Version 2.0.2 in the migration directory under the main installation home. Use these if further migration of Rules and Formats from previous versions of MQSeries Integrator is required subsequent to installation of MQSeries Integrator Version 2.0.2.

See also “Importing rules and formats” on page 198 for further details about the use of the NNFie and NNRie commands.

Adding new Rules and Formats

The NEONRules and NEONFormatter GUI interfaces supplied with MQSeries Integrator Version 2.0.2 can be used to create new Rules and Formats and edit those already in the Rules and Formats database. Previous versions of the NEONRules and NEONFormatter GUI are *not* compatible with the new database format and should *not* be used.

For information about how access the NEON Rules and Formats from a message flow see “Appendix B. Using NEON” on page 195.

Reloading Rules and Formats

Previous versions of MQSeries Integrator allowed the Rules and Formats database to be reloaded dynamically.

- MQSeries Integrator Version 1 performed this action on receiving a message with the OPT_RELOAD option set.
- MQSeries Integrator Version 2.0.1 supplied the **mqsinrfreload** command.

The **mqsinrfreload** command is still supplied with MQSeries Integrator Version 2.0.2, and it continues to work as previously with the NEONRules and NEONFormatter nodes, and the NEON domain message parser. However it will not work with the three new NEON nodes and the new NEONMSG domain message parser. In order to cause these to reload the NEON Rules and Formats from the database, the broker must be stopped and restarted.

User exits

Refer to the NEONRules and NEONFormatter Support documentation for details of how to use User Exits from previous versions of MQSeries Integrator.

You must compile NEON user exits as multi-thread objects which **must** link with mutli-thread (thread-safe) libraries. Failure to do this will result in unpredictable errors. It is always good programming pratice to follow these guidelines.

Logs and log records

Log records generated by all NEONRules and NEONFormatter Support nodes and parsers are written to the log file defined for the message flow in which the nodes appear. For details of how to specify a log file for a message flow, see *MQSeries Integrator Using the Control Center*.

The NEONTransform, NEONMap and NEONRulesEvaluation nodes can, if required, write additional diagnostic information to a flat text trace file specified as a property of the node.

For a general discussion of logs and log contents, see “Chapter 5. Problem determination” on page 65.

The MQSeries Integrator Version 1 Rules Daemon

MQSeries Integrator Version 1 and Version 1.1 supplied the MQSIRuleng Rules Daemon which accepted messages from an input queue, evaluated them against the NEON Rules and Formats database and then placed them either on a specified output queue or one of various special queues. The Rules Daemon is not supplied in its original form with MQSeries Integrator Version 2.0.2. However, the function which it provided can be reproduced with a simple message flow:

Logging

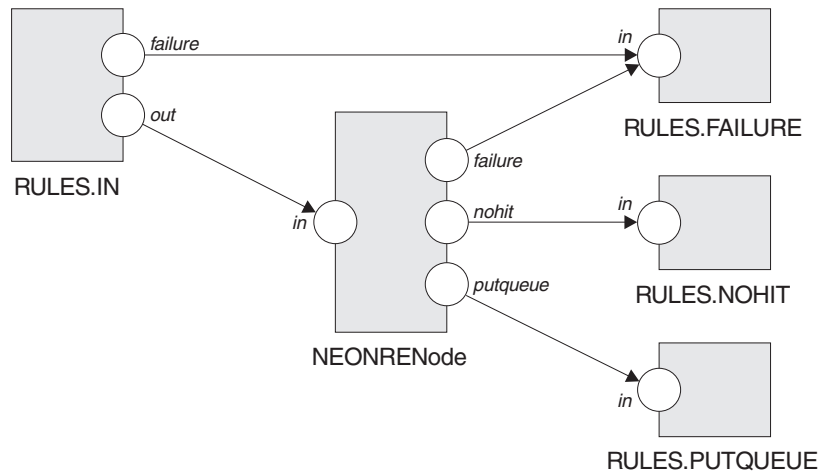


Figure 2. A message flow to reproduce the function of the NEON Rules Daemon

- An MQInput node (RULES.IN) has its *out* terminal connected to the *in* terminal of a NEONRulesEvaluation node (NEONRENode).
- The *route* and *propagate* terminals of the NEONRulesEvaluation node NEONRENode are not connected. These terminals are part of additional MQSeries Integrator Version 2.0.2 NEON functionality and are not required to reproduce the functionality of MQSeries Integrator Version 1.
- Both RULES.IN and NEONRENode have their *failure* terminals connected to the *in* terminal of an MQOutput node (RULES.FAILURE).
- NEONRENode has its *nohit* terminal connected to the *in* terminal of another MQOutput node (RULES.NOHIT).
- NEONRENode has its *putqueue* terminal connected to the *in* terminal of a third MQOutput node (RULES.PUTQUEUE). The RULES.PUTQUEUE node should be configured with an output mode of *list*. This will ensure that messages arriving at it will be placed on the queue specified by the *putqueue* action.
- In order for the NEONRulesEvaluation node to function, you should follow the steps detailed under “Access to Rules and Formats” on page 197.

Chapter 10. MQSeries Publish/Subscribe

This chapter describes the tasks you must complete to integrate your two broker networks, or to migrate your MQSeries Publish/Subscribe brokers to MQSeries Integrator Version 2 brokers.

The following scenarios, described in *MQSeries Integrator Introduction and Planning*, are addressed:

1. "Running two independent broker networks" on page 163.
You can choose to have two independent broker networks, and therefore have two separate broker domains for publications and subscriptions.
2. "Creating and operating a heterogeneous network" on page 163.
You can integrate the two networks to allow publications and subscriptions to flow throughout the mixed network.
3. "Migrating MQSeries Publish/Subscribe brokers" on page 170.
You can selectively and gradually migrate individual brokers from MQSeries Publish/Subscribe to MQSeries Integrator Version 2.

Before you start

Before you start to implement a migration or integration of an MQSeries Publish/Subscribe and MQSeries Integrator network, refer to the detailed planning information provided in *MQSeries Integrator Introduction and Planning*. This gives a description of the options implemented here, and the advantages of each.

If appropriate, you must check that you have updated any of your client applications that are affected by the differences in the two products. See *MQSeries Integrator Introduction and Planning* for details of the differences, and suggested changes, before you implement the network changes described here.

Figures used in this chapter

The figures illustrating the migration scenarios in this chapter use the key shown in Figure 3.

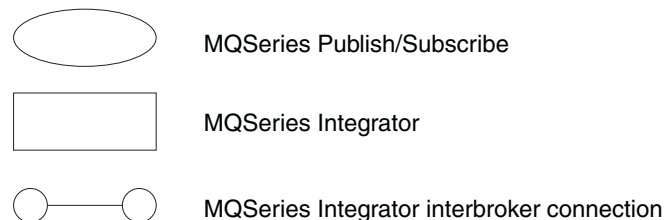


Figure 3. Key to the integration and migration figures

Commands and options

To complete the tasks required to achieve an integrated network, or to migrate MQSeries Publish/Subscribe brokers to MQSeries Integrator, you will use a number of commands supplied by MQSeries Integrator and MQSeries Publish/Subscribe.

The following commands are provided by MQSeries Publish/Subscribe:

- **clrmqbrk.** This command removes an MQSeries Publish/Subscribe broker as a neighbor of an MQSeries Integrator broker.
- **dltmqbrk.** This command deleted an MQSeries Publish/Subscribe broker and removes and neighbor connections.
- **endmqbrk.** This command stops a running MQSeries Publish/Subscribe broker.
- **migmqbrk.** This command transfers state information from an MQSeries Publish/Subscribe broker to an MQSeries Integrator broker. For further details of what this information includes, see “Migrating MQSeries Publish/Subscribe brokers” on page 170.

Note: This command is only available in the latest level of the MQSeries Publish/Subscribe SupportPac. You must download this latest level to implement migration of MQSeries Publish/Subscribe brokers to MQSeries Integrator brokers.

- **strmqbrk.** This command starts an MQSeries Publish/Subscribe broker.

The following commands are provided by MQSeries Integrator:

- **mqscreatebroker.** This command creates an MQSeries Integrator broker.
- **mqsistart.** This command starts an MQSeries Integrator broker.
- **mqsiclearmqpubsub.** This command removes an MQSeries Publish/Subscribe broker as a neighbor of an MQSeries Integrator broker.
- **mqsijoinmqpubsub.** This command joins an MQSeries Integrator broker to an MQSeries Publish/Subscribe broker network.
- **mqsilistmqpubsub.** This command displays the status of MQSeries Publish/Subscribe neighbor brokers after a join command.

Table 6 gives you a further reference to the definition and detail of all these commands. You must check these references for further information, or to correct any errors that occur.

Table 6. Where to find command information

Command	Reference
clrmqbrk	<i>MQSeries Publish/Subscribe User's Guide</i>
dltmqbrk	<i>MQSeries Publish/Subscribe User's Guide</i>
endmqbrk	<i>MQSeries Publish/Subscribe User's Guide</i>
migmqbrk	<i>MQSeries Publish/Subscribe User's Guide</i>
strmqbrk	<i>MQSeries Publish/Subscribe User's Guide</i>
mqsiclearmqpubsub	“mqsiclearmqpubsub (Remove MQSeries Publish/Subscribe broker as a neighbor)” on page 101
mqscreatebroker ¹	“mqscreatebroker (Create broker)” on page 105
mqsijoinmqpubsub	“mqsijoinmqpubsub (Join broker to MQSeries Publish/Subscribe parent broker)” on page 131
mqsilistmqpubsub	“mqsilistmqpubsub (List MQSeries Publish/Subscribe neighbor broker status)” on page 137
mqsistart	“mqsistart (Start component)” on page 148
Note:	
1. This command can also be issued through the Command Assistant. See “Chapter 7. Using the MQSeries Integrator Command Assistant” on page 87 for details.	

Stream queues

The migration examples in this chapter always specify the `noshare` option on the commands that create stream queues. MQSeries Publish/Subscribe required this option to be specified: MQSeries Integrator does not require it. Sharing queues is one way of increasing throughput, but can affect the order in which publications are received by subscribers. For details of throughput and order, see the section entitled *Throughput* in Appendix A of *MQSeries Integrator Introduction and Planning*.

Running two independent broker networks

If you want to run in this mode with two separate, independent networks, you do not have to take any specific actions. You can retain your existing MQSeries Publish/Subscribe network, and install and configure an MQSeries Integrator Version 2 network, without any interaction.

Your existing applications in both networks can work unchanged, and do not interact in any way.

You must be aware that a single queue manager cannot support both an MQSeries Publish/Subscribe broker and an MQSeries Integrator Version 2 broker. If you have brokers of both types on a single system, each must have its own dedicated queue manager.

However, you must ensure that you have the required level of MQSeries for your MQSeries Publish/Subscribe systems:

- **MQSeries Version 5.0.** For MQSeries Publish/Subscribe brokers, you must install CSD7.

Note: You cannot run MQSeries Integrator brokers on MQSeries Version 5.0 at any service level. This option is only valid for MQSeries Publish/Subscribe brokers.

- **MQSeries Version 5.1.** For MQSeries Publish/Subscribe brokers, you must install CSD4 on Windows NT, or CSD1 on other platforms.

- **MQSeries Version 5.2.** For MQSeries Publish/Subscribe brokers, no CSDs are required.

If you do not upgrade MQSeries to the specified levels on the systems on which you intend to run MQSeries Publish/Subscribe brokers, it is possible that some publications sent by MQSeries Integrator brokers will be wrongly put to the dead-letter queue (DLQ) by an MQSeries Publish/Subscribe neighbor broker.

Creating and operating a heterogeneous network

You can integrate your existing MQSeries Publish/Subscribe broker network with an MQSeries Integrator Version 2.0.2 broker network to create a mixed, heterogeneous network. This enables publications and subscriptions to be propagated through one logical network, made up of two or more physical networks. Thus subscribers to the MQSeries Publish/Subscribe brokers can target information being published to the MQSeries Integrator broker network, and vice versa. MQSeries Publish/Subscribe brokers and MQSeries Integrator Version 2.0.2 brokers can be running on Windows NT or other platforms.

Heterogeneous networks

There are two ways in which an MQSeries Integrator broker can be joined to the MQSeries Publish/Subscribe network: it can be joined as a leaf node (that is, as a child of an existing MQSeries Publish/Subscribe broker) or as a parent node (that is, as the parent of an existing MQSeries Publish/Subscribe broker). For details of the advantages of these two options, and why you might choose one or the other, see *MQSeries Integrator Introduction and Planning*.

Every MQSeries Integrator broker you integrate into an MQSeries Publish/Subscribe network must have a minimum of two queues available:

- **SYSTEM.BROKER.DEFAULT.STREAM**
This queue supports the default publication stream. You must create this queue on every broker. You must also create and deploy a message flow that services this stream queue.
- **SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS**
This queue is used by the broker to communicate with neighboring MQSeries Publish/Subscribe brokers. You must create this queue on every broker, however the message flow for this queue is created internally by the broker.

You must ensure that the two queue managers (for example, MQPS_BROKER1 and MQSI_SAMPLE_BROKER) have MQSeries connections. Sender-receiver pairs of channels and a transmission queue at the sending end are required for both queue managers to enable two-way communications. For example:

1. On the queue manager for broker MQPS_BROKER1:

```
define channel('MQPS_TO_MQSI') chltype(sdr) trptype(tcp)
conname('MQSISYS1') xmitq('MQSI_SAMPLE_BROKER) replace

define channel('MQSI_TO_MQPS') chltype(rcvr) trptype(tcp) replace

define qlocal('MQSI_SAMPLE_BROKER') usage(xmitq)
get(enabled) put(enabled) replace
```
2. On the queue manager for broker MQSI_SAMPLE_BROKER:

```
define channel('MQSI_TO_MQPS') chltype(sdr) trptype(tcp)
conname('MQSISYS2') xmitq('MQPS_BROKER1) replace

define channel('MQPS_TO_MQSI') chltype(rcvr) trptype(tcp) replace

define qlocal('MQPS_BROKER1') usage(xmitq)
get(enabled) put(enabled) replace
```

The requirements for channels and transmission queues, and how to activate them is described in the *MQSeries Publish/Subscribe User's Guide*: the same requirements exist whether the broker being joined to the network is an MQSeries Publish/Subscribe broker or an MQSeries Integrator broker.

The steps described in the following sections (“Adding an MQSeries Integrator broker as a leaf node” on page 165 and “Adding an MQSeries Integrator broker as a parent node” on page 167) assume you are joining:

- An MQSeries Integrator broker named MQSI_SAMPLE_BROKER. When this broker was created, the same name was specified for the queue manager.
- An MQSeries Publish/Subscribe broker network with a root broker MQPS_ROOT_BROKER, and two leaf brokers MQPS_BROKER1 and MQPS_BROKER2.

You must substitute the real names of your brokers for these examples wherever they are used.

Note: All commands shown must be issued on the system on which the appropriate resource is defined. MQSeries commands (for example, to define a queue) are shown in MQSC format. For more information about MQSeries commands, see *MQSeries System Administration*.

Adding an MQSeries Integrator broker as a leaf node

This section describes the actions you must complete to add the MQSeries Integrator broker as a leaf node within your MQSeries Publish/Subscribe broker network. This is illustrated in Figure 4 which shows MQSeries Integrator broker **MQSI_SAMPLE_BROKER** joined to the MQSeries Publish/Subscribe network, with broker **MQPS_BROKER1** as its parent broker.

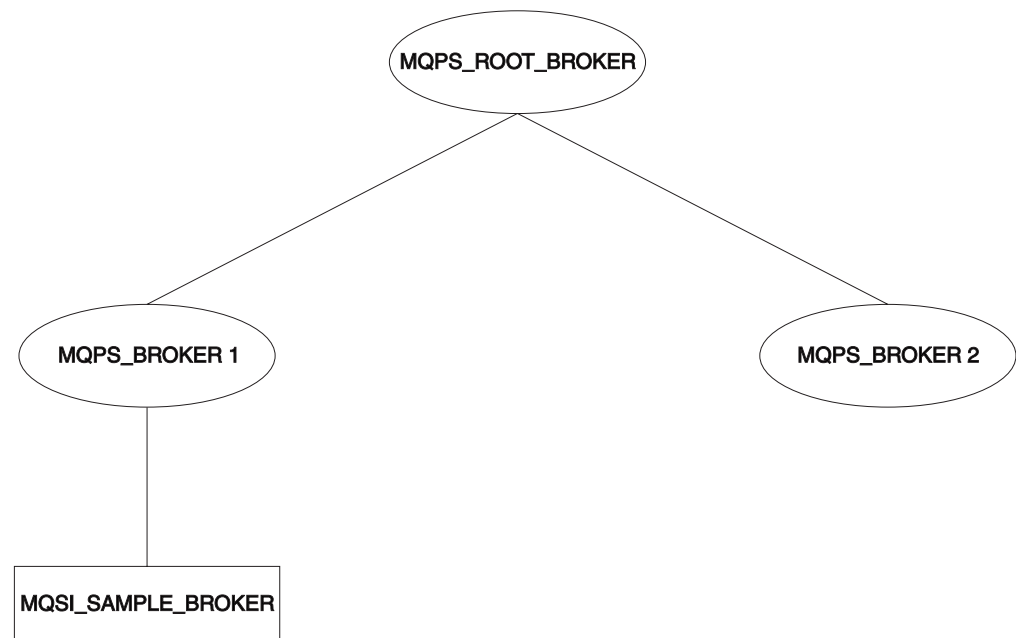


Figure 4. Adding an MQSeries Integrator broker as a leaf node

- Step 1. Ensure that the MQSeries Integrator broker's default execution group is successfully deployed. This execution group is deployed the first time you deploy a newly created MQSeries Integrator broker. You can check the status of both the execution group and the broker from the **Topology** view in the Control Center. For further information about deployment, see *MQSeries Integrator Version 2.0.2 Using the Control Center*.
- Step 2. Define the queue required to support interbroker communications with MQSeries Publish/Subscribe neighbors on the MQSeries Integrator broker's queue manager:


```
define qlocal(SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS) noshare
```
- Step 3. Stop the MQSeries Integrator broker:


```
mqsistop MQSI_SAMPLE_BROKER
```
- Step 4. Restart the MQSeries Integrator broker:


```
mqsistart MQSI_SAMPLE_BROKER
```

When the MQSeries Integrator broker is restarted, the presence of the interbroker queue (defined above) enables the broker to receive and process messages on this queue.

Heterogeneous networks

- Step 5. Create the resources required on the MQSeries Integrator broker to support the default MQSeries Publish/Subscribe stream:
- a. Create the default stream queue:

```
define qlocal(SYSTEM.BROKER.DEFAULT.STREAM) noshare
```
 - b. Create a message flow based on the supplied publish/subscribe message flow:
 - 1) Start up the Control Center and select the message flow view.
 - 2) If you have not already imported and saved the default message flows supplied (described in *MQSeries Integrator Introduction and Planning*), you are recommended to import these now. This will enable you to reuse the default publish/subscribe flow here. Select *File* → *Import* and open the file, called *SamplesWorkspaceForImport*, in the *examples* subdirectory within the MQSeries Integrator home directory. It will take a few minutes to complete. (You can find more detailed instructions for working with this import file in the *MQSeries Integrator Installation Guide* for your computer platform.)
(If you prefer, you can create your own message flow. See the *MQSeries Integrator Version 2.0.2 Using the Control Center* for details of how to do this.)
 - 3) Make a copy of the supplied message flow and rename it.
 - 4) Check the properties of the nodes in the message flow. You must set the appropriate input (stream) queue property for the MQInput node. Check the other properties of the nodes are set correctly for your requirements.
 - 5) Finally, check in your changes and deploy the message flow to the default execution group of the broker MQSI_SAMPLE_BROKER.

You can find full details of how to complete these steps in the online help for the Control Center.

- Step 6. If you are using additional streams in the MQSeries Publish/Subscribe network, you must also enable these on the MQSeries Integrator broker. Although the MQSeries Integrator broker is able to support all the streams of its MQSeries Publish/Subscribe neighbors, you need only define queues, and define and deploy message flows, for those streams requested by MQSeries Integrator subscriber clients.
- a. Create a local queue on the MQSeries Integrator broker's queue manager for each stream on which messages are to be processed. For example:

```
define qlocal(STREAM.X) noshare
```
 - b. Create and deploy a message flow to read and process the MQSeries Publish/Subscribe messages that are sent to each stream (publication) queue.
You can use the supplied publish/subscribe message flow as the basis for each new message flow. Each MQInput node representing a non-default stream must have the property **implicitStreamNaming** set (this is the default setting).

- Step 7. Ensure that the MQSeries Publish/Subscribe broker is running; if not, you can start it using the start command:
- ```
strmqbrk MQPS_BROKER1
```

## Heterogeneous networks

Step 8. Ensure that the MQSeries connection between the two brokers is enabled: you must start the listeners for the receiver channels, and you must then start the sender channels.

Step 9. Join the MQSeries Integrator broker to the MQSeries Publish/Subscribe network as a child of the MQSeries Publish/Subscribe broker:

```
mqsijoinmqpubsub MQSI_SAMPLE_BROKER -p MQPS_BROKER1
```

Step 10. Verify the success of the join command to ensure that the MQSeries Publish/Subscribe broker is an active neighbor:

```
mqsilistmqpubsub MQSI_SAMPLE_BROKER
```

If the join command has completed successfully, you will see a response to the list command that will be similar to:

```
BIP8090I: MQSeries Publish/Subscribe neighbor MQSI_SAMPLE_BROKER
is active
```

```
BIP8091I: Common stream SYSTEM.BROKER.DEFAULT.STREAM
BIP8091I: Common stream STREAM.X
```

## Adding an MQSeries Integrator broker as a parent node

This section describes the actions you must complete to add the MQSeries Integrator broker as a leaf node within your MQSeries Publish/Subscribe broker network. This is illustrated in Figure 5, which shows MQSeries Integrator broker MQSI\_SAMPLE\_BROKER joined to the MQSeries Publish/Subscribe network as the new parent node (that is, as the parent of the original parent node MQPS\_ROOT\_BROKER).

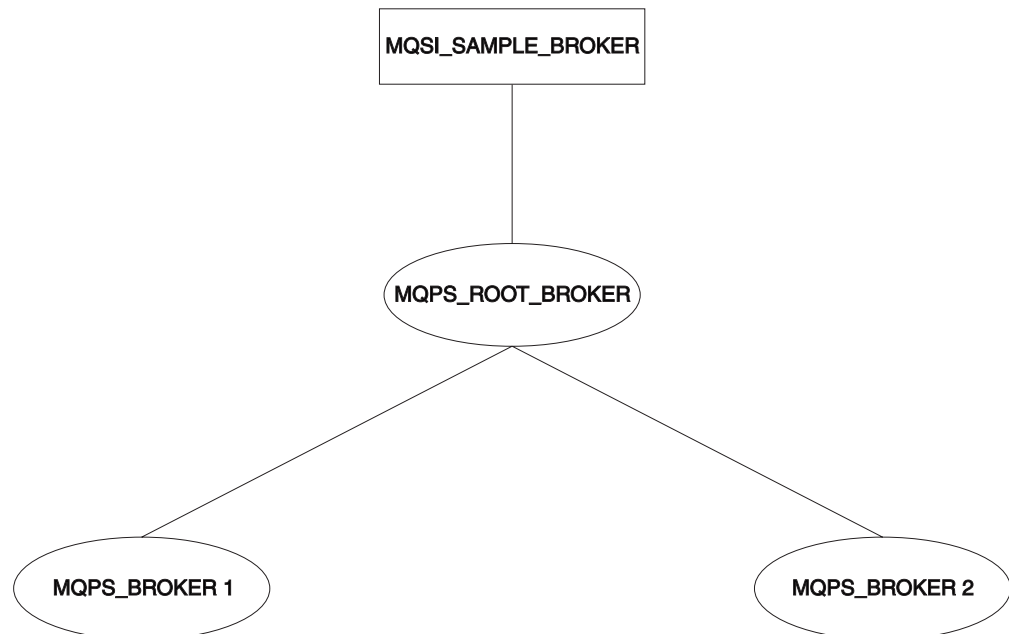


Figure 5. Adding an MQSeries Integrator broker as a parent node

Step 1. Ensure that the MQSeries Integrator broker's default execution group is successfully deployed. This execution group is deployed the first time you deploy a newly created MQSeries Integrator broker. You can check the status of both the execution group and the broker from the **Topology**

## Heterogeneous networks

view in the Control Center. For further information about deployment, see *MQSeries Integrator Version 2.0.2 Using the Control Center*.

- Step 2. Define the queue required to support interbroker communications with MQSeries Publish/Subscribe neighbors on the MQSeries Integrator broker's queue manager:

```
define qlocal(SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS) noshare
```

- Step 3. Stop the MQSeries Integrator broker:

```
mqsisstop MQSI_SAMPLE_BROKER
```

- Step 4. Restart the MQSeries Integrator broker:

```
mqsisstart MQSI_SAMPLE_BROKER
```

When the MQSeries Integrator broker is restarted, it is enabled to receive and process messages on the interbroker queue.

- Step 5. Create the resources required on the MQSeries Integrator broker to support the default MQSeries Publish/Subscribe stream:

- a. Create the default stream queue:

```
define qlocal(SYSTEM.BROKER.DEFAULT.STREAM) noshare
```

- b. Create a message flow for publish/subscribe, either your own, or one based on the supplied publish/subscribe message flow: :

- 1) Start up the Control Center and select the message flow view.
- 2) Make a copy of the supplied message flow and rename it. (You must import this default message flow before you can access and use it: details are in "Adding an MQSeries Integrator broker as a leaf node" on page 165)
- 3) Check the properties of the nodes in the message flow. You must set the appropriate input (stream) queue property for the MQInput node. Check the other properties of the nodes are set correctly for your requirements.
- 4) Finally, deploy the message flow to the default execution group of the broker MQSI\_SAMPLE\_BROKER.

You can find full details of how to complete these steps in the online help for the Control Center.

- Step 6. If you are using additional streams in the MQSeries Publish/Subscribe network, you must also enable these on the MQSeries Integrator broker.

Although the MQSeries Integrator broker is able to support all the streams of its MQSeries Publish/Subscribe neighbors, you need only define queues, and define and deploy message flows, for those streams requested by MQSeries Integrator subscriber clients.

- a. Create a local queue on the MQSeries Integrator broker's queue manager for each stream on which messages are to be processed:

For example:

```
define qlocal(STREAM.X) noshare
```

- b. Create and deploy a message flow to read and process the MQSeries Publish/Subscribe messages that are sent to each stream (publication) queue.

You can use the supplied publish/subscribe message flow as the basis for each new message flow. Each MQInput node representing a non-default stream must have the property **implicitStreamNaming** set.



## Heterogeneous networks

- Step 7. Enter the following MQSeries Publish/Subscribe command against the broker that is the current MQSeries Publish/Subscribe parent broker, to terminate its activities:

```
endmqbrk -c -m MQPS_ROOT_BROKER
```

This requests a controlled shutdown (-c). When the shutdown has completed, the broker can be restarted. You can request an immediate shutdown (by specifying -i instead of -c) if you need to force this shutdown to complete.

- Step 8. Ensure that the MQSeries connection between the two brokers is active: you must start the listeners for the receiver channels, and you must then start the sender channels.

- Step 9. Enter the following MQSeries Publish/Subscribe command against the current MQSeries Publish/Subscribe parent broker to restart it:

```
strmqbrk -m MQPS_ROOT_BROKER -p MQSI_SAMPLE_BROKER
```

If the queue manager associated with the MQSeries Integrator broker MQSI\_SAMPLE\_BROKER has not been created with the same name as the broker, you must specify the queue manager name here after the -p flag, not the broker name.

- Step 10. Verify the success of the integration:

```
mqsilistmqpubsub MQSI_SAMPLE_BROKER
```

If the MQSeries Integrator broker has been integrated into the MQSeries Publish/Subscribe network successfully, you will see a response that will be similar to:

```
BIP8090I: MQSeries Publish/Subscribe neighbor MQSI_SAMPLE_BROKER
is active
```

```
BIP8091I: Common stream SYSTEM.BROKER.DEFAULT.STREAM
BIP8091I: Common stream STREAM.X
```

## Deleting brokers in a heterogeneous network

If you have a mixed broker network, you must take particular care to maintain the integrity of the network if you need to remove or delete a broker from the network:

- When you issue the **mqsideletebroker** command to delete an MQSeries Integrator broker, the MQSeries Publish/Subscribe brokers that are neighbors of this MQSeries Integrator broker are not automatically informed of its deletion. You are therefore recommended to remove the MQSeries Integrator broker from the network using the clear commands **mqsiclearmqpubsub** (at the MQSeries Integrator broker) and **clrmqbrk** (at its MQSeries Publish/Subscribe neighbors) before you delete it.

You can find references to more information about both of these commands in Table 6 on page 162.

- If you delete an MQSeries Integrator broker before you remove it from the network, and it has a parent MQSeries Publish/Subscribe broker, the parent broker continues to attempt to send publication and subscription messages to it. You can correct this behavior by issuing the **clrmqbrk** command at the parent. For example, if you issue:

```
mqsideletebroker -m MQSI_CHILD_BROKER
```

## Heterogeneous networks

while the MQSeries Integrator broker is still known to its parent MQSeries Publish/Subscribe broker, you can then issue the command

```
c1rmqbrk -m MQPS_PARENT_BROKER -c MQSI_CHILD_BROKER
```

to the parent broker to clean up the network.

- When you issue the **dltmqbrk** command to delete an MQSeries Publish/Subscribe broker that is a child of an MQSeries Integrator broker, the MQSeries Integrator broker does receive notification of the deletion. Therefore you do not have to issue the **mqsiclearmqpubsub** command to remove knowledge of the deleted child at the MQSeries Integrator parent broker. For example, if you want to delete the child broker MQPS\_CHILD\_BROKER you must issue the following single command:

```
d1tmqbrk -m MQPS_CHILD_BROKER
```

**Note:** You are prevented from deleting an MQSeries Publish/Subscribe broker that is a parent of any broker: the **dltmqbrk** command fails.

---

## Migrating MQSeries Publish/Subscribe brokers

When you plan for migration of one or more brokers, you must take account of the product differences described in *MQSeries Integrator Introduction and Planning*. You must take any action to make changes to applications, or topics, or both, before you start migration.

The information here tells you the steps you must take to migrate a single broker, and an MQSeries Publish/Subscribe broker network.

These steps result in *replacement* of the MQSeries Publish/Subscribe brokers by MQSeries Integrator brokers. Each replacement MQSeries Integrator broker must be created on the same queue manager as the MQSeries Publish/Subscribe broker it is replacing. Because the MQSeries Publish/Subscribe broker shares the same name as the queue manager that supports it, you must specify the MQSeries Publish/Subscribe broker name as the queue manager parameter on the **mqsicreatebroker** command (the **-q** flag).

Migration involves the transfer of the following state information from the MQSeries Publish/Subscribe broker to the MQSeries Integrator broker:

- Subscriptions.  
All client subscriptions are exported from all streams except SYSTEM.BROKER.ADMIN.STREAM.
- Retained publications.  
All retained publications in MQRFH format are exported from all streams except SYSTEM.BROKER.ADMIN.STREAM.
- Local publishers.  
Registrations for all publishers that are producing local publications are exported from all streams except SYSTEM.BROKER.ADMIN.STREAM.
- Related brokers.  
If the broker is part of a multi-broker hierarchy, details of all of its relations are exported. This includes the names of all streams which the broker to be migrated has in common with the relation.

This information is exported as a series of messages that the migrating MQSeries Publish/Subscribe broker sends to its replacement. When migration is complete the MQSeries Publish/Subscribe broker is deleted automatically, and cannot be recreated.

### The Control Center and migration

If you are migrating an MQSeries Publish/Subscribe broker, you cannot fully deploy it in your MQSeries Integrator broker domain until migration has completed successfully. In particular, you are advised not to deploy additional execution groups or message flows until after you have successfully migrated the MQSeries Publish/Subscribe broker.

Before you start migration, the MQSeries Integrator broker must be created in the Control Center *Topology* view. You must then deploy it by selecting *File* → *Deploy* → *Delta configuration (all types)*.

If migration fails, and you want to revert to your MQSeries Publish/Subscribe broker, you must delete the broker following the guidance given in “Deleting components from the broker domain” on page 40.

### Migrating a single broker

When you migrate an MQSeries Publish/Subscribe broker that is not part of a network, you are replacing it in the network and assigning all the function previously supported by that broker to an MQSeries Integrator broker.

You must shutdown the MQSeries Publish/Subscribe broker before you attempt migration. You are therefore recommended to ensure that all applications using this broker are also quiesced.

#### Preparing for the migration

Before you can migrate a broker, you need to do some preparation.

- Step 1. Identify the MQSeries Publish/Subscribe broker you are going to migrate.  
The steps used here assume you have chosen the name MQSI\_SAMPLE\_BROKER for your new MQSeries Integrator broker, and that the MQSeries Publish/Subscribe broker you are migrating is currently hosted by the queue manager MQPS\_BROKER1.
- Step 2. Backup the queue manager hosting the MQSeries Publish/Subscribe broker.  
Although you are not required to do this, you are advised to complete this backup before you start the migration process. This allows you to retrieve the old MQSeries Publish/Subscribe broker after successful migration, if you should need to do so for any reason. The *MQSeries System Administration* book describes this backup process.
- Step 3. Quiesce any applications registered with the broker.  
Although you are not required to do this, any messages (publications, and control and client publish messages) generated during the migration exercise are queued and could cause performance or capacity problems. Ending the applications as well as the broker ensures that publish/subscribe traffic is only generated when there is a broker ready to process it.
- Step 4. End your MQSeries Publish/Subscribe broker operation:  
`endmqbrk MQPS_BROKER1`

## Migrating brokers

### Preparing the replacement broker

You are now ready to work with the new broker.

Step 1. Create an MQSeries Integrator broker.

You must create the new broker on the system on which the queue manager MQPS\_BROKER1 is defined. You must select the migration option (flag -m) on the command.

```
mqsicreatebroker MQSI_SAMPLE_BROKER -q MQPS_BROKER1
-i mqbroker -a sample -n MQSIBKDB -m
```

Step 2. Start the new MQSeries Integrator broker:

```
mqsistart MQSI_SAMPLE_BROKER
```

Step 3. Configure the broker in the Control Center.

Create the configuration for this broker in the configuration repository by adding the broker to the broker domain topology from the *Topology* view in the Control Center. This action also creates the default execution group for this broker.

Step 4. Check in your changes and deploy the broker by selecting *File* → *Deploy* → *Delta configuration*. This results in initialization messages flowing from the Configuration Manager to the new broker. This causes activation of a message flow that runs in the broker's default execution group to accept and implement the migration messages generated by the MQSeries Publish/Subscribe broker during the next step.

### Migrating the MQSeries Publish/Subscribe broker

The new MQSeries Integrator broker is ready to receive migration data for the MQSeries Publish/Subscribe broker that it is to replace.

Step 1. Migrate the MQSeries Publish/Subscribe broker function to the replacement MQSeries Integrator broker by issuing the following command:

```
migmqbrk -m MQPS_BROKER1
```

This command is supplied as part of the MQSeries Publish/Subscribe package on the Web. You must ensure you have the latest level of this command, and the *MQSeries Publish/Subscribe User's Guide* that describes its use. Check the URL identified in "Where to find more information" on page xii for details.

The command retrieves the persistent information (subscriptions and retained publications) from the MQSeries Publish/Subscribe broker, and sends it in specially constructed messages to the queue SYSTEM.BROKER.INTERBROKER.QUEUE on the new MQSeries Integrator broker. The message flow that services this queue (deployed when you deployed the broker and its default execution group) receives these messages and records the information. Once all the messages have been processed, the message flow is terminated and cannot be re-initialized.

The migration command is implemented such that it can always, but only, be re-invoked if the whole process of migration has not completed successfully. If any error occurs, for any reason, the MQSeries Publish/Subscribe broker is recoverable and can be restarted. You can then continue to use it. The MQSeries Integrator broker also exists, but has not recorded any migration information. You can delete and create this broker to restart the migration process.

If the whole process succeeds, the MQSeries Publish/Subscribe broker no longer exists and cannot be recovered: it, and all its functions, have been subsumed by the new MQSeries Integrator broker.

You receive the following message on successful completion of migration:  
MQSeries Publish/Subscribe broker has been successfully migrated

Once migration has successfully completed for all MQSeries Publish/Subscribe brokers that you plan to migrate, you are recommended to delete or rename the file `strmqbrk.exe`. This will prevent any accidental start up of the MQSeries Publish/Subscribe brokers.

### Deploying the stream queues

The new MQSeries Integrator broker is now set up to take over from the MQSeries Publish/Subscribe broker. You must create and deploy the message flows it needs to activate the streams: you do not need to define the stream queues, because these are already defined to the queue manager (the queue definitions are not deleted when the migration takes place, and the queue manager is the same for the MQSeries Publish/Subscribe broker and the MQSeries Integrator broker that has replaced it).

You can create the message flows you need by following these steps:

Step 1. Start up the Control Center and select the message flow view.

Step 2. For each stream, including the default stream:

- a. Build a basic publish/subscribe message flow by copying and renaming the supplied publish/subscribe message flow (see “Adding an MQSeries Integrator broker as a leaf node” on page 165 for more details about this supplied flow).
- b. Check the properties of the nodes in each message flow you create. You must set the input queue name (the stream queue) property in the input node. You must also set the **implicitStreamNaming** property for every non-default stream queue input node.
- c. Finally, assign the message flow to an execution group of the broker `MQSI_SAMPLE_BROKER`, check in your changes, and deploy the broker.

### Migrating a broker network

The procedure you must follow to migrate an MQSeries Publish/Subscribe broker that is part of a multi-broker network is basically the same as that needed to migrate a single broker.

*MQSeries Integrator Introduction and Planning* gives guidance on planning how to approach migrating a network, including:

- The order in which you migrate the brokers.
- The place of each broker in the network, and the relationships it has with its neighbors.
- The use of collectives in the MQSeries Integrator network.

You must refer to this planning information before starting the migration of your network.

The following sequence of figures illustrates the migration of a network of three brokers. The actions taken to migrate the network assumes that the three brokers

## Migrating brokers

are migrated one at a time, and that all three will be grouped in a single collective in the MQSeries Integrator broker domain.

Figure 6 shows the initial state, with the three brokers (the root NEWYORK plus two child brokers LONDON and TOKYO) connected together as a network of MQSeries Publish/Subscribe brokers.

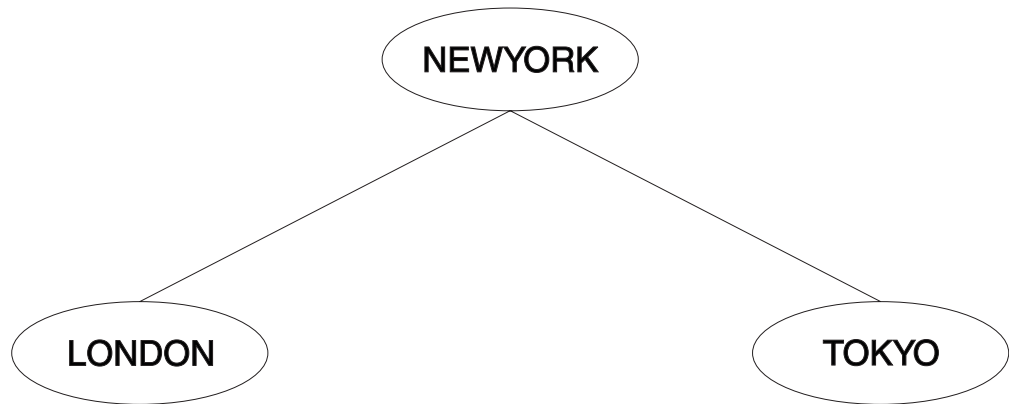


Figure 6. Migrating an MQSeries Publish/Subscribe broker network: initial state

These brokers do not have to be migrated in any particular order. This example shows the migration being done in the following order:

- LONDON
- NEWYORK
- TOKYO

The migration is completed in a number of separate steps, each one taken when network traffic is at a minimum (for example, at weekends). The whole migration is planned in three stages, that can be carried out when appropriate for the business.

### Stage 1: migration of the LONDON broker

The steps you need to take for migrating a single broker within a network are exactly the same as those you need to take for migrating a stand-alone MQSeries Publish/Subscribe broker. You can therefore implement the tasks described in “Migrating a single broker” on page 171 for the LONDON broker. In particular, review the points discussed in “Preparing for the migration” on page 171:

- You are recommended to quiesce all client applications at both the LONDON and NEWYORK brokers. This ensures that no publications are missed by any subscribers while the topology change is taking place.
- You are also recommended to quiesce all other brokers in the network (in this example, the TOKYO broker). This guarantees that no publications are delivered during the topology change.

This results in a mixed network, consisting of two MQSeries Publish/Subscribe brokers and a single MQSeries Integrator broker. This is shown in Figure 7 on page 175.

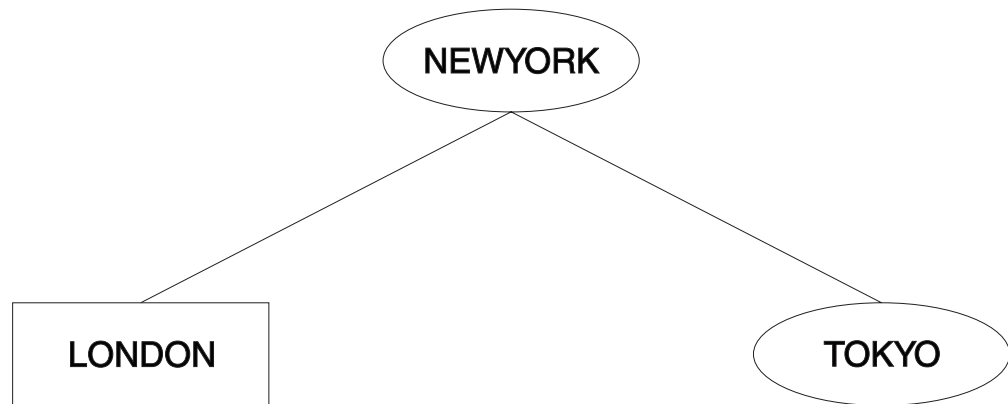


Figure 7. Migrating an MQSeries Publish/Subscribe broker network: first leaf node migration

The connection between the LONDON and NEWYORK brokers is an MQSeries Publish/Subscribe connection. The Control Center only recognizes MQSeries Integrator brokers, and therefore only LONDON has been defined to it. An MQSeries Integrator connection cannot be created at this stage.

This mixed network is in a perfectly valid state. It can be maintained in this state until the next stage can be implemented.

### Stage 2: migration of the NEWYORK broker

You can now follow the step-by-step procedure for migrating a single broker for broker NEWYORK. In particular, review the points discussed in “Preparing for the migration” on page 171:

- You are recommended to quiesce all client applications at all brokers to which NEWYORK is a neighbor (in this network, all brokers). This ensures that no publications are missed by any subscribers while the topology change is taking place.
- You are also recommended to quiesce all the brokers in the network. This guarantees that no publications are delivered during the topology change.

The network now contains two MQSeries Integrator brokers, with one remaining MQSeries Publish/Subscribe broker (TOKYO). This is shown in Figure 8.

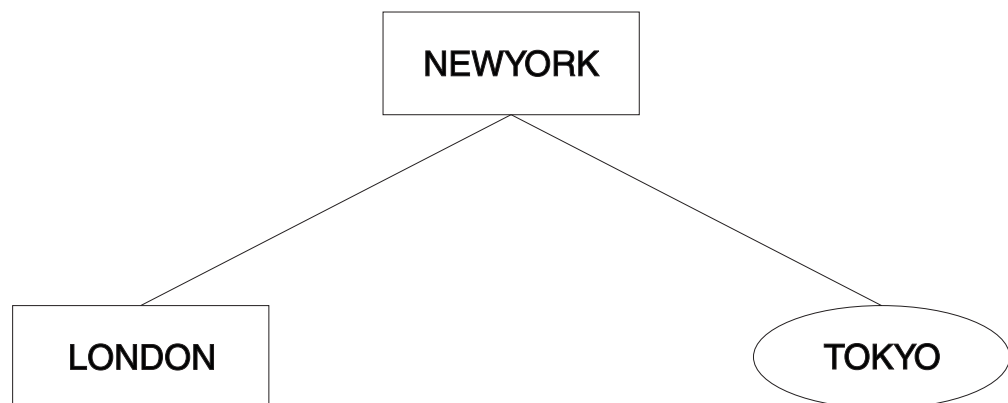


Figure 8. Migrating an MQSeries Publish/Subscribe broker network: root node migration

The LONDON and NEWYORK brokers are still connected by an MQSeries Publish/Subscribe connection. They can remain connected in this way for as long

## Migrating brokers

as necessary. However, as you begin to develop applications to exploit the richer function of MQSeries Integrator you will need to join the two MQSeries Integrator brokers together using the Control Center.

When appropriate, the connection can be upgraded to an MQSeries Integrator connection by completing the following tasks:

1. The original MQSeries Publish/Subscribe connection between LONDON and NEWYORK must be removed.

To the remove this connection, the MQSeries Integrator command **mqsiclearmqpubsub** is issued at both brokers, as follows:

```
mqsiclearmqpubsub NEWYORK -n LONDON
mqsiclearmqpubsub LONDON -n NEWYORK
```

After the connection has been broken, the network is temporarily in the state shown in Figure 9.

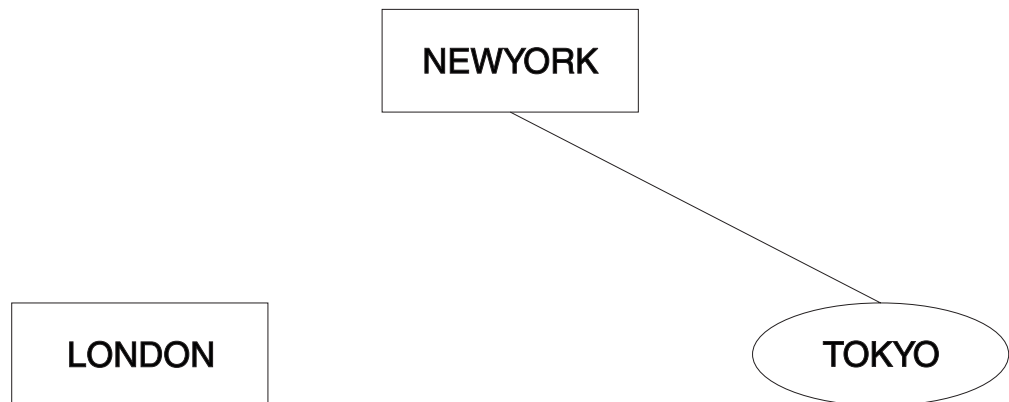


Figure 9. Migrating an MQSeries Publish/Subscribe broker network: breaking the connection

You must now define the relationship between the two brokers using the Control Center. Both brokers are already defined, but the collective to which they are to be assigned is not. You can define this collective from the *Topology* view, and assign the two brokers to it. All brokers in a collective are assumed to be connected, so you do not have to make those connections using the Control Center.

The new topology can now be deployed. The connection between LONDON and NEWYORK is now implemented using MQSeries Integrator functions. The current network state is shown in Figure 10 on page 177.



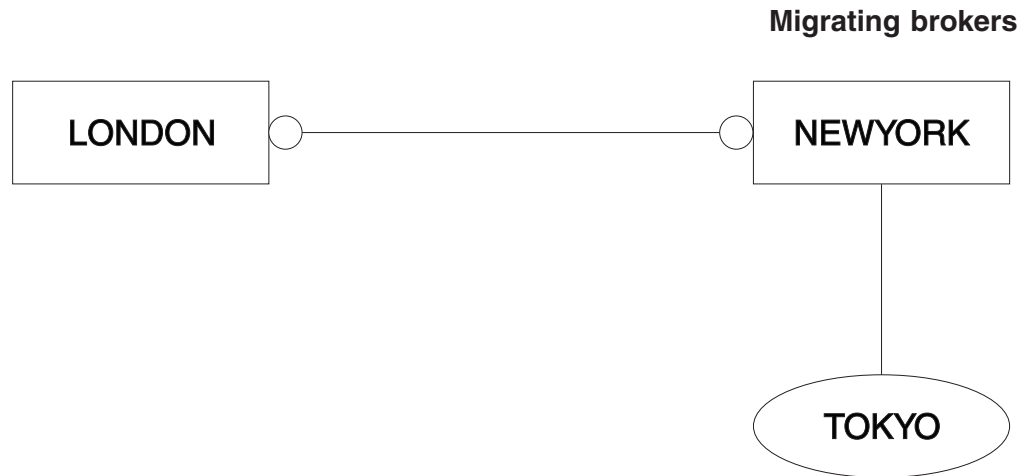


Figure 10. Migrating an MQSeries Publish/Subscribe broker network: rejoining

The two brokers, LONDON and NEWYORK, are no longer in a parent-child relationship but are neighbors within a single collective. The topology of the MQSeries Integrator network is not based on a hierarchical structure as was the MQSeries Publish/Subscribe network. Now that LONDON and NEWYORK form a collective, there is no root node left in the MQSeries Publish/Subscribe network. NEWYORK is providing a gateway service from an MQSeries Publish/Subscribe broker (TOKYO) to the MQSeries Integrator collective of brokers.

### Stage 3: migration of the TOKYO broker

The final MQSeries Publish/Subscribe broker, TOKYO, is now ready to be migrated. The procedure given in “Migrating a single broker” on page 171 is implemented for this third broker.

Following these actions, the network structure is as shown in Figure 11.



Figure 11. Migrating an MQSeries Publish/Subscribe broker network: second leaf node migration

When migration of the broker has completed successfully, the MQSeries Publish/Subscribe connection between TOKYO and NEWYORK can be broken. This must be done by clearing the connection at each end by issuing the following commands to brokers NEWYORK and TOKYO respectively:

```

mqsiclearmqpubsub NEWYORK -n TOKYO
mqsiclearmqpubsub TOKYO -n NEWYORK

```

## Migrating brokers

The TOKYO broker must now be added to the MQSeries Integrator network, and to the appropriate collective, through the Control Center. The operation of a collective requires that all brokers have direct physical connections with each other (via MQSeries).

Before the topology of the new MQSeries Integrator network can be deployed, an additional MQSeries connection between LONDON and TOKYO is required. A series of MQSeries commands must be invoked to define the channels and transmission queues supporting two-way traffic.

When you have completed migration of all the brokers in the collective, you have removed the single point of failure at the NEWYORK broker. Subscribers on the LONDON broker can receive publications from the TOKYO broker even when the NEWYORK broker is not running. Prior to migration, traffic between brokers was always routed through NEWYORK, the root node, which was therefore the single point of failure.

For further details of connecting brokers to each other, see “Chapter 2. How to configure your MQSeries Integrator network” on page 9. For more general information about distributed MQSeries networks, refer to *MQSeries Intercommunication*.

When all migration and associated tasks have been completed, the network comprises a single collective, containing three MQSeries Integrator brokers connected as equals. The final state for these three brokers is shown in Figure 12.

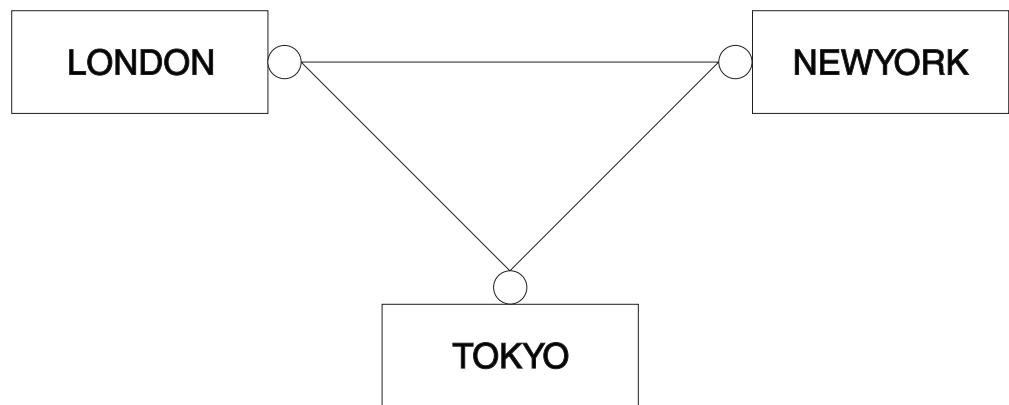


Figure 12. Migrating an MQSeries Publish/Subscribe broker network: final state

## A network of migrated brokers

Figure 13 on page 179 illustrates a possible extension of the small network that has been migrated in this chapter. It represents the ability to integrate a mixed network of brokers for both MQSeries Integrator and MQSeries Publish/Subscribe.

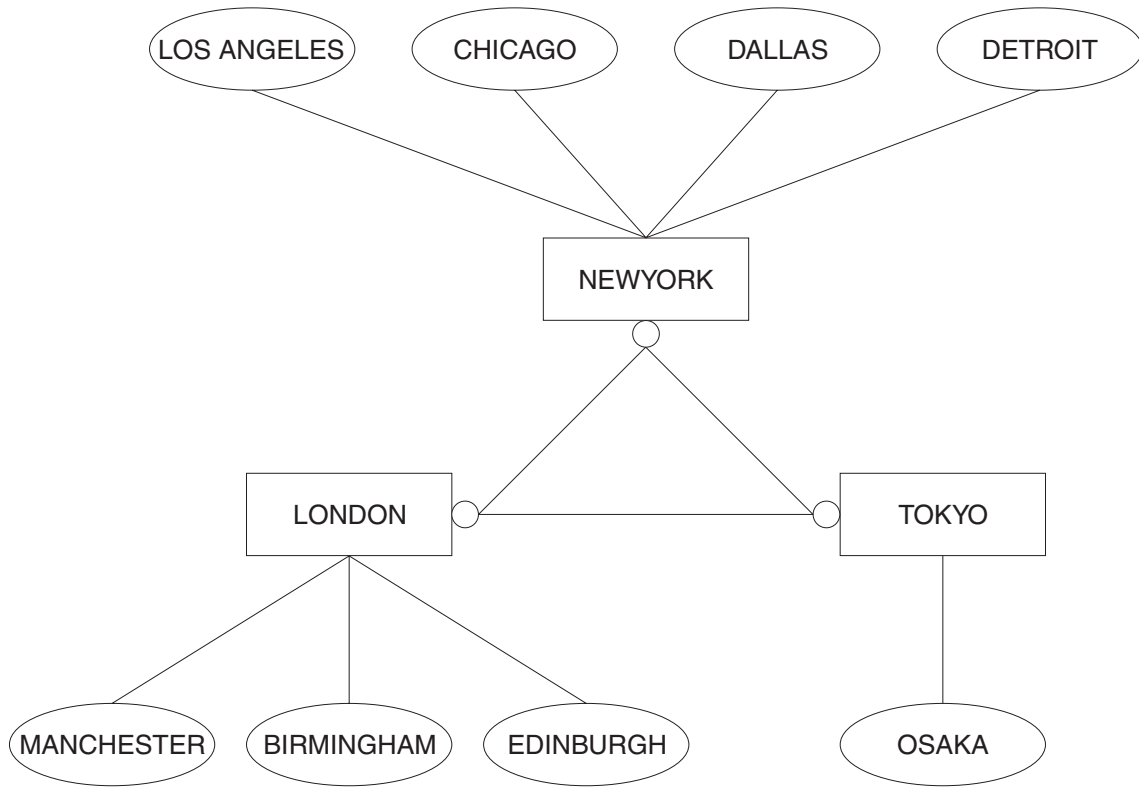


Figure 13. An integrated broker network

## Migrating brokers

---

## Part 4. Appendixes



---

## Appendix A. Event reporting

This chapter describes the event messages that are published by a message broker in response to:

- Configuration changes
- State changes
- User actions (such as subscription registrations, and when retained publications expire)

The events are published on a series of system-defined topics. The body of the message contains additional information in XML format. Every message is generated in code page 1208.

The following set of events can be reported:

- Configuration changes:
  - An execution group has been changed, created, or deleted
  - A message flow has been changed, created, or deleted
  - A neighbor has been created, changed, or deleted
  - An ACL for a topic has been created, changed, or deleted
- Operational information:
  - A broker has been started or stopped
  - A message flow has been started or stopped
  - A subscription has been registered or de-registered
- Operational warnings, for example:
  - A retained publication has expired
  - A subscription has expired

This chapter describes reserved topics on which brokers themselves publish messages after significant events within the broker. By subscribing to these topics, a client can be informed when events occur.

For each topic, the type of event and message body are explained. The body of these messages is in XML format.

An event publication can contain more than one entry if the topic is the same (for example, if several message flows are created in the same operation).

### General architecture

The general form of the system topics on which events are published is:  
\$SYS/Broker/<broker\_name>/<event\_type>/...

where:

**<broker\_name>**

Is the name of the broker issuing or raising this event.

**<event\_type>**

Is the type of the event and is one of:

- Configuration
- Neighbor
- Subscription
- Topic
- Status
- Expiry

This specification of topics helps clients to filter events, based on the broker from which the event originated and the type of the event. For specific events, additional information is included in the topic to help filter on the specific object that raised the event. (The inclusion of the string Broker at the second level of the topic hierarchy allows for future extension should other subsystems (such as the queue manager or configuration manager) that publish system management events through the broker be supported in the future.

---

## Configuration changes

Configuration changes include changes to the operational configuration of a single broker (for example, the addition or removal of a message flow) and changes to the topology for a multi-broker network.

### Changes to the local configuration of the broker

Notification of changes to the broker's configuration (create, change, or delete entities) is provided by publishing events on the following system topic:

\$SYS/Broker/<broker\_name>/Configuration/ExecutionGroup/<exec\_grp\_name>

where:

**<broker\_name>**

Is the name of the broker issuing this message.

**<exec\_grp\_name>**

Is the name of the execution group for which the configuration has changed.

One such event is published for each configuration request message that is received and processed by an execution group within the broker and can thus contain information that reflects complex configuration changes to multiple entities within the broker.

The body of each publication is that part of the configuration request that caused the event to be triggered. If an execution group is renamed, subsequent publications that report the state of that execution group will use the new name.

These events are published non-persistently as non-retained publications.



## Configuration changes

Only create, change, and delete actions on the message flow are reported.

### Configuration change

The following example shows a notification for when a message flow is created. The number of attributes mentioned in the example can vary.

```
<Broker uuid="1234" label="Broker1" version="1">
 <ExecutionGroup uuid="2345" >
 <Create>
 <MessageFlow uuid="3456" label="MessageFlow1">
 <!-- Create the Input and Output Nodes - -->
 <ComIbmMqInputNode uuid="4567"
 queueName="InputQueue1" label="InputNode1" />
 <ComIbmMqOutputNode uuid="5678"
 queueName="OutputQueue1"
 label="OutputNode1"/>
 <ComIbmMqOutputNode uuid="6789"
 queueManagerName="QueueManager1"
 queueName="OutputQueue2"
 label="OutputNode2"/>
 <!-- Create the filter - -->
 <ComIbmFilterNode uuid="7890"
 filterExpression="Company=IBM"
 label="FilterNode1"/>
 <!-- Connect them together - -->
 <Connection sourceNode="4567"
 sourceTerminal="out"
 targetNode="7890" targetTerminal="in"/>
 <Connection sourceNode="7890"
 sourceTerminal="true"
 targetNode="5678" targetTerminal="in"/>
 <Connection sourceNode="7890"
 sourceTerminal="false"
 targetNode="6789" targetTerminal="in"/>
 </MessageFlow>
 </Create>
 </ExecutionGroup>
</Broker>
```

### Neighbor changes

A change in the set of neighbors (the topology) for a given broker causes an event to be published using the following system topic:

```
$$SYS/Broker/<broker_name>/Neighbor
```

where <broker\_name> is the name of the broker issuing this message.

The body of each publication is an XML message that describes the change made.

These events are non-persistent, non-retained publications.

Certain operations cause the deletion of all neighbor (topology) information at a broker. In this case, the individual deleted neighbors are not published in the event publication. Instead, the body of the event publication contains a single XML tag which indicates that all neighbors have been deleted.

### Examples

Here are example event messages for when a neighbor is created, changed, and deleted.

## Configuration changes

### Neighbor created

Event publication topic = "\$SYS/Broker/Broker1/Neighbor"

```
<Broker uuid="1234" label="Broker1" version="1">
 <ControlGroup>
 <DynamicSubscriptionEngine>
 <Create>
 <Neighbor name="5678" collectiveId="">
 <MQBrokerConnection queueManagerName="nbr_QM_Name"/>
 </Neighbor>
 </Create>
 </DynamicSubscriptionEngine>
 </ControlGroup>
</Broker>
```

### Neighbor changed

Event publication topic = "\$SYS/Broker/Broker1/Neighbor"

```
<Broker uuid="1234" label="Broker1" version="1">
 <ControlGroup>
 <DynamicSubscriptionEngine>
 <Change>
 <Neighbor name="5678"
 collectiveId="12345678-1234-1234-1234-123456789abc"/>
 </Change>
 </DynamicSubscriptionEngine>
 </ControlGroup>
</Broker>
```

### Neighbor deleted

Event publication topic = "\$SYS/Broker/Broker1/Neighbor"

```
<Broker uuid="1234" label="Broker1" version="1">
 <ControlGroup>
 <DynamicSubscriptionEngine>
 <Delete>
 <Neighbor name="5678"/>
 </Delete>
 </DynamicSubscriptionEngine>
 </ControlGroup>
</Broker>
```

### All neighbors deleted

Event publication topic = "\$SYS/Broker/Broker1/Neighbor"

```
<Broker uuid="1234" label="Broker1" version="1">
 <ControlGroup>
 <DynamicSubscriptionEngine>
 <Delete>
 <AllNeighbors/>
 </Delete>
 </DynamicSubscriptionEngine>
 </ControlGroup>
</Broker>
```

## ACL updates

The creation, deletion, or modification of the ACL associated with a topic causes a publication using the following system topics:

```
$SYS/Broker/<broker_name>/Topic/<topic>
```

where:

**<broker\_name>**

Is the name of the broker issuing this message.

### <topic>

Is the topic whose ACL is being modified.

The body of each publication is an XML message that describes the ACL update.

These events are non-persistent, non-retained publications.

Certain operations cause the deletion of all ACL entries for a single topic or for all topics. In this case, the individual entries are not published in the event publication. Instead, the body of the event publication contains a single XML tag which indicates that all ACL entries have been deleted.

## Examples

Here are example event messages for when an ACL is created, changed, and deleted.

### ACL created

Event publication topic = "\$SYS/Broker/Broker1/Topic/stock/IBM"

```
<Broker uuid="1234" label="Broker1" version="1">
 <ControlGroup>
 <DynamicSubscriptionEngine>
 <Topic name="stock/IBM">
 <Create>
 <ACLEntry
 principalName="Fred"
 principalType="user"
 publish="false"
 subscribe="inherit"
 persistent="true"/>
 </Create>
 </Topic>
 </DynamicSubscriptionEngine>
 </ControlGroup>
</Broker>
```

### ACL changed

Event publication topic = "\$SYS/Broker/Broker1/Topic/stock/IBM"

```
<Broker uuid="1234" label="Broker1" version="1">
 <ControlGroup>
 <DynamicSubscriptionEngine>
 <Topic name="stock/IBM">
 <Change>
 <ACLEntry
 principalName="Fred"
 principalType="user"
 publish="true"
 subscribe="false"
 persistent="inherit"/>
 </Change>
 </Topic>
 </DynamicSubscriptionEngine>
 </ControlGroup>
</Broker>
```

### ACL deleted

Event publication topic = "\$SYS/Broker/Broker1/Topic/stock/IBM"

```
<Broker uuid="1234" label="Broker1" version="1">
 <ControlGroup>
 <DynamicSubscriptionEngine>
```

## Configuration changes

```
<Topic name="stock/IBM">
 <Delete>
 <ACLEntry principalName="Fred"/>
 </Delete>
</Topic>
</DynamicSubscriptionEngine>
</ControlGroup>
</Broker>
```

### All ACLs deleted on a single topic

Event publication topic = "\$SYS/Broker/Broker1/Topic/stock/IBM"

```
<Broker uuid="1234" label="Broker1" version="1">
 <ControlGroup>
 <DynamicSubscriptionEngine>
 <Topic name="stock/IBM">
 <Delete>
 <AllACLEntries/>
 </Delete>
 </Topic>
 </DynamicSubscriptionEngine>
 </ControlGroup>
</Broker>
```

### All ACLs deleted on all topics

Event publication topic = "\$SYS/Broker/Broker1/Topic"

```
<Broker uuid="1234" label="Broker1" version="1">
 <ControlGroup>
 <DynamicSubscriptionEngine>
 <Delete>
 <AllACLEntries/>
 </Delete>
 </DynamicSubscriptionEngine>
 </ControlGroup>
</Broker>
```

---

## Operational information

Changes to the execution state of a broker or an individual message flow cause events to be published using the following system topics:

\$SYS/Broker/<broker\_name>/Status

\$SYS/Broker/<broker\_name>/Status/ExecutionGroup/<exec\_grp\_name>

where:

**<broker\_name>**

Is the name of the broker whose execution state has changed.

**<exec\_grp\_name>**

Is the name of the execution group which contains the message flow whose execution state has changed.

The body of each publication is an XML message giving additional information concerning the state change which caused the event to be triggered, specifically indicating whether the entity has been started or stopped.

Thus, for example, starting a message flow will generate the following:

```
<Broker uuid="1234" label="Broker1" version="1">
<ExecutionGroup uuid="5678">
 <Start>
```

```

 <MessageFlow uuid="7812"/>
 </Start>
</ExecutionGroup>
</Broker>

```

Stopping a broker generates the following message body:

```

<Broker uuid="1234" label="Broker1" version="1">
 <StatusChange state="Stopped"/>
</Broker>

```

Currently, the only states notified for both brokers and message flows are Started and Stopped.

These events are non-persistent, retained publications.

## Subscriptions and topics

Events are published to provide notification of changes to the subscription tables, changes in the list of defined topics or their access control lists.

## Subscription registration and deregistration

Registration or deregistration of subscriptions causes events to be published using the following system topics:

```
$SYS/Broker/<broker_name>/Subscription/<topic>
```

where:

**<broker\_name>**

Is the name of the broker issuing this message.

**<topic>**

Is the original topic on which the subscription is being, or was, registered.

The body of each publication is an XML message giving additional information concerning the registration or deregistration request.

These events are non-persistent, non-retained publications.

## Examples

Here are example event messages for when a subscription is created, changed, deleted, and expired.

### Subscription created

Event publication topic = "\$SYS/Broker/Broker1/Subscription/stock/IBM"

```

<Broker uuid="1234" label="Broker1" version="1">
 <ControlGroup>
 <DynamicSubscriptionEngine>
 <Topic name="stock/IBM">
 <Create>
 <Subscription
 clientId="mqrh2:Broker1:client1queue"
 subscriptionPoint="poundsSterling"
 filter="currentPrice>100"
 user="Fred"
 persistent="true"
 localOnly="false"
 pubOnReqOnly="false"
 informIfRet="true"
 expiryTimeStamp="2000-12-31 23:59:59"
 >

```

## Operational information

```
 createTimeStamp="2000-01-01 00:00:00"
 tempDynamicQueue="false"
 clientContext="hex digits"/>
 </Create>
</Topic>
</DynamicSubscriptionEngine>
</ControlGroup>
</Broker>
```

### Subscription changed

Event publication topic = "\$SYS/Broker/Broker1/Subscription/stock/IBM"

```
<Broker uuid="1234" label="Broker1" version="1">
 <ControlGroup>
 <DynamicSubscriptionEngine>
 <Topic name="stock/IBM">
 <Change>
 <Subscription
 clientId="mqrfh2:Broker1:client1queue"
 subscriptionPoint="poundsSterling"
 filter="currentPrice>100"
 user="Fred"
 persistent="false"
 localOnly="true"
 pubOnReqOnly="true"
 informIfRet="false"
 expiryTimeStamp="2005-12-31 23:59:59"
 createTimeStamp="2000-01-01 00:00:00"
 tempDynamicQueue="false"
 clientContext="hex digits"/>
 </Change>
 </Topic>
 </DynamicSubscriptionEngine>
 </ControlGroup>
 </Broker>
```

### Subscription deleted (deregistered)

Event publication topic = "\$SYS/Broker/Broker1/Subscription/stock/IBM"

```
<Broker uuid="1234" label="Broker1" version="1">
 <ControlGroup>
 <DynamicSubscriptionEngine>
 <Topic name="stock/IBM">
 <Delete>
 <Subscription
 clientId="mqrfh2:Broker1:client1queue"
 subscriptionPoint="poundsSterling"
 filter="currentPrice>100"
 user="Fred"/>
 </Delete>
 </Topic>
 </DynamicSubscriptionEngine>
 </ControlGroup>
 </Broker>
```

## Operational warnings

Operational warnings provide notification about expired subscriptions or retained publications.

### Expired publications and subscriptions

Notification of expiry of retained publications and subscriptions is published on the following system topics:

```
$$SYS/Broker/<broker_name>/warning/expiry/Publication/<timestamp>/<topic>
```

```
$$SYS/Broker/<broker_name>/warning/expiry/Subscription/<timestamp>/<topic>
```

where:

**<broker\_name>**

Is the name of the broker issuing this message.

**<timestamp>**

Is the timestamp of the expiry of the subscription or retained publication (expressed as a GMT time).

**<topic>**

Is the topic of the subscription or retained publication.

These events are non-persistent, non-retained publications.

### Examples

Here are examples event message for when a retained publication and subscription have expired.

#### Publication expired

```
Event publication topic =
"$$SYS/Broker/Broker1/warning/expiry/Publication/2000-12-31
23:59:59/stock/IBM"
```

```
<Broker uuid="1234" label="Broker1" version="1">
 <ControlGroup>
 <DynamicSubscriptionEngine>
 <Topic name="stock/IBM">
 <Delete>
 <RetainedPublication
 subscriptionPoint="poundsSterling"/>
 </Delete>
 </Topic>
 </DynamicSubscriptionEngine>
 </ControlGroup>
</Broker>
```

#### Subscription expired

Here is an example event message for when a subscription has expired.

```
Event publication topic =
"$$SYS/Broker/Broker1/warning/expiry/Subscription/2000-12-31
23:59:59/stock/IBM"
```

```
<Broker uuid="1234" label="Broker1" version="1">
 <ControlGroup>
 <DynamicSubscriptionEngine>
 <Topic name="stock/IBM">
 <Delete>
 <Subscription
 clientId="mqrfh2:Broker1:client1queue"

```

## Operational warnings

```
 subscriptionPoint="poundsSterling"
 filter="currentPrice>100"
 user="Fred"/>
 </Delete>
</Topic>
</DynamicSubscriptionEngine>
</ControlGroup>
</Broker>
```

---

## Notification message schema

The following specification describes the structure of all valid notification messages. Note that this diagram describes the structure of the messages only. It says nothing about how many or the order of elements within the messages. The rules for the number of elements are:

- One broker element.
- Other elements: zero, one, or more.

There are no rules for order in notification messages.

```
<Broker identifier label>
. <ExecutionGroup identifier>
. . <Create>
. . . <MessageFlow message_flow_identifier message_flow_attributes>
. . . . <???Node node_identifier node_attributes>
. . . . <Connection connection_identifier>
. . <Change>
. . . <MessageFlow message_flow_identifier message_flow_attributes>
. . <Delete>
. . . <AllMessageFlows>
. . . <MessageFlow message_flow_identifier>
. . <Start>
. . . <AllMessageFlows>
. . . <MessageFlow message_flow_identifier>
. . <Stop>
. . . <AllMessageFlows>
. . . <MessageFlow message_flow_identifier>
. <ControlGroup>
. . <DynamicSubscriptionEngine>
. . . <Create>
. . . . <Neighbor neighbor_identifier neighbor_attributes>
. <MQBrokerConnection mqbrokerconnection_attributes>
. . . . <Neighbor neighbor_identifier neighbor_attributes>
. . . . <Topic topic_identifier>
. <ACLEntry>
. <Subscription>
. . . <Change>
. . . . <Neighbor neighbor_identifier neighbor_attributes>
. . . <Delete>
. . . . <AllNeighbors>
. . . . <Neighbor neighbor_identifier>
. . . . <Topic topic_identifier>
. . . . <AllACLEntries>
. . . . <AllSubscriptions>
. . . . <AllRetainedPublications>
. . . <Topic topic_identifier>
. . . . <Create>
. <ACLEntry acl_identifier acl_attributes>
. <Subscription subscription_identifier subscription_attributes>
. . . . <Change>
. <ACLEntry acl_identifier acl_attributes>
. <Subscription subscription_identifier subscription_attributes>
. . . . <Delete>
. <AllACLEntries>
. <ACLEntry acl_identifier>
```



```
. <AllSubscriptions>
. <Subscription subscription_identifier>
. <RetainedPublication retained_publication_identifier>
```

### Notes:

1. <...> = XML element
2. ??? = Individual class names allowed

The following section describes the identifiers and attributes that are used in the schema.

Items shown as [ ... ] are optional attributes. Items shown as {xxx | yyy} mean that the value can be one of the alternatives given. Items shown in *italics* mean that the variable can have any value.

#### uuid:

The universally unique identifier for MQSeries Integrator Version 2.0.2 objects (for example, nodes in a message flow)

#### client identifier:

Identifies the delivery destination for the subscriber (for example, mqrh2:MQSI\_SAMPLE\_QM:subscriberqueue or mqrh2:MQSI\_SAMPLE\_QM:subscriberqueue2:CorrelId)

#### identifier:

uuid="*uuid*"

#### label:

label="*label*"

#### message\_flow\_identifier:

uuid="*message\_flow\_identifier*" //Note: This is usually a uuid

#### message\_flow\_attributes:

```
[label='label']
[additionalInstances="number"]
[commitCount="number"]
[commitInterval"number"]
[coordinatedTransaction={"yes" | "no" }]
```

#### node\_identifier:

uuid='*node\_flow\_identifier*' //Note: This is usually a number of uuids concatenated

#### node\_attributes:

```
[label='label']
Others vary according to type. All are optional. See the node descriptions for details.
```

#### connection\_identifier:

```
sourceNode='source_node_identifier'
sourceTerminal='source_terminal_name'
targetNode='target_node_identifier'
targetTerminal='target_terminal_name'
```

#### neighbor\_identifier:

name="*uuid*"

#### neighbor\_attributes:

```
collectiveId={" " | "uuid" }
(collectiveId is " " if the neighbor is in the same collective as the broker; otherwise, it is the identifier for the neighbor's collective)
```

#### mqbrokerconnection\_attributes:

queueManagerName="*queueManagerName*"

## Message schema

```
topic_identifier:
 name="topicName"

acl_identifier:
 principalName="userIdentifier"

acl_attributes:
 principalType={"user" | "group"}
 [publish={"yes" | "no" | "inherit"}]
 [subscribe={"yes" | "no" | "inherit"}]
 [persistent={"yes" | "no" | "inherit"}]

subscription_identifier:
 clientId="client identifier"
 [subscriptionPoint="subscriptionPointName"]
 [filter="filterExpression"]

subscription_attributes:
 userId="userIdentifier"
 persistent={"true" | "false" | "asPublish" | "asQDef"}
 localOnly={"true" | "false"}
 pubOnReqOnly={"true" | "false"}
 informIfRet={"true" | "false"}
 expiryTimeStamp={"GMTTimeStamp" | "0" }
 createTimeStamp="GMTTimeStamp"
 tempDynamicQueue={"true" | "false"}
 clientContext="clientContext"

retainedpublication_identifier:
 [subscriptionPoint="subscriptionPointName"]
```

---

## Appendix B. Using NEON

The environment variable, `NEON_ROOT`, records the directory containing the NEON files for NEONRules and NEONFormatter Support for MQSeries Integrator. See “Setting the `NEON_ROOT` system environment variable” on page 197 for how this variable is set. In the instructions below, `NEON_ROOT` is used to indicate this directory.

You should refer to the *NEONRules and NEONFormatter Support for MQSeries Integrator System Management Guide* for further information, beyond the scope of this book.

---

### Installing database schema

Use the `inst_db` application to create the physical resources and schema required to successfully run NEONRules and NEONFormatter Support. It creates:

- A database configured for NEON rules and formats.
- An ODBC connection to that database.
- A `neonreg.dat` configuration file.

**Note:** If you are using a DB2 database, you must be logged on as the DB2 instance owner and have DB2ADM privilege (automatically granted if you are a member of NEONGRP) before you start the `inst_db` application.

To complete the parameters requested by `inst_db`, you will need to know the following information:

- The account to be used to create or refresh the database schema. This must be the instance owner if performing a new schema installation. If the schema already exists and a refresh is desired, this account can be the NEON objects owner.
- Password of the instance or objects owner.
- Database server name. This is usually the name of the host machine.
- If you are installing a DB2 database, the database server TCP/IP port number. This is the connection port for the DB2 server, set in the `/etc/services` file on UNIX platforms. The file is located in `WIN_HOME_DIR\system32\drivers\etc` on Windows NT. The default DB2 port for all platforms is 50000.
- Name of the database to be used for NEON rules and formats.
- Formatter literals client code set. This is the code set you wish NEONFormatter literal values to use. In most cases, you should accept the default value (the same code set as used by the host machine), but you can change it if the database being installed will be used by a remote machine using a different client code set.

To install the database schema, follow this procedure:

**Step 1.** Open a command window and configure it to be at least 80 wide by 60 high. If this configuration is not possible, add scrolling capability to the command window to provide access to the entire user interface.

**Step 2.**

On Windows NT, change to the `NEON_ROOT\install.sql_rulfmt52\inst_db` directory.

## Installing database schema

On UNIX platforms, change to the  
/NEON\_ROOT/install.sql\_rulfmt52/inst\_db directory.

**Note:** During inst\_db processing, individual component processing might fail.

If correction is not possible, abort the installation by typing -1 in any component processing screen. This action rolls back work performed by the inst\_db process and leaves the database server environment in a known state for subsequent inst\_db executions.

After Oracle and Sybase rollbacks, make sure you manually delete the created database files from the database install target directory before running inst\_db.

Step 3. Run the following command: inst\_db. The following screen appears:

```
=====
 NEON Database Installation
=====
Enter type of database (1 = DB2;
 2 = Microsoft 6.x;
 3 = Microsoft 7.x;
 4 = Oracle;
 5 = Sybase;
 -1 = Exit):

```

Step 4. Select the database type and press ENTER. The following screen appears:

```
=====
 NEON Database Installation
=====
Enter type of installation (1 = Basic;
 2 = Advanced;
 -1 = Exit):

```

Step 5. Select the type of installation and press ENTER:

- **Basic (1):** Performs the installation based on the database size that you select during the installation process. Basic mode allows for three database sizes: small (10MB), medium (20 MB) and large (60 MB).
- **Advanced (2):** Adjusts the database size based on options that you select throughout the installation process.

The appropriate database screen appears based on your database selection.

Step 6. Follow the procedures at the bottom of each installation screen. After you have completed the installation, continue with step 7 below.

**Note:** During the inst\_db installation, note the following additions to the online procedures:

- In the Database Owner Account window, do one of the following to set the database owner account:
  - To accept the default account (sa), leave the value blank.
  - To create a new account, enter a value other than (sa).
- In the Database User Name window, the database username account must be unique. It cannot be set to (sa) or to the database owner account that you created in the database owner account screen.

Step 7. When the installation successfully completes, the following message appears: *Installation completed SUCCESSFULLY.*

Step 8. To review installation details, view the `inst_db.log` file in the current directory.

**Note:** For DB2, it is recommended that you remove the DBADM privilege from NEONGRP after running the `inst_db` installation because the DBADM privilege is automatically granted to all users in NEONGRP.

---

### Setting the NEON\_ROOT system environment variable

On Windows NT, the User environment variable `NEON_ROOT` is set automatically during the installation process to run NEONRules and NEONFormatter Support. You can also set it manually. Set the variable to the path where NEONRules and NEONFormatter Support components are installed; typically `C:\neonsoft`.

On UNIX platforms, the environment variable `NEON_ROOT` must be set to run NEONRules and NEONFormatter Support. Either add the following to your profile, or enter the information on the command line each time you start an executable:

```
export NEON_ROOT=/opt/neonsoft (On AIX: export NEON_ROOT=/usr/lpp/neonsoft .)
```

---

### Access to Rules and Formats

NEON Rules and Formats are not deployed to individual brokers. Instead, each machine on which a broker requiring access to the NEON Rules and Formats resides must be configured so that it can locate the NEON Rules and Formats database.

This is achieved by setting the `NN_CONFIG_FILE_PATH` environment variable to identify the directory in which the configuration file for the new NEONRules and NEONFormatter Support package has been placed. (Note that `NN_CONFIG_FILE_PATH` should point to a directory, not the file itself.) This configuration file must be named `neonreg.dat` (see “Editing the database configuration file (`neonreg.dat`)”). A sample `neonreg.dat` file for the new package, which can be used for reference when editing the working file, is included in the `<mqsi_root>\examples` directory on Windows NT, or the `<mqsi_root>/sample/NEON` directory on UNIX platforms.

---

### Editing the database configuration file (`neonreg.dat`)

The `inst_db` process creates a `neonreg.dat` file in the `NEON_ROOT\install.sql_rulfmt52\inst_db` directory, using the data given as input to `inst_db`. For example (passwords are shown in the file):

```
Session.sys_inst_db
NNOT_SHARED_LIBRARY = dbt22db250
NNOT_FACTORY_FUNCTION = NNSesDB2Factory
NN_SES_SERVER = NEWNEON
NN_SES_USER_ID = mqsiuid
NN_SES_PASSWORD = mqsipw
Session.inst_db
NNOT_SHARED_LIBRARY = dbt22db250
NNOT_FACTORY_FUNCTION = NNSesDB2Factory
NN_SES_SERVER = NEWNEON
NN_SES_USER_ID = mqsiuid
NN_SES_PASSWORD = mqsipw
```

## Editing neonreg.dat

You need to extend this file to include sessions for MQSI\_CONFIG for the configuration manager and MQSI\_PLUGIN for the NEON nodes. Apart from the names of the sessions, all details are the same as those that already exist in the neonreg.dat file:

```
Session.MQSI_CONFIG
NNOT_SHARED_LIBRARY = dbt22db250
NNOT_FACTORY_FUNCTION = NNSesDB2Factory
NN_SES_SERVER = NEWNEON
NN_SES_USER_ID = mqsiuid
NN_SES_PASSWORD = mqsipw
Session.MQSI_PLUGIN
NNOT_SHARED_LIBRARY = dbt22db250
NNOT_FACTORY_FUNCTION = NNSesDB2Factory
NN_SES_SERVER = NEWNEON
NN_SES_USER_ID = mqsiuid
NN_SES_PASSWORD = mqsipw
```

Some NEON executables also require sessions. For example, msgtest requires a session called new\_format\_demo. If you plan to use these executables, you must add the necessary sessions to neonreg.dat (again, all details are the same apart from the names of the sessions):

```
Session.new_format_demo
Session.nnr1e
Session.nnf1e
Session.nnrmie
```

---

## Importing rules and formats

To import files, ensure that the NNFie and NNRie commands are in the executable path and that the NN\_CONFIG\_FILE\_PATH environment variable (see above) is set. You should also ensure that all the sessions previously described have been defined in neonreg.dat.

To import formats, run NNFie. For example, NNFie -i formats.exp.

To import rules, run NNRie. For example, NNRie -i rules.exp.

Some online help is available for the import utilities by entering the commands with no additional parameters.

---

## Appendix C. 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

## 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.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.



---

## Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AIX	DB2	DB2 Universal Database
FFST	First Failure Support Technology	IBM
MQSeries	SupportPac	VisualAge

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

Other company, product, and service names may be trademarks or service marks of others.



---

## Glossary of terms and abbreviations

This glossary defines MQSeries Integrator terms and abbreviations used in this book. If you do not find the term you are looking for, see the index or the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

This glossary includes terms and definitions from the *American National Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute. Copies may be ordered 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

**Access Control List (ACL).** The list of principals that have explicit permissions (to publish, to subscribe to, and to request persistent delivery of a publication message) against a topic in the topic tree. The ACLs define the implementation of topic-based security.

**ACL.** Access Control List.

**AMI.** Application Messaging Interface.

**Application Messaging Interface (AMI).** The programming interface provided by MQSeries that defines a high level interface to message queuing services. See also *MQI* and *JMS*.

### B

**blob.** Binary Large Object. A block of bytes of data (for example, the body of a message) that has no discernible meaning, but is treated as one solid entity that cannot be interpreted. Also written as BLOB.

**broker.** See *message broker*.

**broker domain.** A collection of brokers that share a common configuration, together with the single Configuration Manager that controls them.

### C

**callback function.** See *implementation function*.

**category.** An optional grouping of messages that are related in some way. For example, messages that relate to a particular application.

**check in.** The Control Center action that stores a new or updated resource in the configuration or message repository.

**check out.** The Control Center action that extracts and locks a resource from the configuration or message repository for local modification by a user. Resources from the two repositories can only be worked on when they are checked out by an authorized user, but can be viewed (read only) without being checked out.

**collective.** A hyperconnected (totally connected) set of brokers forming part of a multi-broker network for publish/subscribe applications.

**configuration.** In the broker domain, the brokers, execution groups, message flows and message sets assigned to them, topics and access control specifications.

**Configuration Manager.** A component of MQSeries Integrator that acts as the interface between the configuration repository and an executing set of brokers. It provides brokers with their initial configuration, and updates them with any subsequent changes. It maintains the broker domain configuration.

**configuration repository.** Persistent storage for broker configuration and topology definition.

**connector.** See *message processing node connector*.

**content-based filter.** An expression that is applied to the content of a message to determine how the message is to be processed.

**context tag.** A tag that is applied to an element within a message to enable that element to be treated differently in different contexts. For example, an element could be mandatory in one context and optional in another.

**Control Center.** The graphical interface that provides facilities for defining, configuring, deploying, and monitoring resources of the MQSeries Integrator network.

### D

**datagram.** The simplest form of message that MQSeries supports. Also known as *send-and-forget*. This type of message does not require a reply. Compare with *request/reply*.

| **debugger.** A facility on the *Message Flows* view in the  
| Control Center that enables message flows to be  
| debugged.

## Glossary

**deploy.** Make operational the configuration and topology of the broker domain.

**destination list.** A list of internal and external destinations to which a message is sent. These can be nodes within a message flow (for example, when using the RouteToLabel and Label nodes) or MQSeries queues (when the list is examined by an MQOutput node to determine the final target for the message).

**distribution list.** A list of MQSeries queues to which a message can be put using a single statement.

**Document Type Definition (DTD).** The rules that specify the structure for a particular class of SGML or XML documents. The DTD defines the structure with elements, attributes, and notations, and it establishes constraints for how each element, attribute, and notation can be used within the particular class of documents. A DTD is analogous to a database schema in that the DTD completely describes the structure for a particular markup language.

**DTD.** Document Type Definition

## E

**e-business.** A term describing the commercial use of the Internet and World Wide Web to conduct business (short for electronic-business).

**element.** A unit of data within a message that has business meaning, for example, street name

**element qualifier.** See *context tag*.

**ESQL.** Extended SQL. A specialized set of SQL statements based on regular SQL, but extended with statements that provide specialized functions unique to MQSeries Integrator.

**exception list.** A list of exceptions that have been generated during the processing of a message, with supporting information.

**execution group.** A named grouping of message flows that have been assigned to a broker. The broker is guaranteed to enforce some degree of isolation between message flows in distinct execution groups by ensuring that they execute in separate address spaces, or as unique processes.

**Extensible Markup Language (XML).** A W3C standard for the representation of data.

| **external reference.** A reference within a message set to  
| a component that has been defined outside the current  
| message set. For example, an integer that defines the  
| length of a string element might be defined in one  
| message set but used in several message sets.

## F

| **field reference.** A sequence of period-separated values  
| that identify a specific field (which might be a  
| structure) within a message tree. An example of a field  
| reference might be something like  
| `Body.Invoice.InvoiceNo`.

**filter.** An expression that is applied to the content of a message to determine how the message is to be processed.

**format.** A format defines the internal structure of a message, in terms of the fields and order of those fields. A format can be self-defining, in which case the message is interpreted dynamically when read.

## G

**graphical user interface (GUI).** An interface to a software product that is graphical rather than textual. It refers to window-based operational characteristics.

## I

**implementation function.** Function written by a third-party developer for a plug-in node or parser. Also known as a *callback function*.

**input node.** A message flow node that represents a source of messages for the message flow.

**installation mode.** The installation mode can be Full, Custom, or Broker only. The mode defines the components of the product installed by the installation process on Windows NT systems.

## J

**Java Database Connectivity (JDBC).** An application programming interface that has the same characteristics as ODBC but is specifically designed for use by Java database applications.

**Java Development Kit (JDK).** A software package that can be used to write, compile, debug, and run Java applets and applications.

**Java Message Service (JMS).** An application programming interface that provides Java language functions for handling messages.

**Java Runtime Environment (JRE).** A subset of the Java Development Kit (JDK) that contains the core executables and files that constitute the standard Java platform. The JRE includes the Java Virtual Machine, core classes and supporting files.

**JDBC.** Java Database Connectivity.

**JDK™.** Java Development Kit.

**JMS.** Java Message Service. See also *AMI* and *MQI*.

**JRE.** Java Runtime Environment.

## L

**local error log.** A generic term that refers to the logs to which MQSeries Integrator writes records on the local system. On Windows NT, this is the Event log. On UNIX systems, this is the syslog. See also *system log*. Note that MQSeries records many events in the log that are not errors, but information about events that occur during operation, for example, successful deployment of a configuration.

## M

**message broker.** A set of execution processes hosting one or more message flows.

**messages.** Entities exchanged between a broker and its clients.

**message dictionary.** A repository for (predefined) message type specifications.

| **message domain.** The value that determines how the  
| message is interpreted (parsed). The following domains  
| are recognized:  
| • MRM, which identifies messages defined using the  
| Control Center  
| • NEONMSG<sup>1</sup>, which identifies messages created using  
| the NEONFORMATTER user interfaces.  
| • XML, which identifies messages that are self-defining  
| • BLOB, which identifies messages that are undefined

| You can also create your own message domains: if you  
| do so, you must supply your own message parser.

**message flow.** A directed graph that represents the set of activities performed on a message or event as it passes through a broker. A message flow consists of a set of message processing nodes and message processing node connectors.

**message flow component.** See *message flow*.

**message parser.** A program that interprets a message bitstream.

**message processing node.** A node in the message flow, representing a well defined processing stage. A message processing node can be one of several primitive types or can represent a subflow.

**message processing node connector.** An entity that connects the output terminal of one message processing node to the input terminal of another. A message

processing node connector represents the flow of control and data between two message flow nodes.

**message queue interface (MQI).** The programming interface provided by MQSeries queue managers. The programming interface allows application programs to access message queuing services. See also *AMI* and *JMS*.

**message repository.** A database holding message template definitions.

| **message repository manager (MRM).** A component of  
| the Configuration Manager that handles message  
| definition and control. A message defined to the MRM  
| has a message domain set to MRM.

**message set.** A grouping of related messages.

**message template.** A named and managed entity that represents the format of a particular message. Message templates represent a business asset of an organization.

**message type.** The logical structure of the data within a message. For example, the number and location of character strings.

**metadata.** Data that describes the characteristic of stored data.

| **MQe.** MQSeries Everyplace.

**MQI.** Message queue interface.

| **MQIsdp.** MQSeries Integrator SCADA device  
| protocol. A lightweight publish/subscribe protocol  
| flowing over TCP/IP.

**MQRFH.** An architected message header that is used to provide metadata for the processing of a message. This header is supported by MQSeries Publish/Subscribe.

**MQRFH2.** An extended version of MQRFH, providing enhanced function in message processing.

| **MQSeries Everyplace.** A generally available MQSeries  
| product that provides proven MQSeries reliability and  
| security in a mobile environment.

| **MRM.** Message Repository Manager.

**multilevel wildcard.** A wildcard that can be specified in subscriptions to match any number of levels in a topic.

## N

**node.** See *message processing node*.

## O

**ODBC.** Open Database Connectivity.

---

1. The message domain NEON is also recognized for compatibility with previous releases.

## Glossary

**Open Database Connectivity.** A standard application programming interface (API) for accessing data in both relational and non-relational database management systems. Using this API, database applications can access data stored in database management systems on a variety of computers even if each database management system uses a different data storage format and programming interface. ODBC is based on the call level interface (CLI) specification of the X/Open SQL Access Group.

**output node.** A message processing node that represents a point at which messages flow out of the message flow.

## P

**plug-in.** An extension to the broker, written by a third-party developer, to provide a new message processing node or message parser in addition to those supplied with the product. See also *implementation function* and *utility function*.

**point-to-point.** Style of messaging application in which the sending application knows the destination of the message. Compare with *publish/subscribe*.

**POSIX.** Portable Operating System Interface For Computer Environments. An IEEE standard for computer operating systems (for example, AIX and Sun Solaris).

**predefined message.** A message with a structure that is defined before the message is created or referenced. Compare with *self-defining message*.

**primitive.** A message processing node that is supplied with the product.

**principal.** An individual user ID (for example, a log-in ID) or a group. A group can contain individual user IDs and other groups, to the level of nesting supported by the underlying facility.

**property.** One of a set of characteristics that define the values and behaviors of objects in the Control Center. For example, message processing nodes and deployed message flows have properties.

**publication node.** An end point of a specific path through a message flow to which a client application subscribes. A publication node has an attribute, subscription point. If this is not specified, the publication node represents the default subscription point for the message flow.

**publish/subscribe.** Style of messaging application in which the providers of information (publishers) are decoupled from the consumers of that information (subscribers) using a broker. Compare with *point-to-point*. See also *topic*.

**publisher.** An application that makes information about a specified topic available to a broker in a publish/subscribe system.

## 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.** A system program that provides queuing services to applications. It provides an application programming interface (the MQI) so that programs can access messages on the queues that the queue manager owns.

## R

**retained publication.** A published message that is kept at the broker for propagation to clients that subscribe at some point in the future.

**request/reply.** Type of messaging application in which a request message is used to request a reply from another application. Compare with *datagram*.

**rule.** A rule is a definition of a process, or set of processes, applied to a message on receipt by the broker. Rules are defined on a message format basis, so any message of a particular format will be subjected to the same set of rules.

## S

| **SCADA.** Supervisory, Control, And Data Acquisition.

**self-defining message.** A message that defines its structure within its content. For example, a message coded in XML is self-defining. Compare with *pre-defined message*.

**send and forget.** See *datagram*.

**setup type.** The definition of the type of installation requested on Windows NT systems. This can be one of **Full, Broker only**, or **Custom**.

**shared.** All configuration data that is shared by users of the Control Center. This data is not operational until it has been deployed.

**signature.** The definition of the external characteristics of a message processing node.

**single-level wildcard.** A wildcard that can be specified in subscriptions to match a single level in a topic.

**stream.** A method of topic partitioning used by MQSeries Publish/Subscribe applications.

**subscriber.** An application that requests information about a specified topic from a publish/subscribe broker.

**subscription.** Information held within a publication node, that records the details of a subscriber application, including the identity of the queue on which that subscriber wants to receive relevant publications.

**subscription filter.** A predicate that specifies a subset of messages to be delivered to a particular subscriber.

**subscription point.** An attribute of a publication node that differentiates it from other publication nodes on the same message flow and therefore represents a specific path through the message flow. An unnamed publication node (that is, one without a specific subscription point) is known as the default publication node.

**Supervisory, Control, And Data Acquisition.** A broad term, used to describe any form of remote telemetry system used for gathering data from remote sensor devices (for example, flow rate meters on an oil pipeline) and for the near real time control of remote equipment (for example, pipeline valves).

**system log.** A generic term used in the MQSeries Integrator messages (BIPxxx) that refers to the local error logs to which records are written on the local system. On Windows NT, this is the Event log. On UNIX systems, this is the syslog. See also *local error log*.

## T

**terminal.** The point at which one node in a message flow is connected to another node. Terminals enable you to control the route that a message takes, depending whether the operation performed by a node on that message is successful.

**topic.** A character string that describes the nature of the data that is being published in a publish/subscribe system.

**topic based subscription.** A subscription specified by a subscribing application that includes a topic for filtering of publications.

**topic security.** The use of ACLs applied to one or more topics to control subscriber access to published messages.

**topology.** In the broker domain, the brokers, collectives, and connections between them.

**transform.** A defined way in which a message of one format is converted into one or more messages of another format.

## U

**Uniform Resource Identifier.** The generic set of all names and addresses that refer to World Wide Web resources.

**Uniform Resource Locator.** A specific form of URI that identifies the address of an item on the World Wide Web. It includes the protocol followed by the fully qualified domain name (sometimes called the host name) and the request. The Web server typically maps the request portion of the URL to a path and file name. Also known as Universal Resource Locator.

**URI.** Uniform Resource Identifier

**URL.** Uniform Resource Locator

**User Name Server.** The MQSeries Integrator component that interfaces with operating system facilities to determine valid users and groups.

**utility function.** Function provided by MQSeries Integrator for the benefit of third-party developers writing plug-in nodes or parsers.

## W

**warehouse.** A persistent, historical datastore for events (or messages). The **Warehouse** node within a message flow supports the recording of information in a database for subsequent retrieval and processing by other applications.

**wildcard.** A character that can be specified in subscriptions to match a range of topics. See also *multilevel wildcard* and *single-level wildcard*.

**wire format.** This describes the physical representation of a message within the bit-stream.

**W3C.** World Wide Web Consortium. An international industry consortium set up to develop common protocols to promote evolution and interoperability of the World Wide Web.

## X

**XML.** Extensible Markup Language.

## Glossary



---

## Bibliography

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

---

### MQSeries Integrator Version 2.0.2 cross-platform publications

The MQSeries Integrator cross-platform publications are:

- *MQSeries Integrator Introduction and Planning*, GC34-5599
- *MQSeries Integrator Using the Control Center*, GC34-5602
- *MQSeries Integrator Messages*, GC34-5601
- *MQSeries Integrator Programming Guide*, SC34-5603
- *MQSeries Integrator Administration Guide*, SC34-5792
- *MQSeries Integrator ESQL Reference*, SC34-5923

These books are all available in hardcopy.

You can order publications from the IBMLink™ Web site at:

<http://www.ibm.com/ibmlink>

In the United States, you can also order publications by dialing 1-800-879-2755.

In Canada, you can order publications by dialing 1-800-IBM-4YOU (1-800-426-4968).

For further information about ordering publications contact your IBM authorized dealer or marketing representative.

---

### MQSeries Integrator Version 2.0.2 platform-specific publications

| Each MQSeries Integrator product provides one  
| platform-specific installation guide, which is  
| supplied in hardcopy.

#### MQSeries Integrator for AIX Version 2.0.2

*MQSeries Integrator for AIX Installation Guide*, GC34-5841

#### MQSeries Integrator for HP-UX Version 2.0.2

*MQSeries Integrator for HP-UX Installation Guide*, GC34-5907

#### MQSeries Integrator for Sun Solaris Version 2.0.2

*MQSeries Integrator for Sun Solaris Installation Guide*, GC34-5842

#### MQSeries Integrator for Windows NT Version 2.0.2

*MQSeries Integrator for Windows NT Installation Guide*, GC34-5600

---

### MQSeries Everyplace publications

| If you intend to connect MQSeries Everyplace  
| applications to message flows that include the  
| MQSeries Everyplace message flow nodes, you  
| will find the following publications useful:

- | • *MQSeries Everyplace for Multiplatforms Version 1.1 Introduction*, GC34-5843
- | • *MQSeries Everyplace for Multiplatforms Version 1.1 Programming Guide*, SC34-5845
- | • *MQSeries Everyplace for Multiplatforms Version 1.1 Programming Reference*, SC34-5846
- | • *MQSeries Everyplace for Multiplatforms Version 1.1 Native Client Information*, SC34-5880

| You can find these books on the MQSeries Web  
| site (see “MQSeries information available on the  
| Internet” on page 212). Translated versions of  
| these books are also available in some languages  
| from the same Web site.

---

### NEONRules and NEONFormatter Support for MQSeries Integrator publications

| The following publications are supplied on the  
| product CD in PDF format, and are installed with  
| the Documentation component.

- *NEONRules and NEONFormatter Support for MQSeries Integrator User's Guide*
- *NEONRules and NEONFormatter Support for MQSeries Integrator System Management Guide*

## Bibliography

- *NEONRules and NEONFormatter Support for MQSeries Integrator Programming Reference for NEONRules*
- *NEONRules and NEONFormatter Support for MQSeries Integrator Programming Reference for NEONFormatter*
- *NEONRules and NEONFormatter Support for MQSeries Integrator Application Development Guide*

These books are provided in US English only.

## Softcopy books

All the MQSeries Integrator books are available in softcopy formats.

## Portable Document Format (PDF)

All books in the MQSeries Integrator library are supplied in US English only in a searchable PDF library on the product CD.

You can install the library as follows:

- On AIX, invoke `install -d` and select the documentation fileset. After installation, run the command `mqsdocs`. This launches Acrobat Reader and opens the PDF package.
- On HP-UX, invoke `swinstall -d` and select `MQSI-DOCS` from the menu. After installation, run the command `mqsdocs`. This launches Acrobat Reader and opens the PDF package.
- On Sun Solaris, invoke `pkgadd -d` and select `mqs-docs` from the menu. After installation, run the command `mqsdocs`. This launches Acrobat Reader and opens the PDF package.
- On Windows NT, select the Online Documentation component on a custom installation, or do a full installation. After installation, select *Start*—>*Programs*—>*IBM MQSeries Integrator 2.0*—>*Documentation*.

In addition, PDF files for books that have been translated are installed into the location `mqs_i_root/bin/book/pdf/<locale>` (on UNIX) or `mqs_i_root\bin\book\pdf\<locale>` (on Windows NT) where `<locale>` is one of the following:

- **de\_DE** for German
- **en\_US** for US English
- **es\_ES** for Spanish
- **fr\_FR** for French
- **it\_IT** for Italian
- **ja\_JP** for Japanese
- **ko\_KR** for Korean
- **pt\_BR** for Brazilian Portuguese

- **zh\_CN** for Simplified Chinese
- **zh\_TW** for Traditional Chinese

An index file (in HTML format) that provides a link to each book is supplied for each language. For example, the French index file is called `indexfr.htm`. The files are stored in the following directory:

- On UNIX, `<mqs_i_root>/docs/`
- On Windows NT, `<mqs_i_root>\bin\book`

Each index file has an entry for every book: if a particular book has not been translated into the appropriate language for that index file, a link to the English PDF is included. You can use any Web browser to view the index file. On Windows NT, you can also access the index file through the *Start* menu.

The PDF file names for the English books are shown in Table 7.

Table 7. File names of MQSeries Integrator book PDFs

Book title	File name
<i>MQSeries Integrator for AIX Installation Guide</i>	bipaac04.pdf
<i>MQSeries Integrator for HP-UX Installation Guide</i>	bipcac00.pdf
<i>MQSeries Integrator for Sun Solaris Installation Guide</i>	bip7ac03.pdf
<i>MQSeries Integrator for Windows NT Installation Guide</i>	bipyac03.pdf
<i>MQSeries Integrator Introduction and Planning</i>	bipyab02.pdf
<i>MQSeries Integrator Administration Guide</i>	bipyag04.pdf
<i>MQSeries Integrator Using the Control Center</i>	bipyar03.pdf
<i>MQSeries Integrator ESQL Reference</i>	bipyae00.pdf
<i>MQSeries Integrator Programming Guide</i>	bipyal02.pdf
<i>MQSeries Integrator Messages</i>	bipyao02.pdf

The fifth character of the file name indicates the language of the book (**a** indicates US English). You can deduce the file names of translated books by using the following substitutions for the fifth character:

- **g** for German
- **s** for Spanish
- **f** for French
- **i** for Italian

- | • **j** for Japanese
- | • **k** for Korean
- | • **b** for Brazilian Portuguese
- | • **z** for Simplified Chinese
- | • **t** for Traditional Chinese

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

- | If you cut and paste examples of commands from
- | PDF files to a command line for execution, you
- | must check that the content is correct before you
- | press Enter. Some characters might be corrupted
- | by local system and font settings.

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 all current MQSeries Integrator books are also available from the MQSeries product family Web site at:

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

---

## MQSeries publications

The following books are referenced in this book to point you to the information you need to complete MQSeries Messaging product tasks as part of MQSeries Integrator tasks.

- *MQSeries Planning Guide*, GC33-1349.  
This book describes some key MQSeries concepts, and discusses items that must be considered before MQSeries is installed.
- *MQSeries System Administration*, SC33-1873.  
This book supports day-to-day management of local and remote MQSeries objects.
- *MQSeries Programmable System Management*, SC33-1482.  
This book provides reference and guidance information for users of MQSeries events, programmable command formats (PCFs), and installable services.
- *MQSeries Intercommunication*, SC33-1872.  
This book defines the concepts of distributed queuing and explains how to set up a distributed queueing network.
- *MQSeries Clients*, GC33-1632.  
This book describes how to install, configure, use, and manage MQSeries clients.

- *MQSeries Command Reference*, SC33-1369.  
This book contains the syntax of the MQSC commands.
- *MQSeries Application Programming Reference*, SC33-1673.  
This book provides comprehensive reference information, including MQSeries return codes.
- *MQSeries Application Programming Guide*, SC33-0807.  
This book provides guidance information for programmers.

For a complete list of MQSeries Messaging product publications, refer to the information on the MQSeries Web site (identified in “MQSeries information available on the Internet” on page 212).

---

## MQSeries Publish/Subscribe publications

If you have installed MQSeries Publish/Subscribe and plan to migrate to MQSeries Integrator Version 2, or to establish a mixed broker network, refer to the following publication:

- *MQSeries Publish/Subscribe User's Guide*, GC34-5269

This book and the MQSeries Publish/Subscribe SupportPac package are available from the MQSeries Web site (identified in “MQSeries information available on the Internet” on page 212).

---

## MQSeries Workflow publications

The MQSeries Workflow product has a comprehensive library. Refer to the following book for introductory information, and for details about other product publications:

- *MQSeries Workflow Concepts and Architecture*, GH12-6285

For a complete list of MQSeries Workflow publications, refer to the information on the MQSeries Web site (identified in “MQSeries information available on the Internet” on page 212).

## Bibliography

---

### DB2 publications

If you want more information about DB2, you can download the product publications from the DB2 Web site at

<http://www.ibm.com/software/data/db2>

---

### MQSeries information available on the Internet

The MQSeries product family Web site is at:

<http://www.ibm.com/software/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.
- | • Obtain information about complementary offerings by following these links:
  - | – IBM Business Partners
  - | – Partner Offerings (within *Related links*)
- Download an MQSeries SupportPac.

---

# Index

## Special Characters

\$SYS/Broker 184

## A

about this book xi  
ACL update event messages 186  
administration  
  broker domain 4  
  overview 3  
  problem determination 4  
  system management 5  
  tasks 3  
application clients 29  
  MQSeries resources 30  
architecture of event messages 184  
authorization  
  DB2 database 19  
  SQL Server database 20  
authorization tasks, networks 9

## B

backout processing 44  
  SCADA 45  
backup and recovery 47  
BIP messages 74  
blanks in commands 81  
broker  
  deleting 40  
  MQSeries Integrator Version 2.0,  
  migration 156  
  starting 26  
  trace 65  
broker domain  
  administration 4  
  enhancing and updating 37  
  management 4, 35  
  performance, managing 49  
  securing MQSeries resources 62  
  User Name Server, adding 38  
  workload, managing 49

## C

case sensitivity  
  broker names and fixed names 82  
  primary keywords and  
  parameters 81  
changes for this edition  
  SC34-5792-04 xiii  
  SC34-5792-02 xiii  
  SC34-5792-03 xiii  
channels 23  
character set for commands 82  
  Unicode characters 82  
clients, application 29  
CMRMUUIID, Configuration Manager,  
  migration 158

code page support 5  
  UNIX syslog 67  
collective 38  
Command Assistant 87  
commands  
  character set 82  
  list of 89  
  naming rules 82  
  platform availability 89  
  responses 83  
  rules for using 81  
  syntax help 83  
  use of 81  
  using MQSeries Integrator Command  
  Assistant 87  
complementary offerings  
  IBM Business Partners 212  
  Partner Offerings 212  
components  
  deleting 40  
  listing 36  
  modifying 36  
  starting and stopping 35  
  viewing and modifying 35  
configuration 9  
  application client tasks 29  
  authorization tasks 9  
  connecting components 22  
  connection tasks 21  
  Control Center user IDs 11  
  database 11  
  defining MQSeries Integrator  
  components 20  
  defining MQSeries resources 21  
  definition tasks 9  
  general guidance 32  
  initialization tasks 24  
  serviceuserID  
  UNIX platforms 11  
  Windows NT 10  
  user data tasks 29  
  user IDs 9  
configuration changes event  
  messages 184  
Configuration Manager  
  Control Center connection 21  
  MQSeries Integrator Version 2.0,  
  migration 157  
  starting 25  
  starting a listener 22  
  trace 65  
connection  
  DB2 database 18  
  SQL Server database 18  
connections  
  JDBC 18  
  ODBC 18  
contacting IBM 76  
Control Center  
  connecting to Configuration  
  Manager 21

Control Center (*continued*)  
  creating collectives 38  
  defining ACLs 39  
  defining topology 28  
  deploying configuration 28  
  MQSeries Integrator Version 2.0,  
  migration 156  
  MQSeries Integrator Version 2.0.1,  
  migration 155  
  MQSeries Publish/Subscribe  
  migration 171  
  populating collectives 38  
  starting 27  
  superuser 55  
  user IDs, authorizing 11  
creating and populating collectives 38

## D

database  
  access 36  
  access from message flows 30  
  authority 19  
  authorization 11  
  authorizing access 19  
  broker 16  
  code page support 36  
  configuration repository 16  
  deadlock 31  
  defining internal connections 18  
  definition 11  
  logs 75  
  management 36  
  message flow transactionality 31  
  message repository 16  
  Microsoft SQLServer 11  
  NEON data 20  
  Oracle 11  
  security 63  
  setup and configuration 12  
  UNIX platforms 13  
  Windows NT 12  
  SQL Server 18  
  Sybase 11  
DB2 6, 11, 16  
  as transaction manager 39  
  defining and authorizing 11  
  heap size 17  
  logs 75  
  publications 212  
  services 64  
  UNIX processes 64  
  Windows NT services 64  
deadlock, database 31  
debug trace 67  
defining ACLs 39  
definition tasks, networks 9  
delete broker 40  
deleting components 40  
deleting MQSeries Integrator  
  components 40

## E

- enhancing broker domains 37
- environment variables
  - library search path 15
  - NEON 197
  - trace 74
  - UNIX codepage 67
- error messages 74
- event reporting 183
  - ACL updates 186
  - configuration changes 184
  - examples 185
  - expired messages 191
  - general architecture 184
  - neighbor changes messages 185
  - notification message schema 192
  - operational information 188
  - operational warnings 191
  - subscription registration and deregistration 189
  - subscriptions and topics 189
- expired publications and subscriptions
  - event messages 191
- exporting message sets 41

## F

- formatted trace
  - debug example 72
  - normal example 70

## H

- heap size
  - DB2 17
  - JVM 94

## I

- IBM Business Partners 212
- IBM Support Center 76
- IBMMQSI2 superuser ID 55
- importing message sets 41
- information on the Internet
  - complementary offerings 212
  - MQSeries family libraries 212
  - MQSeries products 212
  - MQSeries SupportPacs 212

## L

- LD\_LIBRARY\_PATH 15
- local error log 65
  - UNIX syslog 66
  - Windows NT event log 65
- losing subscriptions 45

## M

- message
  - persistence 43
  - persistence for SCADA 43
  - under syncpoint control 43

- message flow
    - database access from 30
    - transactionality and database design 31
    - tuning for performance 50
  - message flow debugger 74
  - message set, importing and exporting 41
  - MHI1 SupportPac for problem determination 65
  - migration 155
    - adding new NEON rules and formats 158
  - MQSeries Integrator Version 1 158
    - access to NEON rules and formats 197
    - NEON logs 159
    - NEON Rules Daemon 159
    - NEON user exits 159
  - MQSeries Integrator Version 2.0
    - broker 156
    - CMRMUID column name 158
    - Configuration Manager 157
    - Control Center 156
  - MQSeries Integrator Version 2.0.1 155
    - Control Center 155
  - NEONRules and NEONFormatter Support 158
  - reloading NEON rules and formats 159
- MQSeries
- defining channels 23
  - defining resources 21
  - events 75
  - logs 75
  - publications 211
  - securing resources 62
  - stopping queue managers 35
  - transaction manager 39
  - transmission queue 23
  - trusted applications 49
- MQSeries Everyplace publications 209
- MQSeries Integrator
- starting the broker 26
  - starting the User Name Server 26
- MQSeries Integrator Command Assistant 4, 87
- commands supported 88
  - invocation 87
  - navigation 88
- MQSeries Integrator on the Internet 212
- MQSeries Integrator publications 209
- national language 210
  - platform-specific 209
- MQSeries Integrator Version 1
- migration 158
  - using NEON nodes 195
- MQSeries Integrator Version 2.0
- migration 156
- MQSeries Integrator Version 2.0.1
- migration 155
- MQSeries Integrator Version 2.0.2
- administration 3
  - changes for this edition, list xiii
  - connecting components 22

## MQSeries Integrator Version 2.0.2 (continued)

- connecting MQSeries Integrator components 23
  - defining components 20
    - broker 20
    - Configuration Manager 20
    - User Name Server 21
  - listing components 36
  - modifying components 35, 36
  - MQSeries Integrator Command Assistant 87
  - queue manager as Windows NT service 24
  - securing resources 51
  - starting 35
  - starting the Configuration Manager 25
  - starting the Control Center 27
  - stopping 35
  - stopping queue managers 35
  - viewing components 35
- MQSeries Publish/Subscribe migration 161
- MQSeries requirements 163
  - publications 211
- MQSeries Publish/Subscribe migration 4
- broker, network 173
  - broker, single 171
    - deployment 173
    - implementation 172
    - preparation 171, 172
  - brokers, transfer of data 170
  - commands and options 161
  - Control Center 171
  - deleting brokers 169
  - heterogeneous networks 163
  - independent networks 163
  - key to figures 161
  - migrating brokers 170
  - mixed network
    - adding a leaf node 165
    - adding a parent node 167
    - MQSeries channels 164
    - MQSeries queues 164
  - preparation 161
  - stream queues 163
- MQSeries Workflow publications 211
- MQSI\_LOCAL\_CCSID 67
- MQSI\_UTILITY\_TRACE 74
- MQSI\_UTILITY\_TRACESIZE 74
- mqsichangebroker 90
- modifying components 36
- mqsichangeconfigmgr 93
- modifying components 36
- mqsichangetrace 95
- changing user trace 69
  - starting user trace 68
  - stopping user trace 73
  - use in optional trace 67
- mqsichangeusernameserver 99
- modifying components 36
- mqsiclearmqpubsub 101
- mqsicopymsgset 103
- mqsicreatebroker 105
- managing database access 37

- mqscreatebroker 105 (*continued*)
  - MQSeries security 62
  - MQSeries trusted applications 49
  - recovery and restart 48
- mqscreateconfigmgr 111
  - managing database access 37
  - MQSeries security 62
  - recovery and restart 48
  - security domains 52
  - superuser ID 56
- mqscreateusernameserver 117
  - MQSeries security 63
  - recovery and restart 49
  - security domains 52
  - User Name Server, adding 38
- mqsdeletebroker 120
  - deleting a broker 40
  - recovery and restart 48
- mqsdeleteconfigmgr 122
  - recovery and restart 48
- mqsdeleteusernameserver 125
  - recovery and restart 49
  - removing topic-based security 41
- mqsformatlog 127
  - formatting user trace 70
- mqsimpexpmsgset 129
  - message set, importing and exporting 41
- mqsjoinmqpubsub 131
- mqsilcc 133
- mqsilist 135
  - listing components 36
- mqsilistmqpubsub 137
- mqsigratebroker 156
- mqsigrateconfigmgr 157
- mqsirmcopymsgset (deprecated) 103
- mqsirmimpexp (deprecated) 129, 140
- mqsinrfreload 140
- mqsireadlog 142
  - reading user trace 70
  - retrieving user trace 69
- mqsireporttrace 146
  - checking user trace 69
- mqsistart 148
- mqsistop 150
  - stopping queue managers 35

## N

- National Language Support 5
  - changing locale on UNIX platforms 6
  - changing locale on Windows NT 6
  - locales 6
  - UNIX syslog 7, 67
- NEON databases
  - database configuration file 197
  - installing database schema 195
  - setting system environment variables 197
- NEON\_ROOT 197
- NEON Rules Daemon 159
- neonreg.dat
  - NEON database configuration file 197

- NEONRules and NEONFormatter Support
  - code page support restrictions 5
  - migration 158
  - NNFie and NNRie 158, 198
- NEONRules and NEONFormatter Support publications 209
- network, managing MQSeries Integrator 35
- network configuration 9
- NN\_CONFIG\_FILE\_PATH 197, 198
- NNFie and NNRie 158, 198
- nodes, new in Version 2.0.1 156, 157
- normal trace 67
- notification message schema 192
- notification messages 183

## O

- ODBC tracing 75
- operational warnings 191
- Oracle database, defining and authorizing 11

## P

- parameters
  - a *ServicePassword*
    - mqschangebroker 90
    - mqschangeconfigmgr 93
    - mqschangeusernameserver 99
    - mqscreatebroker 107
    - mqscreateconfigmgr 113
    - mqscreateusernameserver 118
  - b (agent subcomponent)
    - mqschangetrace 98
    - mqsireporttrace 147
  - b *brokername*
    - mqsinrfreload 140
  - b *qualifier*
    - mqsireadlog 143
  - c *size*
    - mqschangetrace 97
  - d *SecurityDomainName*
    - mqschangeconfigmgr 94
    - mqschangeusernameserver 100
    - mqscreateconfigmgr 114
    - mqscreateusernameserver 118
  - e (export message set)
    - mqsimpexpmsgset 129
  - e *egroup*
    - mqschangetrace 96
    - mqsilist 135
    - mqsireadlog 143
    - mqsireporttrace 146
  - e *MRMDataSourceUserID*
    - mqscreateconfigmgr 113
  - f *mflow*
    - mqschangetrace 96
    - mqsireporttrace 146
  - f (read from file system)
    - mqsireadlog 143
  - i (immediate stop)
    - mqsistop 150
  - i (import message set)
    - mqsimpexpmsgset 129

- parameters (*continued*)
  - i *inputfilename*
    - mqsformatlog 127
  - i *ServiceUserID*
    - mqschangebroker 90
    - mqschangeconfigmgr 93
    - mqschangeusernameserver 99
    - mqscreatebroker 106
    - mqscreateconfigmgr 112
    - mqscreateusernameserver 117
  - j *MaxJVMHeapSize*
    - mqschangeconfigmgr 94
  - k *TargetMessageSetLevel*
    - mqsicopymsgset 104
  - l *level*
    - mqschangetrace 97
  - l *SourceMessageSetLevel*
    - mqsicopymsgset 104
    - mqsimpexpmsgset 129
  - m (migrate existing broker)
    - mqscreatebroker 108
  - m *mode*
    - mqschangetrace 97
  - m *MRMDataSourceName*
    - mqscreateconfigmgr 113
  - n -m (delete repositories)
    - mqsdeleteconfigmgr 122
  - n *DataBaseName*
    - mqscreateconfigmgr 113
  - n *DataSourceName*
    - mqscreatebroker 107
  - n *MRMDataSourceName*
    - mqsicopymsgset 103
    - mqsimpexpmsgset 130
  - n *NeighborQueueManagerName*
    - mqsiclearmqpubsub 101
  - n (stop running as trusted application)
    - mqschangebroker 91
  - o *outputfilename*
    - mqsformatlog 127
    - mqsireadlog 143
  - p *DataBasePassword*
    - mqschangeconfigmgr 94
  - p *DataSourcePassword*
    - mqschangebroker 91
    - mqscreatebroker 108
    - mqscreateconfigmgr 113
  - p *MRMDataSourcePassword*
    - mqsicopymsgset 104
    - mqsimpexpmsgset 130
  - p *ParentQueueManagerName*
    - mqsjoinmqpubsub 131
  - q (delete queue manager)
    - mqsdeletebroker 121
    - mqsdeleteconfigmgr 122
    - mqsdeleteusernameserver 125
  - q *QueueManagerName*
    - mqscreatebroker 107
    - mqscreateconfigmgr 113
    - mqscreateusernameserver 118
  - q (stop MQSeries queue manager)
    - mqsistop 150
  - r *MRMDataSourcePassword*
    - mqschangeconfigmgr 94
    - mqscreateconfigmgr 113

parameters (*continued*)

- r *RefreshInterval*
  - mqsichangeusernameserver 100
  - mqsicreateusernameserver 118
- r (reset trace log)
  - mqsichangetrace 96
- s *SourceMessageSetName*
  - mqsicopymsgset 104
  - mqsiiimpexpmsgset 129
- s *UserNameServerQueueManagerName*
  - mqsichangebroker 91
  - mqsichangeconfigmgr 94
  - mqsicreatebroker 108
  - mqsicreateconfigmgr 114
- t *TargetMessageSetName*
  - mqsicopymsgset 104
- t (read from service trace log)
  - mqsireadlog 143
  - mqsireporttrace 147
- t (run as trusted application)
  - mqsichangebroker 91
  - mqsicreatebroker 108
- t (service trace options)
  - mqsichangetrace 98
- u *DataBaseUserID*
  - mqsicreateconfigmgr 113
- u *DataSourceUserID*
  - mqsicreatebroker 107
- u *MRMDDataSourceUserID*
  - mqsicopymsgset 104
  - mqsiiimpexpmsgset 130
- u (read from user trace log)
  - mqsireadlog 143
  - mqsireporttrace 146
- u (user trace options)
  - mqsichangetrace 96
- w (delete workpath files)
  - mqsdeletebroker 121
  - mqsdeleteconfigmgr 122
  - mqsdeleteusernameserver 125
- w *WorkPath*
  - mqsicreatebroker 108
  - mqsicreateconfigmgr 114
  - mqsicreateusernameserver 118
- 0, 1 or 2
  - mqsilcc 133
- brokername
  - mqsichangebroker 90
  - mqsiclearmqpubsub 101
  - mqsicreatebroker 106
  - mqsdeletebroker 121
  - mqsijoinmqpubsub 131
  - mqsilist 135
  - mqsilistmqpubsub 137
- component
  - mqsichangetrace 96
  - mqsireadlog 142
  - mqsireporttrace 146
  - mqsistart 148
  - mqsistop 150
- FileName
  - mqsiiimpexpmsgset 130
- Partner Offerings 212
- PDF (Portable Document Format) 210
- performance 49
  - SCADA nodes 50
  - tuning message flows 50

- Portable Document Format (PDF) 210
- position of parameters 81
- previous versions of MQSeries
  - Integrator 155
- primary security domain, Windows
  - NT 53
- principals
  - global groups 53
  - IBMMQSI2 55
  - MQSeries Integrator groups 52
- problem determination 65
  - MHI1 SupportPac 65
- publications
  - DB2 212
  - MQSeries 211
  - MQSeries Everyplace 209
  - MQSeries Integrator 209
  - MQSeries Publish/Subscribe 211
  - MQSeries Workflow 211
- publish/subscribe
  - collective 38
  - removing topic-based security 41
  - setting up a network 37
  - unused subscriptions 45

## Q

- queue manager
  - log 75
  - stopping 35
  - Windows NT service, starting as 24

## R

- recovery and restart 42
  - application messages 43
  - backout processing 44
  - backup 47
  - broker internal messages 43
  - messages 42
  - recovery scenarios 48
    - broker 48
    - Configuration Manager 48
    - User Name Server 49
  - restart scenarios 45
    - broker 45
    - Configuration Manager 46
    - User Name Server 46
  - subscriptions 45
  - unused subscriptions 45
- responses to commands 83
- restarting components 42, 45
- rules for using commands 81
  - blanks in commands 81
  - case sensitivity
    - broker names and fixed names 82
    - primary keywords and parameters 81
    - naming of identifiers 82

## S

- SCADA
  - backout processing 45
  - message persistence 43
  - performance 50

- security 51
  - brokers and other components 51
  - Control Center 51
  - database resources 63
  - MQSeries Integrator resources 51
  - MQSeries resources 62
  - principals 52
  - queues and queue managers 51
  - scenarios 56
    - in a local domain 60
    - in a primary domain 56
    - in a trusted domain 58
  - topic-based 51
  - UNIX security domain 53
  - UNIX security domains 61
  - Windows NT
    - primary security domain 53
    - trusted security domain 53
    - Windows NT security domain 52
- service trace 67, 73
- service user ID, authorizing
  - UNIX platforms 11
  - Windows NT 10
- setting up security 51
- softcopy books 210
- SQLServer database, defining and authorizing 11
- starting user trace 68
- state changes event messages 188
- stopping user trace 73
- subscription registration and deregistration event messages 189
- subscriptions
  - losing 45
  - unused, managing 45
- subscriptions and topics event messages 189
- superuser ID, IBMMQSI2 55
- Support Center, IBM 76
- SupportPac 212
- Sybase database, defining and authorizing 11
- syntax diagrams, how to read 83
- syntax help, commands 83
- system administration overview 4
- system management 5, 183

## T

- terms used in this book xi
- topic-based security 37
- trace 65
  - changing options 69
  - current options 69
  - debug 67
  - formatting 70
  - levels 68
  - log files 68
  - mqsichangetrace 95
  - mqsiformatlog 127
  - mqsilcc 133
  - mqsireadlog 142
  - normal 67
  - ODBC 75
  - optional 67
  - reading 70
  - retrieving logs 69



- trace 65 (*continued*)
  - service 67, 73
  - starting trace 68
  - stopping 73
  - Trace node 67
  - unformatted, viewing 70
  - UNIX codepage 67
  - UNIX syslog 66
  - user 67
  - viewing 73
  - Windows NT event log 65
- transaction management, using MQSeries 39
- trusted applications 49
- trusted security domain, Windows NT 53

## U

- unformatted trace, XML viewer
  - example 70
- Unicode characters 82
- UNIX database setup and configuration 13
- UNIX security domains 61
- UNIX syslog 66
  - National Language Support 7, 67
- updating broker domains 37
- upgrading MQSeries Integrator 155
- user IDs
  - authorization 9
  - database authority 19
  - definition 9
- User Name Server 75
  - adding to broker domain 38
  - removing topic-based security 41
  - starting 26
  - starting user trace 68
  - trace 65
- user trace 67

## V

- viewing user trace 73

## W

- Windows 2000 xi
- Windows NT xi
  - database setup and configuration 12
  - event log 65
  - security domains 53
  - service, starting queue managers
    - as 24
- workload 49



---

## Sending your comments to IBM

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 make comments about the functions of IBM products or systems, 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, to this address:

User Technologies Department (MP095)  
IBM United Kingdom Laboratories  
Hursley Park  
WINCHESTER,  
Hampshire  
SO21 2JN  
United Kingdom

- By fax:
  - From outside the U.K., after your international access code use 44-1962-842327
  - From within the U.K., use 01962-842327
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.







Printed in U.S.A.

SC34-5792-04

