

MQSeries[®] Integrator



Introduction and Planning

Version 2.0.2

MQSeries® Integrator



Introduction and Planning

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 171.

Third 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 xii

What you need to know to understand this book. xii

Terms used in this book xii

Summary of changes xiii

Changes for this edition (GC34-5599-02) xiii

Changes for the second edition (GC34-5599-01) xiii

Part 1. Introduction 1

Chapter 1. MQSeries and business integration 3

The MQSeries family 3

MQSeries 3

MQSeries Integrator 3

MQSeries Workflow 4

Using MQSeries for business integration 4

Using MQSeries Integrator in your business 5

MQSeries Integrator Version 2 and previous IBM offerings. 5

Getting started with MQSeries Integrator 6

Chapter 2. MQSeries Integrator overview and concepts 9

The Configuration Manager 9

Brokers 11

Connecting brokers for publish/subscribe 12

System management interfaces 13

Business processing rules (message flows) 13

Creating message flows 14

Message flow input and output. 15

Publish/subscribe services 15

Associating message flows with brokers. 16

Simple message flow examples. 16

Messages and message sets 17

Messages predefined in the Control Center. 18

Messages predefined by the NEONFormatter 18

Self-defining messages. 19

Parsing messages 19

Associating message sets with brokers 19

The Control Center 20

Updates, assignment, and deployment 20

Exporting and importing resource definitions 21

Help and online Tour 22

Applications and clients 22

Point-to-point applications 23

Publish/subscribe applications 23

Client connections to brokers and message flows 24

Special considerations for MQSeries Everyplace and SCADA 25

The User Name Server 27

Access Control Lists 28

Dependencies. 29

MQSeries dependencies 29

Database dependencies 30

Release to release migration 31

Chapter 3. MQSeries Integrator: a business scenario 33

The retail scenario 33

Business data. 35

Business needs 36

Business solution using MQSeries Integrator

Version 2 36

Implementing the business solution 38

Part 2. Business process planning 39

Chapter 4. Message flows 41

What is a message flow? 41

What does a message flow consist of? 41

Message flows and units of work 42

Parallel processing of message flow instances 42

Interaction of message flows. 43

Transformation 43

Intelligent routing 44

Enriching message content 45

What is a message processing node? 45

Execution groups 48

Message flows and message sets 48

Message flows for publish/subscribe services 49

Supplied message flows and nodes 49

Primitive node types 50

Supplied message flows 53

Adding or enhancing message processing nodes 54

Solving message flow problems with the Debugger 54

Chapter 5. Messages. 55

Predefined and self-defining messages 55

Predefined messages 55

Self-defining messages. 57

How messages are processed in a message flow 58

Exporting and importing MRM message sets 59

Message parsers 59

Default message parsers 59

Using message templates and messages 61

Client access to messages. 61

Creating additional parsers 62

Part 3. Application planning 63

Chapter 6. Application design 65

Communication models	65
Point-to-point communications	66
Publish/subscribe communications	66
Application programming	66
Message headers	67
Reusing existing applications	68
Send and forget	68
Request/reply	68
Publish/subscribe	69
Writing new applications	69
MQSeries queues	70
Message order	70
Publish/subscribe	71
Transaction support	71
Message persistence	72
Security	73
Summary	73

Chapter 7. Designing publish/subscribe applications 75

How publish/subscribe applications interact with a broker	75
Publications	76
Retained publications	76
Local and global publications	78
Conference-type applications	78
Subscriptions	78
Subscription points	79
Filters	80
Local subscriptions	81
Retained publications	81
Message persistence	82
Topics	82
Special characters in topics	83
Topic semantics and usage	84
Using wildcards with topics	85
Multiple topics	85
Broker networks	85
Collectives	86
Topic-based security	86
Principals and the User Name Server	86
Access control lists	87
Checking publications and subscriptions	91
Summary	92

Part 4. Systems planning. 93

Chapter 8. System requirements. 95

Summary of system requirements	95
System requirements for AIX components	96
Hardware requirements	96
Disk space required	97
Software requirements	97
System requirements for HP-UX components	98
Hardware requirements	98
Disk space required	98
Software requirements	98
System requirements for Sun Solaris components	100
Hardware requirements	100

Disk space required	100
Software requirements	100
System requirements for Windows NT components	102
Hardware requirements	102
Disk space required	102
Software requirements	102
Database support	105
Client requirements	105
License information	105
National language support	105

Chapter 9. Planning your MQSeries Integrator network 107

Planning MQSeries Integrator resources	107
Naming conventions	107
Broker domain basics	109
Client applications	113
Designing the MQSeries infrastructure	114
MQSeries resources for brokers	114
MQSeries resources for the Configuration Manager	115
MQSeries resources for the User Name Server	116
MQSeries resources for the Control Center	116
MQSeries resources for client applications	116
MQSeries clusters	117
Planning database resources	118
Database requirements	118
Databases and code pages	119
Database locations	120
Database backup and recovery	120
Planning security	120
Security and principals	121
Operational security	127
Control Center security	128
Application security	129
Planning for data conversion	131

Chapter 10. Managing your MQSeries Integrator network 133

Managing broker domain components	133
Managing application and business processes	134
Monitoring and analysis	134
Problem determination	134
Managing workload and performance	137
System management	138

Chapter 11. Enhancing your broker domain 139

General guidance for writing plug-ins	139
Writing your own message processing node types	139
Writing your own parsers	140

Part 5. Appendixes 141

Appendix A. Planning for migration and integration. 143

MQSeries Integrator Version 1.	143
Installation	143
Run-time	144

MQSeries Publish/Subscribe	147
Scenarios for migration and integration.	147
Product differences	148
Scenario 1: running two independent broker networks	160
Scenario 2: creating and operating a heterogeneous network	161
Scenario 3: migrating MQSeries Publish/Subscribe brokers	162

Appendix B. The product packages 165

The MQSeries Integrator for AIX package	165
The MQSeries Integrator for HP-UX package	166
The MQSeries Integrator for Sun Solaris package	167
The MQSeries Integrator for Windows NT package	168

Appendix C. Notices 171

Trademarks	173
----------------------	-----

Glossary of terms and abbreviations 175

Bibliography. 181

MQSeries Integrator Version 2.0.2 cross-platform publications	181
MQSeries Integrator Version 2.0.2 platform-specific publications	181
MQSeries Everyplace publications	181
NEONRules and NEONFormatter Support for MQSeries Integrator publications	181
Softcopy books	182
Portable Document Format (PDF)	182
MQSeries library references	183
MQSeries Publish/Subscribe publications	183
MQSeries Workflow publications	183
DB2 publications	184
MQSeries information available on the Internet	184

Index 185

Sending your comments to IBM 189

Figures

1. The Configuration Manager	10	12. Branches and back-end systems	35
2. The broker	12	13. SRU receipt	35
3. A collective	13	14. The business flow (simplified)	38
4. Message flow components	15	15. Publish/subscribe with a single broker	76
5. A simple message flow: case 1	16	16. Example topic tree	83
6. A simple message flow: case 2	16	17. Publish/subscribe in a network	86
7. A simple message flow: case 3	17	18. Inheriting ACLs in a topic tree	90
8. A simple message flow: case 4	17	19. Collectives with a broker domain	111
9. Applications connecting to a broker	24	20. A heterogeneous network	153
10. The User Name Server	28	21. Stream authorities	155
11. SRU headquarters and branch hierarchy	34		

Tables

1. Recipient information	36	7. Summary of authorizations in the Windows NT environment	125
2. ACL permissions.	89	8. MQRFH and MQRFH2 mapping	149
3. The ACLs for inheritance	90	9. Summary of message option support	150
4. Summary of installation options	95	10. MQRFH and MQRFH2 client application options	152
5. Supported databases for brokers and user data.	105	11. Migration inhibitors checklist	164
6. Summary of authorization in the UNIX environments	125	12. File names of MQSeries Integrator book PDFs	182

About this book

This book provides an overview of IBM MQSeries Integrator Version 2.0.2. It introduces the concepts of the product, and provides the information to help you plan for an MQSeries Integrator network.

“Part 1. Introduction” on page 1 gives you a broad understanding of the MQSeries family of products, and an introduction to MQSeries Integrator. It also discusses additional, related offerings from IBM. It provides background information that can benefit everyone working with MQSeries Integrator.

“Part 2. Business process planning” on page 39 builds on the introduction in Part 1, providing information that helps your business planners develop message structure and processing requirements that will support a successful MQSeries Integrator environment.

You can find implementation details for the tasks covered in this part in *MQSeries Integrator Using the Control Center* and *MQSeries Integrator ESQL Reference*.

“Part 3. Application planning” on page 63 explores the application aspects of your environment, further clarifying the introduction in Part 1 and guiding you through the considerations for application planning and development.

You can find implementation details for the tasks covered in this part in the *MQSeries Integrator Programming Guide*.

“Part 4. Systems planning” on page 93 provides details of the infrastructure you will need, and how you can configure it, to complement your applications and achieve your business goals. It provides system administrators with hardware and software requirements, and the infrastructure required to support your environment. It also tells you how you can enhance your broker domain by writing plug-ins for new parsers and message processing nodes.

You can find implementation details for the tasks covered in this part in the *MQSeries Integrator Administration Guide*.

Appendixes provide information on migration, and on the contents of the MQSeries Integrator product package.

A glossary and a bibliography are provided at the back of the book.

For details of installing MQSeries Integrator on the supported platforms, see:

- *MQSeries Integrator for AIX Installation Guide*
- *MQSeries Integrator for HP-UX Installation Guide*
- *MQSeries Integrator for Sun Solaris Installation Guide*
- *MQSeries Integrator for Windows NT Installation Guide*

Who this book is for

This book is for business administrators who need an understanding of MQSeries Integrator to enable them to make a purchasing decision. When the product has been purchased, this book provides information to business and system administrators on how to make the best use of the product within their environment.

What you need to know to understand this book

To understand this book, you should have some familiarity with the concepts of application integration, and a thorough understanding of your existing and planned business tasks and objectives.

An understanding of MQSeries concepts is also useful.

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 175. These terms are shown like *this* at their first use.

The book uses the following shortened names:

- 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 “MQSeries information available on the Internet” on page 184).
- CICS[®]: a general term for IBM CICS products including CICS, TXSeries[™], and WebSphere[™].
- DB2[®]: a general term to encompass IBM DB2 Universal Database[®] Enterprise Edition, Connect Enterprise Edition and Extended Enterprise Edition.
- UNIX[®]: the UNIX-based platforms, such as AIX[®], HP-UX, and Sun Solaris.

Summary of changes

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

Changes for this edition (GC34-5599-02)

Major changes for this edition include:

- Addition of the new product MQSeries Integrator for HP-UX Version 2.0.2
- Additional information to cover the following product changes:
 - Improved national language support
 - Debugger function added to the Control Center
 - Windows 2000 added as an alternative platform for MQSeries Integrator for Windows NT
 - Oracle XA support available on MQSeries Integrator for Sun Solaris
 - New NEON nodes and a new parser added to provide access to NEONRules and NEONFormatter Support
 - New MQSeries Everyplace and SCADA nodes added
 - Windows NT Server Version 4.0, running on an Integrated IBM @server xSeries™ installed in an IBM @server iSeries™ 400® (AS/400®), is listed as a suitable platform for the MQSeries Integrator for Windows NT components that run under Windows NT
- Minor technical and editorial improvements throughout the book

The usability of the Control Center is also improved in this version of MQSeries Integrator. See *MQSeries Integrator Using the Control Center* for full details.

Changes for the second edition (GC34-5599-01)

Major changes for this edition include:

- Additional information to cover the following product changes:
 - New products MQSeries Integrator for AIX Version 2.0.1 and MQSeries Integrator for Sun Solaris Version 2.0.1.
 - New IBM primitive nodes (FlowOrder, Label, and RouteToLabel)
- Minor technical and editorial improvements throughout the book

Changes

Part 1. Introduction

This part provides introductory level information that will benefit everyone working with MQSeries Integrator. It includes the following chapters:

- “Chapter 1. MQSeries and business integration” on page 3 introduces the products in the MQSeries family, and the way in which they support business integration.
- “Chapter 2. MQSeries Integrator overview and concepts” on page 9 discusses the function of MQSeries Integrator, giving an outline of the support it provides. It also gives references to more detailed information in the remainder of this book, and in the other books in the MQSeries Integrator library.
- “Chapter 3. MQSeries Integrator: a business scenario” on page 33 explores a business scenario that illustrates the value that MQSeries Integrator adds to your IT environment.

Chapter 1. MQSeries and business integration

The last few years have seen a growing interest and investment in messaging middleware. IBM's MQSeries is an industry leader in this area, and provides a messaging infrastructure to many diverse businesses and applications.

IBM has developed a family of products, based around the messaging transport layer, that provides not only the fundamental requirements of secure, reliable information exchange, but also incorporates services and business process support to help you to make best use of your investment in systems and applications. The richness and flexibility of this support enables you to respond to new opportunities that arise when your business grows and diversifies.

The MQSeries family

The MQSeries family consists of three complementary offerings:

- "MQSeries"
- "MQSeries Integrator"
- "MQSeries Workflow" on page 4

MQSeries

MQSeries provides assured, once-only delivery of *messages* between your IT systems. It connects more than thirty industry platforms including those from IBM, Microsoft[®], and Sun, using a variety of communications protocols.

MQSeries provides rich support for applications:

- Application programming interfaces: the *Message Queue Interface (MQI)* and *Application Messaging Interface (AMI)* are supported in several programming languages.
- Communication models: *point-to-point* (including *request/reply* and *client/server*) and *publish/subscribe* are supported.
- The complexities of communications programming are handled by the messaging services and are therefore removed from the application logic.
- Applications can access other systems and interfaces through gateways and adapters to products such as Lotus[®] Domino[™], Microsoft Exchange/Outlook, SAP/R3, and IBM's CICS and IMS/ESA[®] products.

MQSeries Integrator

MQSeries Integrator works with MQSeries messaging, extending its basic connectivity and transport capabilities to provide a powerful *message broker* solution driven by business *rules*. Messages are formed, routed, and *transformed* according to the rules defined by an easy-to-use *graphical user interface (GUI)*.

Diverse applications can exchange information in unlike forms, with brokers handling the processing required for the information to arrive in the right place in the correct *format*, according to the rules you have defined. The applications have no need to know anything other than their own conventions and requirements.

Applications also have much greater flexibility in selecting which messages they wish to receive, because they can specify a *topic filter*, or a *content-based filter*, or both, to control the messages made available to them.

MQSeries family

MQSeries Integrator provides a framework that supports supplied, basic, functions along with *plug-in* enhancements, to enable rapid construction and modification of business processing rules that are applied to messages in the system.

MQSeries Workflow

MQSeries Workflow works with MQSeries messaging to add a further dimension to your business integration by aligning and integrating an organization's staff resources, processes, and capabilities with business strategies. It enables organizations to accelerate process flow, optimize costs, eliminate errors and improve workgroup productivity.

MQSeries Workflow is designed to enable integration of all participants in the business process, including those external to your organization. It ensures the right information gets to the right person at the right time.

MQSeries Workflow can be used in combination with MQSeries Integrator, providing a high level of flexibility to allow business and message processing to be as simple or as complicated as your business demands.

Using MQSeries for business integration

MQSeries is the focal point of the IBM Business Integration strategy, which addresses integration of applications, data, and processes from both business and IT perspectives.

Business integration is the coordination and cooperation of all your business processes and applications. It involves bringing together the data and process intelligence in your enterprise, and harnessing these to enable all your applications and your users to achieve their business needs.

Business integration means that:

- You can connect customers, suppliers, partners, and service providers, with continuing security and control, to enable newly built and re-engineered applications for more effective business processes (for example, Supply Chain Management).
- You can make mergers and acquisitions a success by integrating dissimilar IT infrastructures from two or more companies so they can work as a single entity.
- You can react more quickly to market trends and opportunities because your IT systems are flexible and dependable, and no longer constraining.
- The barriers of diverse computer systems, geographic boundaries, time differences, language and format differences, and different methods of working can all be overcome.

You can use the MQSeries family products to support your business integration needs:

- MQSeries messaging provides a secure and far-reaching communications infrastructure.
- MQSeries Integrator and MQSeries Workflow provide a range of services that allow you to apply intelligence to your business data as it travels through your network.

Using MQSeries Integrator in your business

MQSeries Integrator addresses the needs of business and application integration through management of information flow. It provides services based on message brokers to allow you to:

- Route a message to several destinations, using rules that act on the contents of one or more of the fields in the message or message header.
- Transform a message, so that applications using different formats can exchange messages in their own formats.
- Store and retrieve a message, or part of a message, in a database.
- Modify the contents of a message (for example, by adding data extracted from a database).
- Publish a message to make it available to other applications. Other applications can choose to receive publications that relate to specific topics, or that have specific content, or both.
- Create structured topic names, topic-based access control functions, content-based *subscriptions*, and *subscription points*.
- Exploit a plug-in interface to develop *message processing node* types that can be incorporated into the broker framework to complement or replace the supplied nodes, or to incorporate node types developed by Independent Software Vendors (ISVs).
- Enable instrumentation by products such as those developed by Tivoli[®], using system management hooks. *Tivoli support is not available on the HP-UX platform.*

The benefits of MQSeries Integrator can be realized both within and beyond your enterprise:

- Your processes and applications can be integrated by providing message and data transformations in a single place, the broker. This helps reduce costs of application upgrades and modifications.
- You can extend your systems to reach your suppliers and customers, by meeting their interface requirements within your brokers. This can help you improve the quality of your interactions and allow you to respond more quickly to changing or additional requirements.

For a practical illustration of the use of MQSeries Integrator in business, see “Chapter 3. MQSeries Integrator: a business scenario” on page 33.

MQSeries Integrator Version 2 and previous IBM offerings

The following offerings from IBM are enhanced and extended by MQSeries Integrator Version 2:

- MQSeries Integrator Version 1
- MQSeries Publish/Subscribe

MQSeries Integrator Version 2 extends the capabilities of MQSeries Integrator Version 1 and MQSeries Publish/Subscribe by supporting:

- Integration of the publish/subscribe and rules and transformation functions, enabling the output from the rules engine to be directed straight to the publish/subscribe service without use of an intermediate *queue*.
- Enhanced publish/subscribe function through exploitation of structured topic names, *access control*, content-based subscriptions, and subscription points.
- Enhancement of message processing through the addition of new message processing nodes to complement or replace the supplied nodes.

Using MQSeries Integrator

- Interfaces that allow messages to be enriched with information from a database, or to be stored in a database.
- XA technology for enhanced transactional integrity on MQSeries Integrator for Sun Solaris using Oracle as well as DB2 databases.
- Easier problem resolution using the Debugger facilities of the *Control Center*.
- Interaction with *MQSeries Everyplace* applications for lightweight and mobile devices, using the new MQeInput and MQeOutput nodes.
- Interaction with very lightweight publish and subscribe applications, used by specialist hardware and companies that use SCADA Device Protocol, by using the new SCADAInput, SCADAOutput, and enhanced Publication nodes.
- Access to the NEONRules and NEONFormatter Support component by using the new NEONTransform, NEONMap, and NEONRulesEvaluation nodes, and the parser for the new NEONMSG message domain.
- Availability of MQSeries Integrator on the HP-UX platform.

You can upgrade your applications, messages, and brokers to take advantage of the enhancements in Version 2. You can also continue to use your existing applications and messages unchanged, by tailoring your Version 2 system to provide compatible support.

MQSeries Integrator Version 2 brokers can interact with MQSeries Publish/Subscribe brokers in a common publish/subscribe environment, to provide coexistence within a single mixed broker network.

If you already have MQSeries Integrator Version 1, or MQSeries Publish/Subscribe, or both, see “Appendix A. Planning for migration and integration” on page 143 and the *MQSeries Integrator Administration Guide* for details of planning for and implementing your migration.

Getting started with MQSeries Integrator

The information in this book helps you to:

1. Assess how MQSeries Integrator meets your business needs, and make a purchasing decision.
 - “Chapter 2. MQSeries Integrator overview and concepts” on page 9 introduces the concepts and components of MQSeries Integrator, and explains their relationships.
 - “Chapter 3. MQSeries Integrator: a business scenario” on page 33 describes a scenario that explains how MQSeries Integrator helps you to solve business integration problems.
2. Plan for implementation and deployment of MQSeries Integrator.

- “Part 2. Business process planning” on page 39 discusses your business processes and entities. It describes *message flows*, *messages*, and *message sets*, and the rules that define how these messages are processed.

When you understand the concepts, and have completed the planning tasks to define your environment, refer to *MQSeries Integrator Using the Control Center* for details of how to implement these plans and carry out your business administration tasks.

- “Part 3. Application planning” on page 63 describes how you can integrate existing applications, and create new ones, to complete the processing of messages flowing through your network.

Detailed guidance for writing and deploying these applications is provided in the *MQSeries Integrator Programming Guide*.

- “Part 4. Systems planning” on page 93 summarizes the infrastructure requirements of your network, and discusses how you can configure the MQSeries Integrator components to provide the support your business processing requires.

You can find full details of the system requirements for MQSeries Integrator in the *MQSeries Integrator Installation Guide* for your operating system. These books also contain instructions for installing MQSeries Integrator on your chosen operating system, and guides you through some simple tasks that help you verify that installation.

You can find details for system administration tasks in the *MQSeries Integrator Administration Guide*.

- “Appendix A. Planning for migration and integration” on page 143 provides the information you require if you already use an earlier version of MQSeries Integrator, or have downloaded and deployed MQSeries Publish/Subscribe. It helps you plan for deployment of MQSeries Integrator Version 2.0.2 brokers in your current environment.

For details of how you can upgrade your current environment to MQSeries Integrator Version 2.0.2, refer to the *MQSeries Integrator Administration Guide*.

Getting started

Chapter 2. MQSeries Integrator overview and concepts

MQSeries Integrator Version 2.0.2 supports business processes that meet your application and business integration needs. This support is provided by a number of components and services that work together to manage the resources required by your applications and business processes.

This chapter looks at the MQSeries Integrator components, their relationships, and the services they provide. It concludes with a summary of MQSeries Integrator's dependencies on other software products and the levels required.

The following topics are introduced:

- “The Configuration Manager”
- “Brokers” on page 11
- “Business processing rules (message flows)” on page 13
- “Messages and message sets” on page 17
- “The Control Center” on page 20
- “Applications and clients” on page 22
- “The User Name Server” on page 27
- “Dependencies” on page 29
- “Release to release migration” on page 31

The Configuration Manager

The *Configuration Manager* is the central component of your MQSeries Integrator environment. The components and resources managed by the Configuration Manager constitute the *broker domain*. The Configuration Manager serves three main functions:

- It maintains configuration details in the *configuration repository*. This is a set of database tables that provide a central record of the broker domain components.
- It manages the initialization and deployment of brokers and message processing operations in response to actions initiated through the Control Center. It communicates with other components in the broker domain using MQSeries transport services.
- It checks the authority of defined user IDs to initiate those actions.

You must install, create, and start a single Configuration Manager to manage your broker domain. The Configuration Manager must be installed and configured in the Windows NT environment. It is not supported on any other operating system. Once started, the Configuration Manager runs in the background.

You can view, create, modify, and delete the contents of the configuration repository using the Control Center. The Control Center must also be installed on Windows NT. It is not supported on any other operating system. A fuller description of the Control Center is given in “The Control Center” on page 20.

The Configuration Manager provides a service to the other components in the broker domain, providing them with *configuration* updates in response to actions you take from the Control Center. The Configuration Manager validates that the user requesting each action from the Control Center is authorized to perform that action.

Configuration Manager

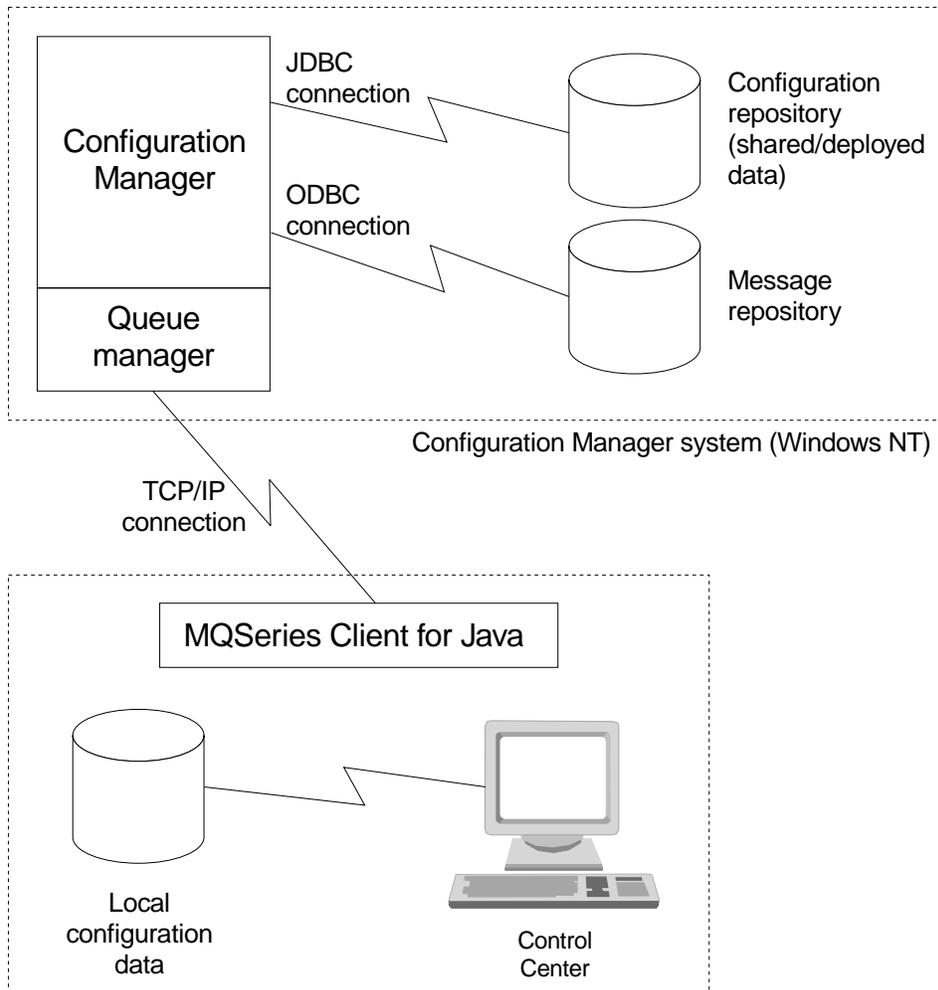


Figure 1. The Configuration Manager

When you create the Configuration Manager, the following resources are also created:

- A set of tables in a database, known as the configuration repository. This database must be created using IBM DB2 Universal Database for Windows NT. The Configuration Manager uses a *JDBC™ (Java™ Database Connectivity)* connection to this database.
- A set of tables in a database, known as the *message repository*. This database must be created using IBM DB2 Universal Database for Windows NT. The Configuration Manager uses an *ODBC (Open Database Connectivity)* connection to this database.
- A set of fixed-name queues, defined to the *queue manager* that hosts the Configuration Manager. You must identify this queue manager when you create the Configuration Manager, and it must exist on the same physical system as the Configuration Manager. It will be created when the Configuration Manager is created, if it does not already exist.
- A server connection, defined to the queue manager that hosts the Configuration Manager. This connection is used by all instances of the Control Center.

Brokers

The broker is a named resource that hosts and controls your business processes, which you define as *message flows*. Your applications communicate with the broker to take advantage of the services provided by the message flows. Applications send new messages to the message flow, and receive processed messages from the message flow, using MQSeries queues and connections.

You can install, create, and start any number of brokers within a broker domain. You can create more than one broker on any one physical system if you choose, but you must specify a unique queue manager for each broker. However, a single broker can share a queue manager with the Configuration Manager.

When you create a broker, the following resources are also created:

- A set of tables in a database to hold the broker's local data. This database can be created using a number of database products, depending on the operating system on which you install MQSeries Integrator:
 - IBM DB2 Universal Database
 - Microsoft SQL Server (Windows NT only)
 - Oracle
 - Sybase

The broker uses an ODBC connection to its database. These broker tables are also referred to as the broker's local persistent store. For more information about supported databases, see Table 5 on page 105.

- A set of fixed-name queues, defined to the queue manager that hosts this broker. You must identify this queue manager when you create the broker, and it must exist on the same physical system as the broker. It is created when the broker is created, if it does not already exist.

When you create a broker on the system on which you have installed the broker component, the information about the broker's configuration is not automatically recorded in the configuration repository (managed by the Configuration Manager). You must use the Control Center (the *Topology* view) to create a reference to this broker with the same name that you specified when you created that broker (see "The Control Center" on page 20 for more information about the Control Center).
Creating a reference:

- Stores the broker information in the configuration repository.
- Defines a default *execution group* on this broker. You can define further execution groups if you want. Each message flow providing a service on this broker must be deployed to an execution group before that service can be used by applications.

Brokers

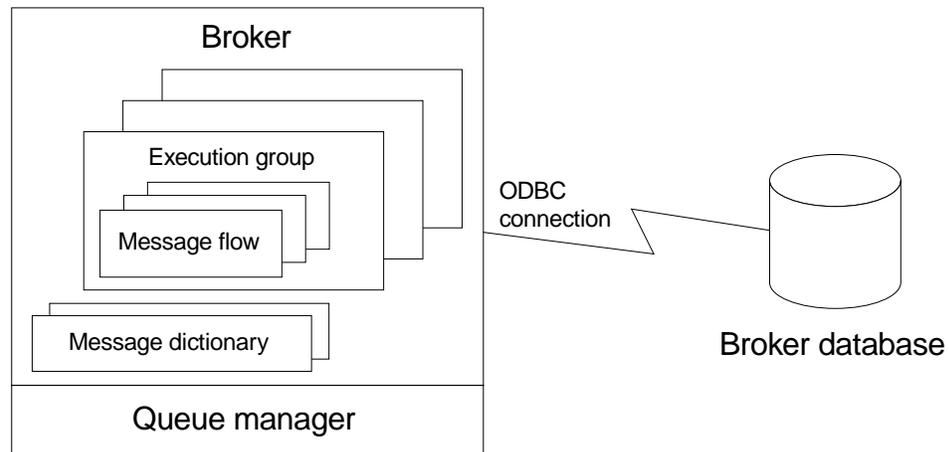


Figure 2. The broker

When you have created the broker reference, you must *deploy* the changes to your broker domain for them to take effect. The deploy action:

- Initiates communications between the Configuration Manager and the broker.
- Initializes the broker so that it is ready to execute message flows. The broker receives configuration information from the Configuration Manager, and stores it in its database.

When you have created the broker reference, you can assign message flows (see “Business processing rules (message flows)” on page 13) to the broker’s execution groups, and any message sets (see “Messages and message sets” on page 17) required by those message flows to the broker. These changes must also be deployed before they can be activated. You can deploy these resources individually, or together, but until all related resources (for example, a broker, a message flow and the message set it uses) are deployed, you cannot use the message flow on that broker.

Connecting brokers for publish/subscribe

If you plan to create message flows that provide a publish/subscribe service, you can consider connecting a number of your brokers in a *collective* using the Control Center. A collective contains a number of brokers that are all physically interconnected (that is, each broker in the collective is able to connect directly through the network to every other broker in the collective). All the broker queue managers must be connected by pairs of MQSeries channels.

A collective optimizes the publish/subscribe messages in your broker domain by reducing the number of clients per broker, without increasing the hops taken by any message by more than one. In this way, collectives are more efficient than a tree hierarchy.

You can also connect collectives to other collectives, and to other individual brokers. If you are connecting one collective to another collective, or to a stand alone broker, only one broker in each collective must provide the connection.

Messages published to any one broker are propagated to all connected brokers (whether or not they are in a collective) to which an application has subscribed to the message’s topic or content.

Figure 3 illustrates a collective of three brokers.

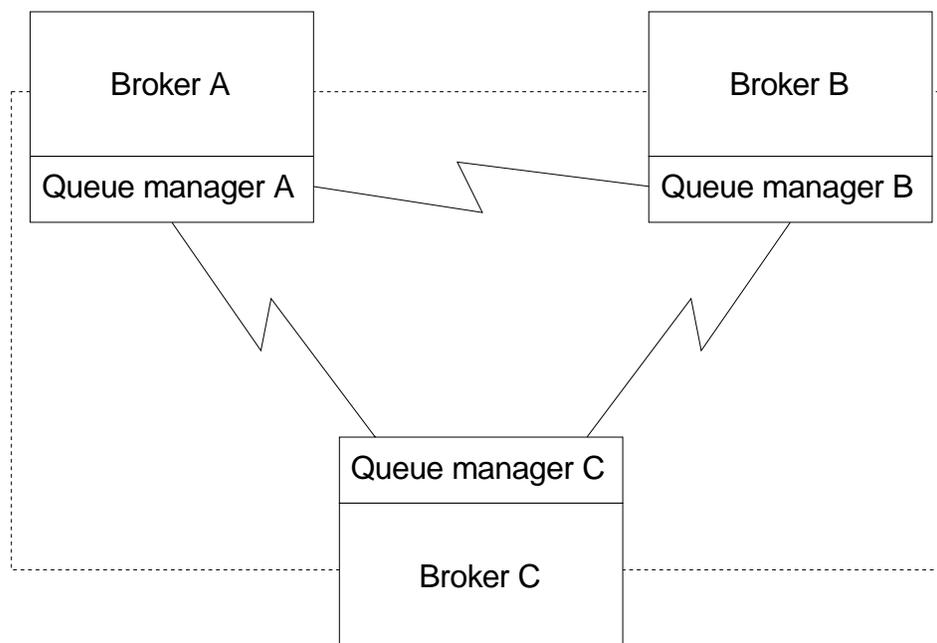


Figure 3. A collective

System management interfaces

The brokers also provide a service for independent system management agents. This enables a central management facility to access information about any network that includes an MQSeries Integrator broker domain.

This support ensures that existing system management agents, such as those developed by Tivoli, can be extended to include MQSeries Integrator resources. You can find information about using the Tivoli interface with MQSeries Integrator on the product CD. *Tivoli support is not available on the HP-UX platform.*

MQSeries Integrator brokers publish event messages, using fixed topics, in response to configuration changes, state changes, and user actions such as subscription registrations.

A system management agent can subscribe to these topics, or to a subset of these topics, to receive the detailed information about activity and state changes in the MQSeries Integrator broker domain. The event messages have a fixed structure, defined in *XML (Extensible Markup Language)*.

For further details of this support, see the *MQSeries Integrator Administration Guide*.

Business processing rules (message flows)

You define the processing for your messages as a set of actions, or rules, executed between receipt of the message by the broker, and delivery of the message to the target applications. Each action, or subset of actions, is performed by a *message processing node*. These nodes are grouped together in a sequence to form a message flow. A particular message flow provides a particular service, that is implemented by the rules that the message flow nodes contain.

Message flows

Creating message flows

You can create message flows by selecting and connecting message processing nodes, using the Control Center. MQSeries Integrator supplies a number of predefined message processing node types, known as *IBM primitives*. These provide basic functions including input, output, filter (on message data content), and compute (manipulate message content: for example, add data from a database). You can connect one node to another (the *output terminal* of the first node and the *input terminal* of the second node) to form a sequence.

The primitives nodes are described in “Chapter 4. Message flows” on page 41. You can include these primitive nodes in your message flows to define the processing you need for each of your messages. If you need additional or alternate function not provided by the primitives, you can create new node types, using a system programming interface supplied by MQSeries Integrator. This interface is described in the *MQSeries Integrator Programming Guide*.

Message flows can range from very simple, performing just one action on a message, to complex, providing a number of actions on the message to transform its format and content.

Within a message flow, you can define the action to be taken according to the *message template*, the message topic, or the data within the message itself. Alternatively, the identity of the message originator, or the destination to which the message is sent, might be important. Any combination of one or more of these attributes, or others, can define the rules by which the messages are processed, and determine the sequence of nodes you put together to form the message flow.

A message flow can process one message in several ways to deliver a number of output messages, perhaps with different format and content, to a number of target applications. You can embed one message flow within another, enabling you to reuse a particular sequence of nodes, that provide a commonly required function, many times.

You can request that the actions taken within a message flow are assured by the implementation of XA technology. That is, all actions succeed or are rolled back to preserve the integrity of your message processing. If the actions taken by your message flow include updating a database, you must use a DB2 database (DB2 or Oracle on MQSeries Integrator for Sun Solaris) to take advantage of this coordination. For more information about transaction coordination, see “Transaction support” on page 71.

If you do not request coordination, or you are not using DB2 (DB2 or Oracle on MQSeries Integrator for Sun Solaris) for your external database, MQSeries Integrator commits or rolls back each action taken by the message flow but cannot assure that success or failure is reflected by all actions.

Figure 4 on page 15 illustrates the components of a message flow.

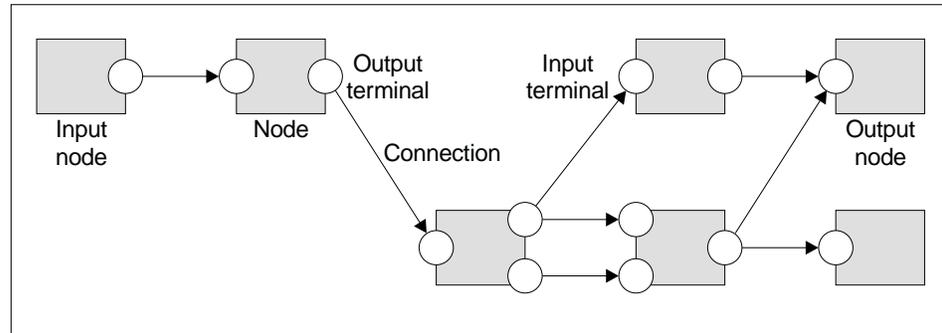


Figure 4. Message flow components

Message flow input and output

The message flows you create receive messages at *input nodes*. Every message flow must have at least one input node.

The input nodes must be one of the supplied primitives: **MQInput**, **MQeInput**, or **SCADAInput**. MQInput nodes represent MQSeries queues, which can be unique to this node, or used to supply messages to multiple nodes. MQeInput nodes represent MQSeries Everyplace queues. SCADAInput nodes represent SCADA input ports.

The sequence of nodes in a message flow usually end with one or more *output nodes* that put one or more messages to one or more queues that are read by applications that want to receive messages processed by that message flow. SCADA message flows end with a Publication node, which knows how to handle SCADA messages.

Several primitive output nodes are supplied, such as **MQOutput**, **MQeOutput**, **MQReply** (that uses the reply-to queue), and **Publication**. (In exceptional circumstances you can use the **SCADAOutput** node as a stand-alone node. In normal circumstances you can use it as a sub-node within the Publication node.) These nodes also represent unique or shared MQSeries or MQSeries Everyplace queues. The queues for published messages are specified by the applications that have registered an interest in the information available, although SCADA output messages go directly to the specified SCADA port without being put to an output queue.

Other message flows might simply store the message in a database for later processing, and not use an output node at all.

Publish/subscribe services

Message flows that incorporate a *publication node* provide a particular service, known as a publish/subscribe service. Messages are supplied to the message flow by *publishers* (applications that publish messages), and retrieved from the message flow by *subscribers* (applications that have registered a subscription with a broker: the subscription defines their interest in published messages).

A single message flow can include more than one publication node. Any number of nodes can be included between the input nodes and the publication nodes, but you cannot define any node to follow the publication node.

Message flows

Each publication node has a *subscription point*. A subscription point differentiates the publication node from other publication nodes on the same message flow, and therefore represents a specific path through the message flow. For example, a message including a share price might be needed in both dollars and sterling. The message is processed, and two messages generated, one with the dollar price, the other in sterling. The subscribers register specifying the identification of the subscription point of the publication node that provides the currency they require.

You can include an unnamed publication node (one that does not have a specific subscription point) in your message flow: this is known as the default subscription point.

You can find out more details about publish/subscribe applications in “Applications and clients” on page 22.

Associating message flows with brokers

When the broker has been defined to the broker domain topology, you can assign a message flow to one of the broker’s execution groups. The same message flow can be assigned to multiple brokers. Each message flow executes in an execution group: each execution group is isolated from all others to increase data integrity within the broker.

From the *Assignment* view of the Control Center you can drag and drop the message flows you have created to the execution group in which they are to execute. Each execution group can host multiple message flows.

Simple message flow examples

Here are a few simple message flows that use the primitives nodes.

1. **MQInput->Compute->MQOutput**. The compute node transforms a message from one format to another, so that sending and receiving applications can communicate with each other in their own formats.

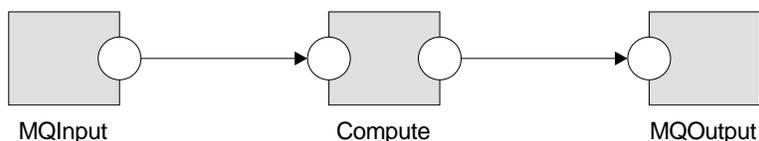


Figure 5. A simple message flow: case 1

2. **MQInput->Filter->MQOutputA or MQOutputB**. A message is routed to application A, or application B, depending on the contents of the message.

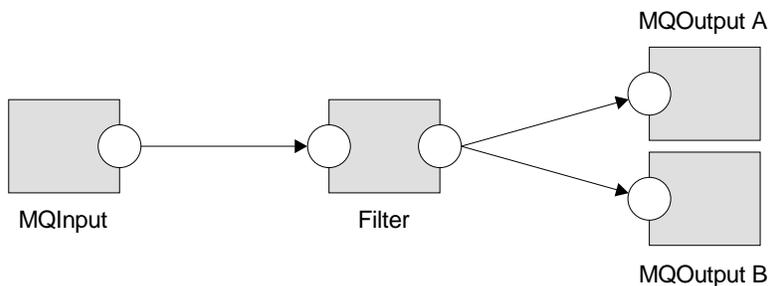


Figure 6. A simple message flow: case 2

3. **MQInput->Database->MQOutput.** The Database node stores a copy of a message in the database, or updates the database with information from the message.

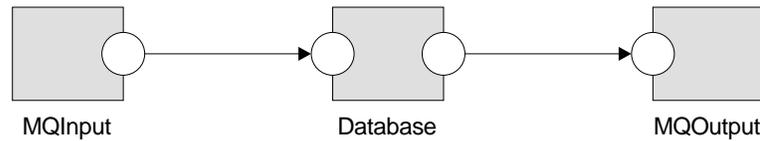


Figure 7. A simple message flow: case 3

4. **MQInput->Publication.** This publish/subscribe service sends publications to registered applications. Applications register with the publish/subscribe service, and are sent the relevant publications directly by the publication node.



Figure 8. A simple message flow: case 4

In each of these examples, you could use MQeInput or SCADAInput nodes instead of the MQInput nodes, and MQeOutput nodes instead of the MQOutput nodes. (The Publication node knows how to handle SCADA messages.) A message flow could start with an MQInput node, and end with an MQeOutput node or the SCADA functionality of a Publication node, but in that case you must have an MQeInput node or a SCADAInput node, as appropriate, in a message flow in the same execution group.

For more information on creating message flows like these, and others, and for details on the message processing node primitives and how to use them, see “Chapter 4. Message flows” on page 41.

If you want to know more about creating your own message processing nodes, see “Chapter 11. Enhancing your broker domain” on page 139.

Messages and message sets

Each message flowing through your system has a specific structure, which is important and meaningful to the applications that send or receive that message. MQSeries Integrator refers to the structure as the *message template*. Message template information comprises the *message domain*, *message set*, *message type*, and *wire format* of the message. Together these values identify the structure of the data the message contains. Every message flow that processes a message conforming to this template must understand the template to enable the message bit-stream to be interpreted.

You can use:

- Messages with a message template predefined to the message repository using the Control Center. These are referred to as *predefined messages*.
- Messages with a message template predefined to the NEON database using the NEONFormatter interface.
- Messages with a self-defining template. These are called *self-defining messages*.

Messages

Messages predefined in the Control Center

When you create a message using the Control Center, you define the fields (*Elements*) in the message, along with any special field types you might need, and any specific values (Valid Values) the fields might be restricted to (note that the Valid Values defined are for documentation purposes only and are not currently policed by the broker).

You can also create messages using the SmartGuide. This provides an easy to use interface to define simple messages, and allows you to define and arrange the fields within the message structure.

Every message predefined in the Control Center must be a member of a message set. You can group related messages together in a message set: for example, request and response messages for a bank account query can be defined in a single message set. All message and message set definitions are maintained in the message repository.

When you assign and deploy a message set to a broker, the definition of that message set is sent by the Configuration Manager to the broker in the form of a *message dictionary* (illustrated in Figure 2 on page 12). The broker can manage multiple message sets simultaneously.

Importing legacy message definitions

You can use the facilities of the Control Center to import message structures previously defined as C and COBOL data structures. The Control Center creates a message set for you in a way that is consistent with all other message definitions in the message repository.

The import facility allows continued use of messages defined in C and COBOL data structures by your existing applications that use those structures. It also enhances the existing support by giving you the flexibility to examine and modify the data in these messages using message processing nodes. You can therefore route and transform these messages using MQSeries Integrator Version 2.0.2 facilities without having to redefine them.

You can find further information on how this is supported in “Chapter 5. Messages” on page 55.

Importing messages predefined by the Control Center

If you create message templates in the message repository on one system, you can export those definitions in XML format to a file, and import them into the message repository on another system. The command `mqsimpexpmsgset` supports both export and import.

Messages predefined by the NEONFormatter

You can use messages that you have defined using the NEONFormatter with MQSeries Integrator Version 2.0.2 message flows. You can continue to use the NEONFormatter to create new definitions of message formats. These definitions are not held in the message repository, but in a separate database set up specifically for this purpose and controlled by the NEONRules and NEONFormatter Support for MQSeries Integrator.

When you want to use these message formats in the broker, you do not assign and deploy them through the Control Center, but must ensure that the broker has access to the database in which the definitions exist.

Three primitive message processing nodes provide the functionality of NEONRules and NEONFormatter Support:

- NEONRulesEvaluation node. This supersedes the NEONRules node, and has an extra route terminal.
- NEONTransform node: This supersedes the NEONFormatter node, and adds mapping operations.
- NEONMap node: This provides mapping operations, a subset of the NEONTransform node's functionality.

These message processing nodes provide processing equivalent to MQSeries Integrator Version 1.1, plus extra functionality. *Note that the NEON nodes cannot be used unless the NEONRules and NEONFormatter Support component is installed.*

Self-defining messages

You can create and route messages that are *self-defining*. These use the XML standard to provide structure to the message, so that it can be interpreted and modified. If you are using *Java Message Service (JMS)* messages (jms_map and jms_stream) these are supported as XML messages.

Self-defining messages can also be predefined in the message repository through the Control Center. This permits the use of the logical message template by nodes within a message flow. However, these message set definitions do not need to be deployed to the brokers that support those message flows.

Parsing messages

Message template information for *predefined messages* is usually included in the message header, so that the message flows recognize the messages when they receive them. Other messages might not have headers that identify the template, but you can set up your message flow input nodes to indicate the structure of messages that are processed by this message flow. If a message is not recognized, it is treated as an opaque unit, known as a *blob*. A blob can be interpreted as a string of hexadecimal characters, and can therefore be modified or examined in the message flow by specifying the location of the subset of the string.

When a message is processed by the nodes in a message flow, and its header or body is referenced by a node, the message bit-stream is decoded by a *message parser*. MQSeries Integrator supplies several message parsers that parse known message templates and message headers. These include parsers for all messages defined to the Control Center or the NEONFormatter, and generic XML messages. The complete list is given in “Message parsers” on page 59.

If you need to process and parse messages that the supplied parsers do not handle, you can create new parsers using an MQSeries Integrator system programming interface. For more details of this interface, see “Chapter 11. Enhancing your broker domain” on page 139.

Associating message sets with brokers

If you create message sets through the Control Center, you must assign them to each broker that hosts a message flow that requires them. A single definition of a message set can be used by the broker for all message flows, and does not have to be assigned to a specific execution group. The same message set can be assigned to multiple brokers. When you deploy the changes, the message set is stored in the broker as a message dictionary.

The Control Center

The Control Center interacts with the Configuration Manager to allow you to configure and control your broker domain. The Control Center and Configuration Manager exchange messages (using MQSeries) to provide the information you request, and to make updates to your broker domain configuration.

Figure 1 on page 10 illustrates the Control Center and its connection to the Configuration Manager.

You can install and invoke any number of Control Center instances in the Windows NT environment (the Control Center is not supported by any other operating system). The Control Center depends on the MQSeries Client for Java for its connection with the Configuration Manager. The Control Center can therefore be installed on the same physical system as the Configuration Manager, or on any other Windows NT system that can connect to the Configuration Manager.

The Control Center uses a client/server connection to connect to the Configuration Manager's queue manager (whether it is on the same or another physical system), which it creates dynamically using information you provide when you first invoke the program. This connection must be a TCP/IP connection.

The Control Center is structured as a number of views on the configuration and message repositories. Users can choose which set of the views are currently included by selecting one of five roles, one of which, "All roles", shows every view.

Within the boundaries of what you are authorized to do, the Control Center allows you to retrieve information selectively from:

- The message repository. This contains all the message templates that you (or any other user) have created using the Control Center, those you have created by importing legacy message definitions, and those you have imported using the **mqsimpexpmsgset** command.
- The configuration repository. This contains configuration information pertaining to all other resources within your broker domain: brokers, execution groups, collectives, message processing nodes, message flows, topics, and subscriptions.

You can use the Control Center to:

- Develop, modify, assign, debug, and deploy message flows.
- Develop, modify, assign, and deploy message sets.
- Define your broker domain topology and create collectives.
- Control *topic security* of messages by topic.
- View status information.
- Export and import resource definitions (excluding message sets).

Updates, assignment, and deployment

When you work with the configuration and message repository data using the Control Center, you can choose to view the resources that are defined, or you can create, modify, and delete those resources. You must be authorized to perform these tasks.

If you want to make any changes, you must *check out* (request a locked copy of) the resource you want to change. This allows updates to the central data to be coordinated by the Configuration Manager. The Control Center shows you which resources you currently have checked out. Once you have locked a resource, you

have exclusive control over it until you return it to the configuration repository using *check in*, or until you relinquish control by unlocking it.

When you have made changes, or have created new resources, you can save a local copy if you want. You can also check in the resources to save your changes in the message or configuration repository, if you are authorized to do so. This makes your changes visible to all other users of the Control Center.

Checking-in an object overwrites the previous version. If you need the ability to recover earlier versions of objects, you should consider downloading MQSeries SupportPac IC04 "MQSeries Integrator V2 - Change Management and naming standards examples". This SupportPac provides suggested procedures for version management and change control of MQSeries Integrator objects, and can be downloaded from the Internet URL given in "MQSeries information available on the Internet" on page 184.

When you have decided which message flows and message sets you want to use in each broker, you can assign them from the *Assignment* view. Message flows are assigned to an execution group within a broker. Message sets are assigned to the broker itself.

Following your assignment of these resources, you must also deploy these changes through the broker domain. Deployment results in the Configuration Manager sending messages and information about the changes you have made to the brokers. You can monitor the success and progress of this step using the *Operations* view and the *Log* view.

For more detailed information about check in and check out, assignment and deployment, and all the other tasks that the Control Center supports, refer to *MQSeries Integrator Using the Control Center*. This book also provides further description of the user roles and the Control Center's interactions with the Configuration Manager.

Exporting and importing resource definitions

The Control Center allows you to export definitions you have created for your broker domain topology, your topics, and your message flows. When you export these definitions, an XML file is generated containing the information retrieved from the configuration repository. You can export individual message flows, and choose whether the resources you export are saved in separate XML files or one big one. You can use definitions exported in this way to populate another configuration repository in another broker domain, by using the import function within the Control Center, specifying the XML file.

See the *MQSeries Integrator Using the Control Center* book and the online help for further information about these options.

You cannot export message set definitions from the Control Center, or import them into the Control Center. You must use an MQSeries Integrator command, **mqsiiimpexpmsgset**, to export and import message set definitions. See "Chapter 5. Messages" on page 55 for further details about message sets, and refer to the *MQSeries Integrator Administration Guide* for details of the import and export command.

Help and online Tour

The Control Center comes with comprehensive online help: it provides context sensitive information for specific assistance, and provides general help, including the *MQSeries Integrator Tour*. The Tour gives you an online overview of the MQSeries Integrator product, its components, and the Control Center interface itself.

The Tour is based on a scenario in which MQSeries Integrator is used to integrate the processes of an international company. It introduces the product in two ways:

- Providing introductory information that you can read, with links to further details in the MQSeries Integrator books and online help.
- Providing animated sequences of actions in the Control Center. For example, you can see how a message flow and message set are created using the Control Center.

To access any section of the Tour, click on the title of the section in the left-hand panel. When you run the animated sequences, there are four buttons at the bottom of the panel. These buttons let you play, pause, stop, and view a particular segment of the movie you are playing.

To use the Tour:

- Your logon user ID must be a member of one of the MQSeries Integrator groups (mqbrkrs, mqbrasgn, mqbrdevt, mqbrops, or mqbrtpic).
- You must have one of the following Control Center roles:
 - Message flow and message set developer
 - Message flow and message set designer
 - Operational domain controller
 - All roles

To run the animated sequences supplied as part of the Tour you need to have installed two plug-ins:

- Java Plug-in 1.1.2_001
- Lotus Media 2.0

If you did not install these plug-ins when you installed the Control Center, you can install them now from the TourXtra directory on the product CD.

Applications and clients

MQSeries Integrator provides support for point-to-point and publish/subscribe application communication models.

Applications generating and consuming messages in either communication mode can take advantage of the services provided by the message flows within the brokers. Sending applications must place their messages on the input resources (MQSeries or MQSeries Everyplace queues or SCADA input port) read by the message flows providing the specific service they require. Receiving applications must retrieve processed messages from the output resources (queues or SCADA output ports) to which the message flow sends them when its processing is complete.

Applications that use messages to send or receive data can communicate in several ways. Most existing messaging middleware applications use point-to-point

communications. Now, using the services supported by MQSeries Integrator Version 2.0.2, applications can exploit topic and content-based filtering in a publish/subscribe communication mode.

Point-to-point applications

MQSeries Integrator continues to support existing point-to-point applications. Typically, these applications use a request/reply or client/server model, or broadcast a message to many target applications using *distribution lists*. Others send one-way *send-and-forget* or *datagram* traffic. You can create message flows to process these messages, in any of these ways, and assign and deploy them to your brokers.

MQSeries Integrator is able to continue to support these existing applications because it supports the application programming interfaces commonly used by messaging applications today. These interfaces, the Message Queue Interface (MQI) and the Application Messaging Interface (AMI), are unaffected by MQSeries Integrator. Existing applications written to these interfaces can usually run unchanged in this new environment. You have only to define your message flows to get messages from, and put messages to, the queues already used by your applications, for the additional message processing to be completed without the applications being aware of the change.

Publish/subscribe applications

MQSeries Integrator also supports the application communication model known as publish/subscribe. In this model, applications known as publishers send messages and others, known as subscribers, receive messages. Applications can also be both publishers and subscribers.

The publishers are not interested in where their publications are going, and the subscribers are not concerned where the messages they receive have come from. The network of brokers assures the integrity of the message source, and manages the distribution of the message according to the valid subscriptions registered in that network.

If you already have applications that are written to the publish/subscribe model, and use the MQI and AMI, you can probably integrate these applications into an MQSeries Integrator broker domain without change.

You can also modify these applications, or write new ones, to take advantage of the significant enhancements in publish/subscribe processing, particularly for subscribers.

With MQSeries Integrator Version 2.0.2, your subscribing applications can now select which publications they receive based not only on the topic of the publication, but also on specific content, or both.

Every message, even one used for content-based subscriptions, must have an associated topic (specified by the publisher or defined by the input node).

Subscribers can also use the subscription point of the publication nodes in the message flows to receive messages that have followed a particular path through the message flow, and have therefore been processed in a specific way.

A topic is used to categorize the information in the message in some way that is understood by subscribers. Each topic has a structure, delimited by the forward slash character (/). The use of structuring creates the topics in a *topic tree*, in which

Applications

each node topic attaches to the branch that contains the previous structure level. The top level topic is known as the *topic root*.

A topic can be associated with the publication message by the publisher. You can also specify a topic on the input node of your message flow: it is set as a *property* of the node and is associated with a message when it arrives in the message flow providing the publish/subscribe service. In the latter case, the topic defined by the input node is used to determine the publication's routing, but is not passed on to the subscriber. Messages without explicit topics are currently treated as local only and are not sent to other brokers in the topology.

If the publisher does not provide a topic, and the input node is not set up to define a topic where one is needed, the Publication node treats the message as an error and it is handled in whatever way you have determined in this message flow.

Client connections to brokers and message flows

All MQSeries Integrator applications, like MQSeries applications, can use all the supported MQSeries interfaces to put messages to the message flow queues. In fact, every MQSeries application is a potential MQSeries Integrator application, and vice versa.

The applications can be connected as clients to any queue manager in the MQSeries network, or can execute on the same system as the broker's queue manager, and connect locally. Figure 9 illustrates three applications connecting to a broker.

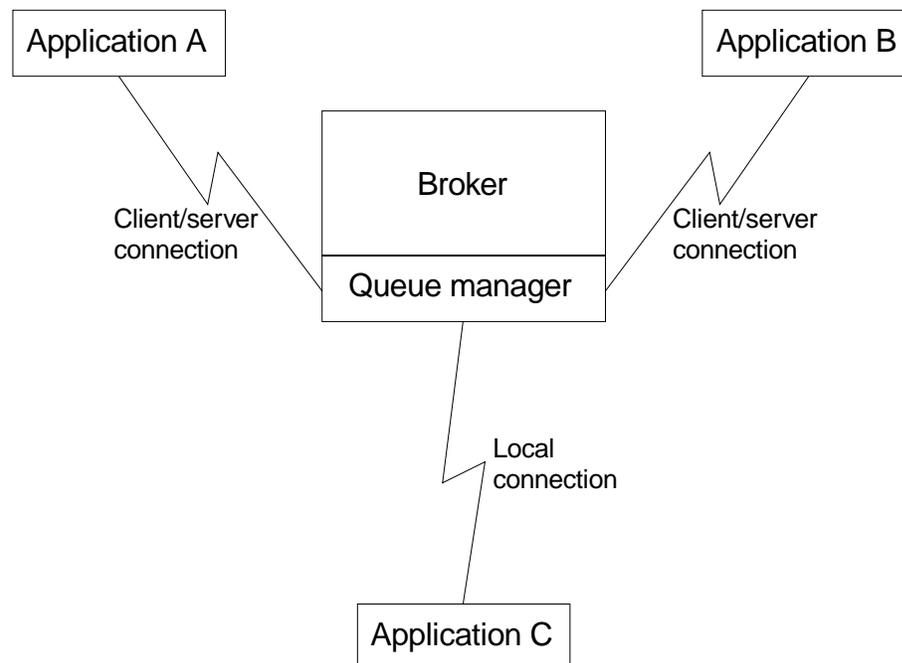


Figure 9. Applications connecting to a broker

Receiving applications can get the messages put to the output queue or queues of a message flow when they have been processed by that message flow. The applications must be connected, either by a client/server connection, or via a local connection, to the queue manager that owns the queue or queues defined as the

target for their messages. If the message flow provides a publish/subscribe service, the publication node puts the messages to the queue specified by the subscriber as its local receiver queue.

Special considerations for MQSeries Everyplace and SCADA

MQSeries Everyplace and the SCADA protocols are specifically designed for use with pervasive computing devices. MQSeries Everyplace is typically used on small, handheld devices, such as mobile phones and PDAs. SCADA is also used on small footprint devices, but these tend to be remote, unattended sensors and controllers of levels, flowrates, or temperatures, for example.

MQSeries Everyplace and SCADA applications work in rather different ways to 'normal' MQSeries Integrator applications — there are different concepts and procedures involved in setting up and configuring an MQSeries Integrator system to operate with MQSeries Everyplace and SCADA.

MQSeries Everyplace and SCADA are described separately. This is a very brief summary — if you want to use either of them with your MQSeries Integrator applications, refer to SupportPac ID03, "MQSeries Integrator - Working with MQSeries Everyplace and SCADA" which can be downloaded from the Internet URL given in "MQSeries information available on the Internet" on page 184.

MQSeries Everyplace applications

If you want to use MQSeries Everyplace messages with MQSeries Integrator, you will need an understanding of how MQSeries Everyplace works. MQSeries Everyplace is configured through the use of 'devices' and 'gateways'. The 'device' in this case is a computer running the MQSeries Everyplace device code. This code is supplied as part of the MQSeries Integrator installation. If a device is running on the same machine as an MQSeries Integrator installation, and the location of the appropriate jar files are defined in the CLASSPATH environment variable, it is *not* necessary to install the MQSeries Everyplace device code separately.

An MQSeries Everyplace gateway is a computer running the MQSeries Everyplace gateway code. Gateways are primarily used as servers to simultaneously attach multiple MQSeries Everyplace devices. Gateways are also the mechanism through which an MQSeries Everyplace network exchanges messages with an MQSeries network, using part of the MQSeries Everyplace code referred to as the MQSeries Bridge code. The MQeInput node provides access to MQSeries Everyplace gateway function.

Devices use dynamic channels (so called to distinguish them from the MQSeries client and messaging channels) to communicate. Dynamic channels are bi-directional, and support both synchronous and asynchronous messaging. Gateways necessarily also support dynamic channels, to communicate with devices, and optionally support MQSeries client channels to communicate with MQSeries servers.

Unlike MQSeries messages (which are defined as byte arrays with a message header and a message body), MQSeries Everyplace messages are all passed as Java objects, typically objects of the MQeMsgObject class. This class does not put any restrictions on the fields it can contain, and so only predefined fields are transferred to the MQMD when the message is passed to an MQSeries network — the remaining fields are put, 'unparsed', in the message body. You should use this class if it is not necessary to parse the payload of the message.

Applications

An alternative class, `MQeMMsgObject`, is specifically designed to map to an MQMD — any undefined fields are ignored. This class is more restrictive than `MQeMsgObject`, but it does allow you to define the payload of the message so that it can be parsed by nodes in a message flow — for example, to be stored in a database. These two message classes produce different behaviors, and each has advantages in particular circumstances.

You can incorporate MQSeries Everyplace applications into MQSeries Integrator by getting input from an `MQeInput` node which has defined an MQSeries Everyplace bridge queue listening on a particular port. The MQSeries Everyplace queue manager runs as part of MQSeries Integrator. Returning output to MQSeries Everyplace is achieved through the `MQeOutput` node, which puts messages destined for MQSeries Everyplace on an MQSeries Everyplace queue. If appropriate, the `Publication` node can also be used for output to MQSeries Everyplace, because it embeds an `MQeOutput` node. All MQSeries Integrator message flows that incorporate an `MQeOutput` node must also have an `MQeInput` node, which is used to configure MQSeries Everyplace.

The SupportPac ID03 gives further details of the requirements needed to achieve intercommunication between MQSeries Everyplace and MQSeries Integrator. For further details of how to program with MQSeries Everyplace, you should refer to the MQSeries Everyplace library. *Whitepaper: MQSeries Everyplace for Windows Version 1.1* provides a useful overview of MQSeries Everyplace. This Whitepaper is available by following the Library-Whitepaper links from the Internet URL given in “MQSeries information available on the Internet” on page 184. For information specific to the use of MQSeries Everyplace nodes in MQSeries Integrator, refer to *Appendix D. “MQSeries Everyplace Nodes”* in the *MQSeries Integrator Programming Guide* and *Chapter 5. “Working with message flows”* in the *MQSeries Integrator Using the Control Center* book.

SCADA applications

The device protocol, *MQIsdp*, is specialized for gaining access to a message broker, typically from remote devices characterized by requiring a low bandwidth communication. It employs TCP/IP for communication and uses a publish/subscribe communications model. A typical system might comprise several hundred client devices communicating with a single MQSeries Integrator broker, with each client identified by a unique ID.

You can incorporate SCADA applications into MQSeries Integrator by getting input from a `SCADAInput` node. But because SCADA is solely a publish/subscribe protocol, you would normally use the `Publication` node (which embeds a `SCADAOutput` node) to return output to the SCADA application. Only in advanced applications might you need to use the `SCADAOutput` node directly — for example if you want to write your own publication node.

Unlike both MQSeries and MQSeries Everyplace, SCADA does *not* incorporate any form of security, although you can encrypt data if necessary.

SCADA has a concept of ‘Quality of Service’ (QoS), similar to persistence in MQSeries:

- QoS0: “At most once” delivery. Delivery is not assured; no acknowledgment is expected.
- QoS1: “At least once” delivery. Successful delivery is assured and an acknowledgment sent.

- QoS2: "Exactly once" delivery. As for QoS1, but the message is assured not to be duplicated.

The SCADAInput node listens on a defined port and receives messages from clients using the MQIsdp protocol. A SCADA listener can be started and stopped using a publish message with a specific topic. This can be done for all ports, or for a single port identified in the message.

The Publication node filters and transmits the output from a message flow to subscribers who have registered an interest in a particular set of topics. Normally, this would be done by putting the message to the queue on the queue manager specified in the subscription. But because the Publication node contains a SCADAOutput node, the message can just as easily be delivered to a subscribing SCADA client over TCP/IP.

The SupportPac ID03 gives further details of the requirements needed to achieve intercommunication between MQSeries Integrator and SCADA. For details of the SCADA protocol, see *Appendix C. "MQSeries Integrator SCADA Device Protocol"* in the *MQSeries Integrator Programming Guide*, and for information specific to the use of SCADA nodes in MQSeries Integrator, refer to *Chapter 5. "Working with message flows"* in the *MQSeries Integrator Using the Control Center* book.

The User Name Server

If you plan to deploy message flows that provide a publish/subscribe service to your applications, you might want to employ topic-based security. Topic-based security gives you the ability to control the authority of applications, identified by the user ID under which they are executing, to publish on topics, to subscribe to topics, and to request persistent delivery of messages on topics.

To implement topic security, you must install, create, and start one *User Name Server*. (Under exceptional circumstances you might consider installing more than one, subject to your license agreement: this is discussed in "Employing topic-based security" on page 112.)

The User Name Server can be configured on any supported platform. Within the environment in which it is running, the User Name Server monitors the underlying security subsystem (for example, the Windows NT User Manager), and provides information about the valid *principals* in the system. (Principal is a general term for users and groups of users.) The User Name Server shares this information with your brokers and the Configuration Manager, and updates it at frequent intervals.

When you create the User Name Server, the following resources are also created:

- A set of fixed-name queues, defined to the queue manager that hosts the User Name Server. You must identify this queue manager when you create the User Name Server, and it must exist on the same physical system. It is created when the User Name Server is created, if it does not already exist. The User Name Server can share a queue manager with the Configuration Manager, or with a single broker, or both, if supported by the product you have purchased. (For a summary of which components can be installed on which operating systems, see Table 4 on page 95.)

Figure 10 on page 28 illustrates the place of the User Name Server in the broker domain.

User Name Server

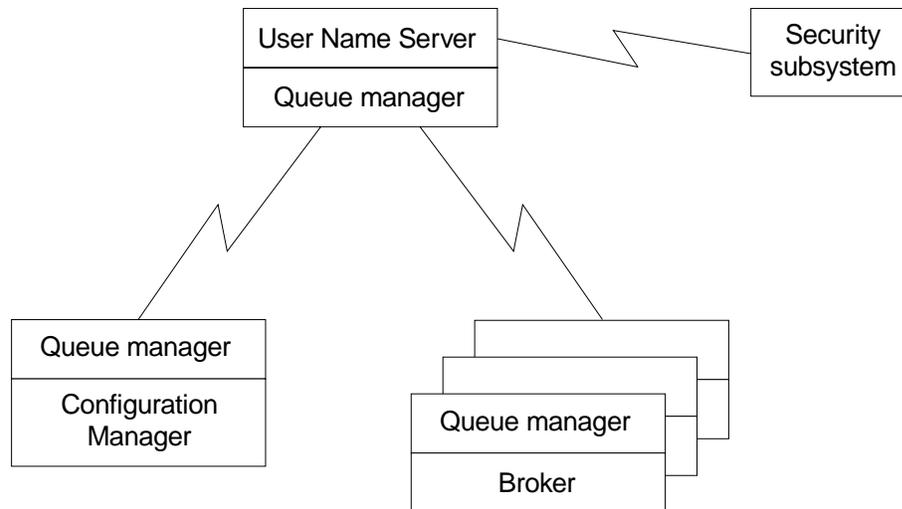


Figure 10. The User Name Server

If you do not plan to support any publish/subscribe services in your brokers, or you are willing to let every client have full access to all topics, you do not need to consider topic-based security, nor do you need to install and create a User Name Server in your broker domain. However, if it is possible that your requirements will change later, it is easier to include a User Name Server in your broker domain when you first design it. If you set global access (to all users) at the highest topic level (the topic root), this is equivalent to having no specific topic-based security. You can then introduce topic-based security on a more selective basis when you need to do so.

MQSeries Integrator relies on the operating system's security control mechanism to define and maintain definition of principals.

- On Windows NT, the User Manager supports the definition and deletion of principals (users and groups), and the assignment of user IDs to groups.
- On UNIX systems, the basic user/group control (in the file system) supports creation, deletion, and modification of users and groups, and the assignment of users to groups.

The User Name Server interrogates the operating system and makes the principals information available to other components in the broker domain.

For more information about configuring a User Name Server in your domain, and deploying topic security, see "Chapter 9. Planning your MQSeries Integrator network" on page 107.

Access Control Lists

If you want to implement topic-based security, you must define *Access Control Lists (ACLs)*. You can create and maintain ACLs in the *Topics* view of the Control Center. This view provides a display of the valid principals in your broker domain, and allows you to associate these principals with specific topics. You are also able to view the complete set of defined topics using this view.

You can create an explicit ACL for any topic in the topic tree, up to and including the topic root. An ACL allows, denies, or inherits the authority to publish, to subscribe, and to request persistent message delivery. If any topic does not have an explicit ACL, it is governed by the ACL it inherits from its higher level (parent)

topic in the tree. The default ACL setting for the topic root is to allow public access. This can be modified to restrict access by introducing ACLs at specific points in the tree.

MQSeries Integrator also supports applications publishing messages on topics created dynamically. If this option is used, the ACL applied is inherited from the closest topic above it in the tree. For example, if the topic “Stock/IT” is defined in the topic tree with an ACL, and a publisher publishes a message with topic “Stock/IT/IBM” which is not defined in the topic tree, the ACL for the parent of that topic is inherited. Therefore if this publisher is not allowed to publish on that topic, it is prohibited from publishing on the dynamic topic, too.

For more information about publish/subscribe applications, and the use of topics and ACLs, see “Chapter 7. Designing publish/subscribe applications” on page 75.

Dependencies

A number of dependencies have been highlighted by this discussion of MQSeries Integrator and its components. These dependencies are summarized here, to help clarify the requirements that MQSeries Integrator has on your systems. For details of software levels for other products (databases and MQSeries), see “Chapter 8. System requirements” on page 95.

MQSeries dependencies

MQSeries Integrator is heavily dependent on the facilities of MQSeries messaging to provide connectivity, message integrity, and some transactional support. In summary, these dependencies are:

- Queue managers. A single MQSeries queue manager can host at most one broker. The Configuration Manager and the User Name Server both depend on a queue manager, but can share this queue manager with each other, or with a single broker, or both.
- Communications. When you set up a network of queue managers to support MQSeries Integrator, you must define their connectivity. You can use any one of the communications protocols supported by the underlying MQSeries product (this varies according to operating system environment).

The client/server connection between the Control Center and the Configuration Manager, however, is limited to a TCP/IP connection, and the Control Center depends on the MQSeries client for Java.

- The Configuration Manager depends on a queue manager, with a set of fixed-name queues and a server connection channel that is defined when it is created.

The Configuration Manager also needs sender and receiver channels to be able to communicate with every broker in the broker domain (except the one broker, if defined, that is created with the same host queue manager).

- Each broker depends on a dedicated queue manager (a broker **cannot** share a queue manager with another broker, although it can share a queue manager with the Configuration Manager, or the User Name Server, or both). It also needs a set of fixed-name queues that are defined when the broker is created.

The broker needs sender and receiver channels to be able to communicate with the Configuration Manager. It also needs sender and receiver channels to communicate with the User Name Server, and sender and receiver channels to communicate with all brokers in the same collective, or to which it is identified as a neighbor in the topology.

Dependencies

- Each MQSeries application using MQSeries Integrator services must be able to connect to a queue manager in the MQSeries network to allow it to put messages to the queue serviced by the message flow that provides the service it requires. This connection can be local, or can use any supported MQSeries client product, with the appropriate server and client connection definitions.

Each MQSeries application retrieving messages from a queue populated by a message flow must be able to connect to the queue manager that owns that queue (which can be local or remote to the queue manager that hosts the message flow putting the message). This connection can be local, or can use any supported MQSeries client product, with the appropriate server and client connection definitions.

If the MQSeries application retrieving messages is a subscriber to a publish/subscribe service, the messages it receives are propagated to the broker to which it has subscribed, regardless of the proximity of the broker (and its queue manager) that hosts that publish/subscribe service.

- SCADA and MQSeries Everyplace message flows containing SCADA and MQSeries Everyplace nodes must reside in a single execution group within the broker. If a message flow uses an MQeOutput node there must be an MQeInput node in a message flow in the same execution group. If a message flow uses the SCADA functionality of a Publication node there must be a SCADAInput node in a message flow in the same execution group.
- The User Name Server depends on a queue manager, with a set of fixed-name queues defined when it is created. It can share a queue manager with the Configuration Manager, or a single broker, or both.

The User Name Server also needs sender and receiver channels to be able to communicate with the Configuration Manager, and with every broker in the broker domain to which it provides principal definitions (except to the Configuration Manager, or one broker, or both, with which it shares its host queue manager).

Further information on these dependencies is provided in “Chapter 9. Planning your MQSeries Integrator network” on page 107, and full details of exactly which component of MQSeries Integrator depends on which MQSeries component, and the software levels supported, are provided in the *MQSeries Integrator Installation Guide* for your product.

Database dependencies

The MQSeries Integrator components use databases to store configuration and operational information. In summary, these dependencies are:

- The Configuration Manager needs two independent sets of tables to support the message repository and the configuration repository.

These tables are created and initialized when the Configuration Manager is created. The two repositories can be created within a single database, or in two separate databases. Both repositories must be created in a DB2 database.

The Configuration Manager can use either a local connection to the databases, or a remote connection.

- Each broker needs access to a set of tables to support its operation.

These tables are created and initialized when the first broker is created. The broker tables can be created in the following databases:

- IBM DB2 Universal Database
- Microsoft SQL Server (Windows NT only)
- Oracle
- Sybase

For more information about supported databases, see Table 5 on page 105.

If you are using DB2 and your broker is on Windows NT, the broker tables can be created within the same database as the configuration repository, or the message repository, or both. However, three separate databases (for the broker tables, the configuration repository, and the message repository) is the preferred arrangement for easier restart and recovery.

When you create subsequent brokers, they can share the same set of tables, because every entry on each table (row) identifies an individual broker. If you prefer, you can set up separate databases (and therefore sets of tables) for each broker.

The broker can use either a local connection to the databases, or a remote connection.

You can find instructions that tell you how to create these databases, the ODBC connections they require, and the need for Microsoft Data Access Component (MDAC), in the *MQSeries Integrator Administration Guide*.

The actions supported by the Control Center provide the only interface you have to the database tables used by MQSeries Integrator. You must not access these tables directly using any other means, or you risk destroying the integrity of that data.

Further information on these dependencies is provided in “Chapter 9. Planning your MQSeries Integrator network” on page 107, and exact details of database product levels supported are provided in the *MQSeries Integrator Installation Guide* for your product.

Release to release migration

If you are migrating to MQSeries Integrator Version 2.0.2 from Version 2.0 or from Version 2.0.1, you must refer to the `Readme.txt` file that is provided on the product CD. This gives the latest information about migration requirements. You must also be aware of the following points:

- The Version 2.0.2 Control Center only operates if the Configuration Manager is also at Version 2.0.2.
- You are recommended to upgrade all instances of the Control Center to Version 2.0.2 when you upgrade the Configuration Manager to Version 2.0.2.
You must delete and recreate the Configuration Manager when you migrate it to Version 2.0.2. This makes available the new nodes and message sets supplied with the product.
- You need to migrate NEON Rules and NEON Formats in databases used with Version 2.0 or 2.0.1 to the new format that is compatible with the updated NEONRules and NEONFormatter support included with Version 2.0.2.

For information about migrating to Version 2 from Version 1, or from MQSeries Publish/Subscribe, see “Appendix A. Planning for migration and integration” on page 143.

Release to release migration

Chapter 3. MQSeries Integrator: a business scenario

This chapter uses a hypothetical scenario to explore a company's existing business problems, and to summarize the ways in which MQSeries Integrator solves those problems.

This scenario illustrates the deployment of MQSeries Integrator, and refers to the components and concepts discussed in "Chapter 2. MQSeries Integrator overview and concepts" on page 9, enhancing your understanding by providing a practical application of those concepts.

However, this is just one scenario that has been chosen for more detailed examination. Other scenarios might include:

- The financial industry, which needs to make sure that vast numbers of transactions happen, happen correctly, and that they happen once and once only.
- The health-care industry, which needs accurate information available across multiple, heterogeneous, and often legacy, systems.

The retail scenario, and others, are available by downloading the MQSeries SupportPac IC03. The Internet URL is given in "MQSeries information available on the Internet" on page 184.

The retail scenario

SRU Corporation (SRU) is a chainstore that sells food. It has expanded rapidly in the last three years with new branches opening in Amsterdam and London, and it has greatly extended its range of products.

The company headquarters are in Vancouver, Canada. Its current warehouse branches are in San Diego, California and Santiago, Chile, and the retail stores are spread throughout the world. Trading information from all stores needs to be available to many different members of SRU staff at different locations, using different applications. A subset of information needs to be made available to supplier companies. In the future, SRU intends to expand its business to support shoppers on the Internet, and wants to introduce a loyalty card system.

SRU wants to bridge the gap between its existing applications and the increased number of back-end systems. It also wants to ensure that access to some information can be restricted to those applications that need it. It also needs a solution that can be enhanced in the future. MQSeries Integrator can provide the facilities that meet these major objectives:

- **Bridging the gap:** MQSeries Integrator provides message transformation facilities that support the receipt of a message in one format and the distribution of that message in one or more different formats, according to the business needs of the target applications, without any application modification.
- **Restricting information:** MQSeries Integrator provides topic and content-based message routing using controls to restrict the recipients of any message.
- **Future extension:** MQSeries Integrator provides a basic framework that can be extended by parties such as ISVs. Message processing can be enriched by the

Retail scenario

inclusion of tailor-made nodes in the message flow. New message formats can be added to meet new application requirements, for example when new systems are added to the network.

Figure 11 shows the overall hierarchy of the SRU IT configuration.

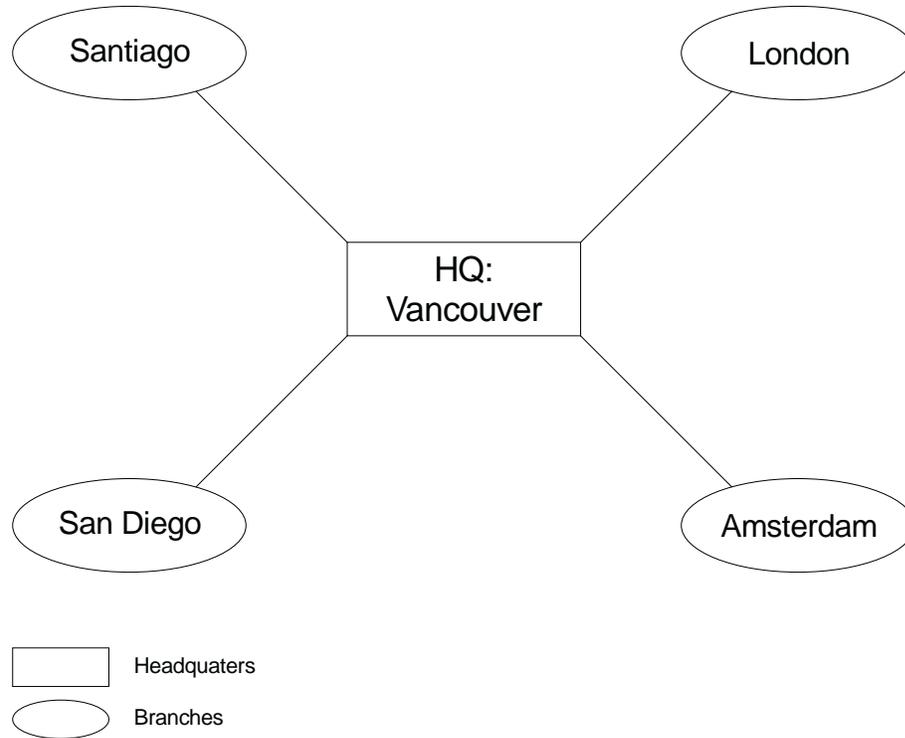


Figure 11. SRU headquarters and branch hierarchy

Figure 12 on page 35 shows the relationships between the warehouse branches and the back-end systems.

Retail scenario

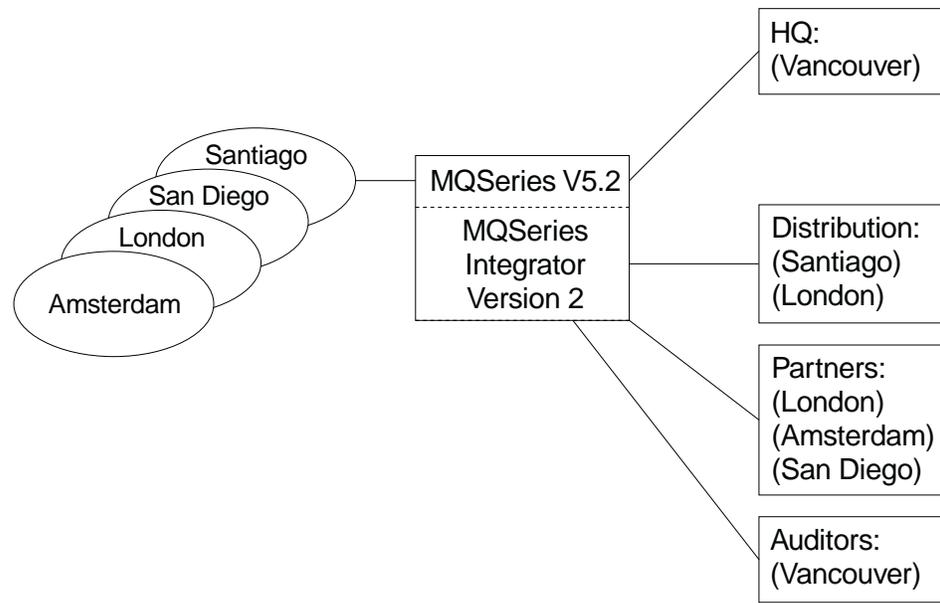


Figure 12. Branches and back-end systems

Business data

Data is taken from receipts generated for each transaction that takes place within each retail store. SRU gathers this data from its retail stores.

Figure 13 shows an example of a receipt from one of the stores:

```
SRU
SOUTHAMPTON
HAMPSHIRE

Cashier 112
Till no. 03
Date 00/04/01
Time 15:30

Purchases

1    tinned ham      3.99
    056784637
1    tinned ham      3.99
    056784637
1    garlic mash     2.92
    047388567
1    skimmed milk    1.63
    037809462

Total items      4
Multibuy         Yes
Multibuy item    tinned ham
Multibuy quantity 2
Total sales      11.46
Change           0.54
```

Figure 13. SRU receipt

Retail scenario

Business needs

From this data, SRU has the following major information needs:

1. An audit trail of all transactions in the branches. All receipt information must be **stored** for audit reasons. This information can also be used for offline data access and mining.
2. Financial reports per store per month. With the rapid growth over the last three years, sales within SRU are doing well. Headquarters are very happy with this but want to know exact figures from each store on a monthly basis. The finance department have been instructed to **gather** this information from each store at the end of each month.
3. Stock levels for products supplied for its Distribution group. As sales are doing well, stock levels must be kept up so that customers do not go elsewhere for their goods. To do this, the number of items sold must be sent to the Stock Distribution department so they are prepared to **maintain stock levels**. The Stock Distribution department is on a back-end system that uses a different operating system to that in use by the warehouse branches, so information must be formatted in such a way that it can be understood by the Stock Distribution system and applications.
4. Predicted peaks in demand for products from its partners. Headquarters want to keep track of products that are doing well so that they do not run out of them. If more than one of the same product is bought on the same transaction, this is called a “multibuy”. For each of these multibuy, headquarters want to **inform their partners** in order to ensure more of the product can be supplied.

Table 1 shows the information required by the recipients:

Table 1. Recipient information

Recipient	Information required
1. Auditors	All data from receipts
2. Finance	Total Sales per receipt
3. Distribution	Branch, Item Name, Item Quantity
4. Partners	Multibuy items

Business solution using MQSeries Integrator Version 2

The business needs listed above can be satisfied using a solution with MQSeries Integrator Version 2.0.2.

The first task is to represent the business data (that is, the receipt) as a message with a structured format:

```
Store Details
  Store Name
  Branch No
  Cashier No
  Till No
  Date
  Time
Purchases
  Item Name
  Item Code
  Item Price
  Item Quantity
Totals
```

Total Items
 Multibuy
 Total Sales
 Change

This message can be processed to handle the specific business needs. A message flow is created that takes the message from an input queue and processes the message to produce the required output messages. A simplified form of the message flow is shown in Figure 14 on page 38.

The message flow includes four subflows that perform the processing required to satisfy the business needs:

1. Initially, all messages are retrieved from the input queue by the MQInput node. Each message is checked to ensure that it is a message of the correct format for a receipt (in a Check node) and stored in a database (in a *Warehouse* node) before it is passed to all three of the remaining subflows.
2. In the **Finance flow**, the fields Branch No, Date, Time and Total Sales are extracted from the input message in an Extract node. Each message is then traced by the Trace node. The records that are written to the trace log enable you to ensure that only the data extracted by the Extract node is sent to the Finance department by the Output node.
3. In the **Stock flow**, a Compute node sums up all instances of each item from the input message. This information can then be formatted and sent in a message containing Branch No, Item Name and Item Quantity by the Output node to the Stock Distribution department.
4. In the **Partner flow**, multibuy records are placed into a database by a DataInsert node and published to registered partners by the Publication node. Partners subscribe to messages based on a topic (Multibuy) or refine their subscriptions further by filtering on the content of a field (the item name) in a message.

A reusable exception handling message flow is used to help with problem determination while the main message flow is being developed or upgraded. This exception handling flow can be embedded anywhere in the main flow.

Access Control Lists are defined in the Control Center to ensure that user IDs associated with partners are restricted to subscribing to the items they produce. For example, the partners producing the tinned ham product register a subscription based on the topic *Multibuy/tinned ham*, and are not authorized to receive messages published on any other topic.

Different applications within the partner organizations can also choose to restrict the messages they receive even further, using content based subscriptions. For example, one application might want to process only those messages which indicate that a quantity of ten or more items have been bought in one transaction.

Retail scenario

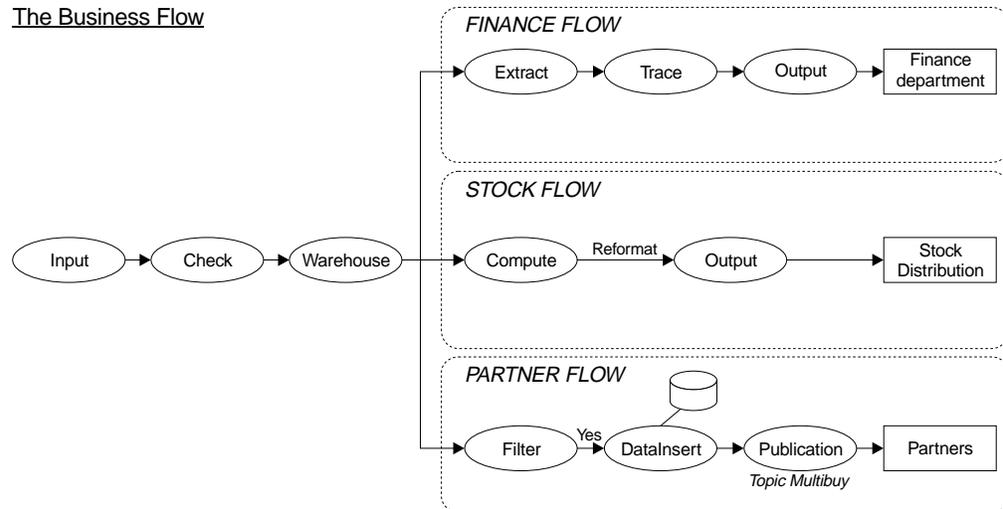


Figure 14. The business flow (simplified)

Implementing the business solution

There are several steps you need to take to implement this, or any other business solution:

1. First, you must plan your MQSeries Integrator system. For example, you must decide how many brokers you need to meet your business requirements, what message flows you need, and so on. This chapter has identified just one message flow, but a more complex setup is very likely. This book helps you to plan your MQSeries Integrator configuration, and to understand the implications for the MQSeries infrastructure, security, performance and so on.
2. Next, you must implement the MQSeries Integrator system you have designed by installing and configuring the components you need on the appropriate systems following the guidance provided in the *MQSeries Integrator Administration Guide* and in the *MQSeries Integrator Installation Guide* for your product.
3. You must define your message sets and message flows. *MQSeries Integrator Using the Control Center* has all the information you need to achieve this task.
4. You must write application programs to interact with the message flow using the message structures you have created. The *MQSeries Integrator Programming Guide* provides the information you need to do this.

If you have existing applications on the back-end systems, you must ensure that the message flow generates messages for those existing applications in the right formats, so the application source code does not have to be changed.

MQSeries Integrator Using the Control Center helps you to define the transformations you need to provide compatible message formats from your message flows.

5. Last, but not least, you must test the applications that generate the messages for your message flow, and check the results.

Part 2. Business process planning

This part helps you plan the tasks to deploy the facilities of MQSeries Integrator to meet your business objectives.

It provides the information your business planners need to understand the environment that MQSeries Integrator provides for applications. It explains the concepts that are introduced in “Chapter 2. MQSeries Integrator overview and concepts” on page 9, and gives more details about the implications of using the various functions of the product.

It contains the following chapters:

- “Chapter 4. Message flows” on page 41
- “Chapter 5. Messages” on page 55

The information here is an introduction to the detail provided in *MQSeries Integrator Using the Control Center*.

Chapter 4. Message flows

This chapter gives you more information on message flows, and how they are constructed and deployed in your broker domain. It covers the following:

- “What is a message flow?”
- “Execution groups” on page 48
- “Message flows and message sets” on page 48
- “Message flows for publish/subscribe services” on page 49
- “Supplied message flows and nodes” on page 49
- “Adding or enhancing message processing nodes” on page 54

What is a message flow?

The MQSeries Integrator message broker supports processing for messages after one application has put a message to a queue, and before another application gets that message from a queue. It provides this support by directing the message from the initial queue to the target queue (or queues) through a message flow. Note that in the case of SCADA applications, messages arrive and leave via a port rather than a queue, but the principle is the same.

The content and execution characteristics of a message flow are discussed in the following sections:

- “What does a message flow consist of?”
- “Parallel processing of message flow instances” on page 42
- “Interaction of message flows” on page 43
- “Transformation” on page 43
- “Intelligent routing” on page 44
- “Enriching message content” on page 45
- “What is a message processing node?” on page 45

What does a message flow consist of?

A message flow is a sequence of operations on a message, performed by a series of message processing nodes. The actions are defined in terms of the message format, its content, and the results of individual actions along the message flow.

MQSeries Integrator includes a range of message processing nodes, called primitives, that provide most of the function that you will need in most situations. A few of these nodes are used to illustrate the nature of a message flow in this discussion. For details of these nodes, see “Primitive node types” on page 50. For details of how you can define your own message processing nodes to extend the function available to a message flow, see “Chapter 11. Enhancing your broker domain” on page 139.

A message flow and the message processing nodes it contains describes the transformation and routing applied to an incoming message to transform it into outgoing messages. These actions form the rules by which the message is processed.

A message flow can also be made up of a sequence of other message flows, that are joined together. This function allows you to define a message flow containing a specific sequence of message processing nodes, and reuse that message flow in other message flows wherever that action is needed.

What is a message flow?

One example of why you would use this technique is error handling. You can define a message flow of one or two nodes that perform the action you want taken when an error is encountered, and include that message flow as a sub-message flow in all your other message flows.

When you complete the creation of your message flow, you can assign it for execution to one or more brokers. When you do this, the message flow must be operationally complete. That is, it must contain at least one of the supplied input nodes, such as **MQInput** node (one of the primitives). Most message flows will also contain at least one output node, such as **MQOutput**, or one **Publication** node, although this is not required (both of these nodes are also primitives).

You can choose to limit the number or the type of message flows (and therefore, by inference, the type of messages processed) to run in any broker according to the criteria you decide.

For example, you could deploy all message flows that access a particular database to a single broker. You could choose to deploy the message flows that provide a publish/subscribe service to a specialized group of brokers.

You can also control some aspects of how your message flows run, within a single broker. Each broker can host a number of *execution groups*. An execution group provides an execution environment, that offers protection and isolation.

“Execution groups” on page 48 has more information about using execution groups.

Message flows and units of work

A message flow is transactional: you can define your message flows to perform all processing within a single unit of work. Therefore the receipt of every message by the input node, and the database operations performed as a result of that message being received and processed by the message flow, are coordinated.

If an error occurs within a transactional message flow, the transaction is rolled back and the message will be handled according to normal error handling rules (described in “Error handling” on page 47).

You can also define a message flow to work outside of a unit of work if you do not want this support.

Parallel processing of message flow instances

When you define, assign, and deploy a message flow, the broker automatically starts an instance of the message flow for each input node (one or more). This is the default behavior. Each instance retrieves a message from the input node, and runs in parallel with other instances that retrieve a message from other input nodes.

If you want to further increase the throughput of this message flow, you can set a property of the assigned message flow (that is, the property is available when you have assigned the message flow to the broker’s execution group) that defines how many additional instances are to be started by the broker for that message flow. You can set properties of the input node to exercise control over the order in which messages are processed: for example, you can force all messages received from a single client to be processed in order. This is discussed further in “Message order” on page 70.

What is a message flow?

You can also increase message flow throughput by assigning more than one copy of the message flow to the same broker. However, this is only appropriate if the message order is not important, because the multiple copies of the message flow are handled independently by the broker, with no correlation between them.

Therefore, if more than one copy of the same message flow is active within the broker, each copy can be processing a message at the same time, from the same queue. It is possible for the processing time of a message flow to vary, and multiple message flows accessing the same queue could therefore read the messages from the queue in a random order. Also, the order of messages produced by the message flows might not correspond to the order of the original messages. You can influence the order in which the input node removes messages from the queue (using the Order Mode property).

You are therefore recommended to increase the instances of a single copy of the message flow if you want to increase throughput and parallel processing but wish to have control over the message order.

Interaction of message flows

In general, the message flows you define and deploy do not interact with other message flows, nor will the processing of one message by the message flow influence the processing of another message.

It is possible, however, to create message flows that do interact to achieve particular outcomes. For example, one message flow could store a message in a database: a second message flow could retrieve that message and use its contents (for example, a currency exchange rate) to influence the contents of the message currently being processed, by inserting fields, or recalculating a value.

Each instance of a message flow handles strictly one message at a time. A message flow instance does not accept a second message (that is, read a new message from the input queue or SCADA port) until the first message has been completely processed.

Transformation

Most enterprises have applications that have been developed over many years, on different systems, using different programming languages, and different methods of communication. Standard message queuing technology can bridge differences like these, but applications still need to be aware of, and negotiate, the format in which the messages flow.

MQSeries Integrator changes all that. The knowledge of each application is stored just once in the broker and each message is translated into the receiving application's format.

For example, personal names are held in many forms in different applications. Surname first or last, with or without middle initials, upper or lower case: these are just some of the permutations. Because the broker knows the requirements of each application, it can transform the message to the correct format without the sending or receiving application needing any modification.

A message flow can completely rebuild a message, convert it from one format to another (whether format means order of fields, byte order, language, and so on), remove content from the message, or introduce specific data into it. For example, a

What is a message flow?

node can interact with a database to retrieve additional information, or store a copy of the message (whole or part) in the database for offline processing.

A couple of other examples show how important message transformation can be:

- An order entry application has a Part ID in the body of the message, but its partner stock application expects it in the message header. The message is directed to a message flow that has knowledge of the two different formats, and can therefore reformat the information as it is needed.
- A data-entry application creates messages containing stock trade information. Some applications receiving this message need the information as provided, but others need additional information added to the message about the price to earnings (PE) ratio. The stock trade messages are directed to a message flow which passes the message unchanged to some output nodes, but calculates and adds the extra information for the others. The message flow does this by looking up the current stock price in a database, and uses this value and the trade information in the original message to calculate the PE value before passing on the updated message.

Intelligent routing

Intelligent routing encapsulates business knowledge of how information should be distributed between sending and receiving applications throughout the enterprise. This knowledge is stored in the broker as a set of rules that are applied to each message as it passes through the broker. Routing is independent of the requirement for message transformation, although you will usually define sets of rules (as message flows) that combine the two in some way. Messages are distributed according to criteria applied to the values of fields within the message.

For example, a money transfer application always sends messages to one other application. You decide that every message with a transfer value of more than \$10,000 must now also be sent to a second application, to enable all high-value transactions to be recorded.

In another example, a national auto club offers a premier service to specific members for orders above a threshold value. Most orders are routed through the usual channels, but if the membership number and order value meet certain criteria, the order gets special treatment.

You can also establish a more dynamic routing option by building additional routing information into the message when it is processed. Optional sets of rules are set up to receive messages according to values (destinations) set into the message. You can establish these rules such that a message is processed by one or more of the optional sets of rules, in an order determined by the added message content.

You can create, modify, and use these rules to develop a very flexible approach to the distribution of information. New ideas and requirements can be stated clearly, and turned into new or changed rules in the broker, and your business goals are met. You don't have to rework your applications.

Your business processes range from the simple to the very complex. You can create rules to cover every case, building new rules, and reusing and combining existing ones to develop even the most complex solution.

Enriching message content

When a message is processed by a message flow, it is possible to update and add to the message content. This allows you to add value between sender and receiver in any way you choose.

A typical way in which you can enhance the message content is by adding data from a database. This could be done by appending fields to the message, or merging information from the two sources, for example by calculating a new field value using the database information.

What is a message processing node?

A message processing node is a stand alone procedure defined within a message flow that receives a message, performs a specific action against it, and outputs zero or more messages as a result of the action it has taken.

This section describes the types of nodes, using the primitives included in MQSeries Integrator to illustrate the function they provide.

You can create additional message processing nodes to provide enhanced or replacement function if you choose, except where noted. For further information about this extension to MQSeries Integrator, see “Adding or enhancing message processing nodes” on page 54.

Common node characteristics

Every message processing node has a fixed number of input points and output points. These points are known as terminals. Each node normally has one input terminal (on which it receives messages), and multiple output terminals to handle a variety of situations. Output terminals are defined according to the characteristics of the individual node. For example, a filter node has *true*, *false*, *failure*, and *unknown* output terminals.

A *connector* joins an output terminal of one node to an input terminal of the next node in the message flow. You can leave an output terminal unconnected, or you can connect a single output terminal to more than one target node.

Figure 4 on page 15 illustrates the relationships between connectors, terminals, and nodes.

After a node has finished processing a message, the connectors defined from the node's output terminals determine which node, or nodes, process the message next. If a node has more than one output terminal connected to a target node, the node determines the order in which the different execution paths are executed. If a single output terminal has more than one connector to a target node, the broker determines the order in which the different execution paths are executed. You cannot change the order of processing determined by the node or broker.

A node does not always produce an output message for every output terminal: often it produces one output for a specific terminal depending on the message received. For example, a filter node will typically send a message on either the true terminal, or the false terminal, but not both.

When the processing determined by one connector has been completed, the node issues the message again to the next connector, until all possible paths have been completed. Updates to a message are never propagated to previously executed nodes, only to nodes following the node in which the update has been made.

What is a message flow?

The message flow can only accept a new message for processing when all paths through the message flow (that is, all connected nodes from all output terminals, as appropriate) have been completed.

Input and output nodes

Some message nodes have special characteristics: they define points in the message flow to which clients send messages (input nodes such as **MQInput**), or from which clients receive messages (output nodes such as **MQOutput**).

These special nodes reference MQSeries or MQSeries Everyplace queues or SCADA ports — referred to here as I/O resources. Client applications interact with these nodes by putting messages to, or getting messages from, these I/O resources.

A message flow has a set of (one or more) input nodes to which senders can post their messages, and a set of output nodes from which receivers can pick up messages.

If a message is being processed under transactional control, the output node only puts the message to the destination I/O resource when all processing by the message flow has been successfully completed, unless the output node is set up to put the message outside the global (message flow) transaction.

Before you can use a message flow, the input nodes must be associated with I/O resources that represent the sources of messages. An output node must also be associated with an I/O resource in most cases. However, you can set an output node property that causes the node to put the message to every I/O resource in a *destination list*, which is contained within the message itself.

You must use one of the supplied primitive nodes for every message flow input node: you cannot use one of your own. You should use the **MQInput** node in most cases, but you should use the **MQeInput** node when interfacing with MQSeries Everyplace applications, and the **SCADAInput** node when interfacing with remote SCADA (Supervisory Control And Data Acquisition) devices.

You can use the supplied **MQOutput** node, or you can replace it if you choose. You will need to use the supplied **MQeOutput** node for MQSeries Everyplace applications, and the **Publication** node for typical SCADA applications. (The **Publication** node knows how to handle SCADA messages — you will only need to use the **SCADAOutput** node in exceptional circumstances.)

Publication nodes are a special type of output node that use the queues identified by current subscribers whose subscriptions match the characteristics of the current message. Subscribers provide the identity of the queue on which they want to receive all matching publications.

Processing messages

All nodes other than the input and output nodes receive an input message from the previous node in the message flow and transform it into zero or more output messages to be made available to the next node (or nodes) in the message flow. Messages passing between nodes are not put to an intermediate queue: each message is held in local memory.

These nodes can perform any kind of processing on a message. For example, they can:

- Transform the message (**Compute**).
- Subset the data within the message (**Extract**).

- Route the message to one or more targets (**RouteToLabel**).
- Archive the message in a message warehouse (**Warehouse**).
- Update database information from the message content (**Database**).

Error handling

All primitive message processing nodes have a *failure* output terminal, to which a message is transferred if an error is detected within the node. If the failure terminal is not connected to a target node, an exception is generated and propagated back towards the input node:

- If a **TryCatch** node is encountered before the exception reaches the input node, the flow of control proceeds down the **catch** terminal. The message that is propagated through the catch terminal is the message originally received by the **TryCatch** node: any changes made to the message by later nodes in the message flow are not preserved. However, any external processing (for example, updates to a database through a **Database** node) are preserved. It is not possible to rollback these database updates from within the message flow.

Before the **TryCatch** node passes on the message to the node connected to the catch terminal, it adds the exception information to the *ExceptionList* item in the message tree. Existing information in the *ExceptionList* field in the message is written to the *local error log*, and then overwritten with the new exception information.

For further information about ExceptionLists, see the *MQSeries Integrator ESQL Reference* book.

- If the message reaches the input node:
 - If the input node's catch terminal is connected to another node, the message is propagated to that node. In this case, an error is not recorded in the local error log (for further details of how MQSeries Integrator logs errors, see "Problem determination" on page 134).
 - If the input node's catch terminal is not connected, and the message is being processed under transactional control, the message is returned to the input queue. An error is recorded in the local error log. The input node will then read the message again for retry. It first checks to see if the backout count for this message has now exceeded the backout threshold:
 - If the backout count has not exceeded the threshold, the message processing is retried.
 - If the backout count has exceeded the threshold, and the failure terminal is connected to another node, the message is propagated to that node.
If the failure terminal is not connected, the message is put on the backout queue, if one is defined for this input queue, or the queue manager's dead-letter queue (DLQ), if a backout queue does not exist.
If the queue manager does not have a DLQ defined, the message is left on the input queue. (If the broker's queue manager has been created by the create broker command **mqsicreatebroker**, a DLQ has been defined and enabled for this queue manager.)
 - If the catch terminal is not connected and the message is not being processed under transactional control, the message is discarded.

For more information about message processing under transactional control, see "Transaction support" on page 71.

What is a message flow?

You can provide a minimum level of error handling within every message flow you define if you choose. This minimum level might include:

- Define a dead-letter queue (DLQ) on the broker's queue manager (or use the default supplied DLQ).
- Change the queue manager's attributes to use this DLQ.

For details of incorporating more sophisticated error handling, for example, the use of the TryCatch node, see *MQSeries Integrator Using the Control Center*.

Execution groups

The broker provides the run-time environment for a set of deployed message flows: this environment is called an execution group. An execution group provides an isolated execution environment, because each is started as a separate operating system process.

One execution group, the default execution group, is set up ready for use whenever you create a broker. By setting up additional execution groups, you can isolate message flows that handle sensitive data such as payroll records, or security information, or unannounced product information, from other nonsensitive message flows.

If you create additional execution groups, you must give each a name that is unique within the broker, and assign and deploy one or more message flows to each one.

Within an execution group, the assigned message flows run in different thread pools. You can specify the size of the thread pool (that is, the number of threads) that are assigned for each message flow by specifying the number of additional instances of each message flow (this is also discussed in "Parallel processing of message flow instances" on page 42).

You do the creation, deployment, and assignment (of message flows and threads for the message flows) using the Control Center.

For example, you might want to set up one execution group to support a connected set of applications and their messages, and a second execution group for another distinct set of applications and their messages.

The broker guarantees operating isolation of each execution group, thus guaranteeing data integrity between execution groups, and improving robustness of message flows.

Message flows and message sets

When you have created your message flows using the Control Center, you must assign them to the brokers on which you want them to run, again using the Control Center. Their assignment and subsequent deployment prompts the Configuration Manager to send data and control information to the broker, enabling it to load and execute the code contained within the message flow.

The same message flow can be assigned to any number of brokers, perhaps for workload distribution. Similarly, a number of different message flows can be assigned to the same broker.

Message flows and message sets

However many message flows a broker hosts, the broker needs access to the definitions of your predefined (that is, not self-defining) messages expected or generated by those message flows.

Therefore if you assign a message flow that uses predefined messages to an execution group, you must also assign one or more message sets to that broker, to ensure the details of the messages are available when the message flow executes.

The message sets are assigned and deployed to the broker, but message flows are assigned and deployed to an individual execution group.

The relationship between message flows and message sets is unlikely to be one to one. You are very likely to have a number of related message flows executing in one broker that use some or all of the same message sets.

For further details about messages and message sets, and how you define them, refer to “Chapter 5. Messages” on page 55.

Message flows for publish/subscribe services

MQSeries Integrator supports message flows that provide publish/subscribe services. If you define a message flow to support publish/subscribe, you must:

- Define the publish/subscribe topology that identifies a broker’s neighbors, to which publications are propagated, using the Control Center. A publication is only routed to a broker if there is a subscriber at that broker who has registered an interest in the topic of that publication.
- Include a Publication node as the last message processing node in at least one path through the message flow. A message flow can have a path in which the end node is a Publication node as well as a path in which the end node is an output node. It is also possible to have more than one Publication node or output node on each path.

This node handles published messages by forwarding them on to all registered subscribers, that is to applications that have registered an interest on the topic, or content, or both, of the message at this node.

To support publish/subscribe applications, you must define at least one message flow as described above, and design applications that publish to the input queue of the publish/subscribe message flow (the publication queue, identified by the input node) and applications that register subscriptions for the published messages.

You can increase the throughput of this publish/subscribe message flow from publication queue to subscribers by increasing the number of instances of that message flow operating in the execution group. You can also deploy the same message flow to multiple brokers, as you can with any message flow.

For more information about publish/subscribe processing, see “Chapter 7. Designing publish/subscribe applications” on page 75.

Supplied message flows and nodes

MQSeries Integrator includes a number of message processing nodes and message flows that you can use.

Supplied message flows and nodes

Primitive node types

Message processing nodes perform the real work of handling the message within the message flows deployed to the broker. MQSeries Integrator incorporates a set of node types that provide basic out-of-the-box message processing function. These are known as the primitive node types, and can be considered in the following categories:

- Receiving and routing messages
- Transforming a message to an alternative representation
- Selecting a message for further processing based upon the message's content
- Interacting with an external repository to augment a message or store the whole or part of a message
- Responding to events and errors

The NEONRules and NEONFormatter Support component of MQSeries Integrator Version 2.0.2 replaces the NEONRules and NEONFormatter engines from MQSeries Integrator Version 1. You can use the NEON nodes that incorporate this function within any message flow, as you can any other message processing node. This allows you to migrate applications and messages from MQSeries Integrator Version 1. The MQSeries Integrator Version 2.0.2 Control Center also supplies the graphical user interfaces (GUIs) that support management of the rules and formats in this scheme.

Receiving and routing

Label Provide a target for routing from the RouteToLabel node. The identity of the Label node is used to match message content and thus determine the route a particular message takes at a particular point in the message flow.

FlowOrder

Define the exact order in which subsequent subflows in the message flow are executed. The order is normally unpredictable: you can use this node to force one particular path to process the message before another path.

MQeInput

Allows MQSeries Everyplace clients to connect and propagate messages into the broker via an MQSeries Everyplace queue, and retrieve messages from the MQSeries Everyplace queues. You must use this node for MQSeries Everyplace input, you cannot replace this with your own input node.

MQeOutput

Write the current message to the MQSeries Everyplace queue specified by the node properties, or defined by a destination list associated with the message, or to the reply queue specified in the message header.

MQInput

Read the next message from the input queue and establish the processing environment for this message (for example, the transactional context). You cannot replace this node with your own input node.

MQOutput

Write the current message to the queue specified by the node properties, or defined by a destination list associated with the message.

MQReply

Write the current message to the reply queue defined by the message's header or the node properties.

Supplied message flows and nodes

NEONRulesEvaluation (supersedes NEONRules)

Invoke the NEONRules engine. This provides routing function defined by the NEONRules GUI.

Note that the NEON nodes cannot be used unless the NEONRules and NEONFormatter Support component is installed.

Publication

Deliver the message to a set of MQSeries, MQSeries Everyplace or SCADA subscribers that are defined in the subscription table and have a subscription for the node's subscription point. This node also stores *retained publications* when appropriate.

RouteToLabel

Interrogate the destination list in the message to determine the identity of the Label node within the current message flow to which the message must be sent for processing. You can create or modify a destination list within the Compute node.

SCADAInput

Allows MQIsdp clients to publish and subscribe to the information within a publication node via the port specified within this node.

SCADAOutput

Write the current message to all the subscribed SCADA clients defined by the destination list associated with the message. This node is available, and typically used, as a sub-node within the Publication node. You should use it as a stand-alone node only in advanced solutions where the normal publish/subscribe environment is not suitable.

Message transformation

Compute

Derive an output message from the contents of an input message.

Output message elements (including destination lists and exception lists) can be defined using expressions, defined in ESQL, based on input message elements and external data sources. The expressions can use arithmetic operators, text operators (for example, concatenation), logical operators, and other built-in functions.

The rich subset of ESQL operations you can specify in this node is described in *MQSeries Integrator ESQL Reference*.

The output message can inherit all of the headers associated with the input message, or the node can be set up to select a subset of those headers for the input header, or to insert a new header (or headers), replacing the input message headers.

The Compute node can be used to make a copy of a message (that is, duplicate the message), prior to manipulating the message content.

Extract

Create a new message from the input message using only specified elements. Any elements from the input message that are not specified for the output message are discarded. This is a specialized form of Compute.

NEONTransform (supersedes NEONFormatter)

Invoke the NEONFormatter to map and transform (reformat) a message from a known input format to a specified output format. For details, see *NEONRules and NEONFormatter Support for MQSeries Integrator*

Supplied message flows and nodes

Programming Reference for NEONFormatter. The message definitions and transformations are defined using the NEON Formatter GUI.

Note that the NEON nodes cannot be used unless the NEONRules and NEONFormatter Support component is installed.

NEONMap

This node provides the map function of the NEONTransform node, but no output operations are carried out.

Note that the NEON nodes cannot be used unless the NEONRules and NEONFormatter Support component is installed.

ResetContentDescriptor

Reparses the bit stream of an input message. This node provides equivalent function to an MQOutput node followed by an MQInput node. It allows the message to be interpreted by an additional message parser within a single message flow.

Selecting a message based on content

Check

Test one or more of the message properties within the message template. The exact nature of the test is defined by the node properties.

The Check node validates the message properties in the message header against values specified in the node: it does not check the message body. If the test fails, the message is passed to the failure terminal.

Filter Route the message according to message content, using a filter expression specified in ESQL. In addition to including elements of the message or message properties, the filter expressions can also reference data held in an external database.

The rich subset of ESQL operations you can specify in this node is described in *MQSeries Integrator ESQL Reference*.

Interacting with an external database

Database

Modify the content of one or more database tables using a specified ESQL expression. The expression can contain elements derived from the input message. The input message is propagated along the message flow without change. The Control Center provides customized nodes for simple tasks such as inserting a single row into a table (**DataInsert**), updating a single row in a table (**DataUpdate**), and deleting a single row from a table (**DataDelete**).

The rich subset of ESQL operations you can specify in this node is described in *MQSeries Integrator ESQL Reference*.

Warehouse

Write a message to a data warehouse. This is a specialization of the Database node that inserts a single row into the specified database table, and provides additional options for time-stamping the message and including the entire message content as a blob.

Recording and responding to events and errors

TryCatch

Provide a special handler for exception processing. The input message is

Supplied message flows and nodes

initially propagated on this node's try terminal. If an exception is thrown by a downstream node it is caught by this node, which then propagates the original input message on its catch terminal.

Throw

Provide a means to throw an explicit exception in a message flow.

Trace

Write a specified expression (that can include the content of the fields in the message) to the user trace log, or to a file specified as a property of the node.

Supplied message flows

A set of message flows are supplied with the product. These are in two broad categories:

1. Default message flows. These are:

- The Version 1 Migration/Compatibility flow for MQSeries Integrator Version 2.0.2.

This message flow can be deployed in any Version 2.0.2 broker in your broker domain to provide equivalence to an MQSeries Integrator Version 1.1 daemon. It incorporates the NEONRulesEvaluation node to process messages according to the NEON rules engine. An input node, to read messages from an input queue, and a set of output nodes, that provide failure, no-hit and process action functions, are connected to the NEONRulesEvaluation node.

- The publish/subscribe message flow.

This message flow provides the simplest message flow processing that provides a publish/subscribe service. It emulates exactly the basic publish/subscribe function supported by the MQSeries Publish/Subscribe SupportPac, and is equally appropriate for all publish/subscribe services in which no additional processing of the message content is required.

2. Installation verification message flows. These are:

- The *ScribbleInversion* message flow. This is the message flow required by the Scribble application, described in the *MQSeries Integrator Installation Guide* for your product.
- The *Soccer* message flow. This is the message flow required by the Soccer Results Service, described in the *MQSeries Integrator Installation Guide* for your product.
- The *Postcard* message flow. This is the message flow required by the Postcard application, described in the *MQSeries Integrator Installation Guide* for your product.

The definitions of these message flows are provided in the import file `SamplesWorkspaceForImport`. The import file `PostcardMS.mrp` provides the definitions for the message set required by the Postcard message flow. Each *MQSeries Integrator Installation Guide* describes the Installation Verification Program (IVP) message flows and message set in detail, and tells you how to import the supplied files and save the definitions for future use. *MQSeries Integrator Using the Control Center* provides more information about the default message flows, and guidelines for using them.

A collection of sample business scenarios is available by downloading the MQSeries SupportPac IC03. The Internet URL is given in "MQSeries information available on the Internet" on page 184.

Adding or enhancing message processing nodes

MQSeries Integrator provides an external interface that allows you to add new capabilities to the broker by implementing new node types. The interface comprises a set of calls implemented in the C language. These calls are of two kinds:

- Calls that the broker makes to the node, for example to initialize the node.
- Calls that the node makes to the broker, for example, to inquire about the content of the message being processed.

Examples of additional node types might include:

- A timer node, that re-invokes itself periodically at a set timer interval, to perform a series of actions before passing the message on to the next node in the message flow.
- Reading one or more records from a specified data file: this node type might be used in conjunction with a timer node to provide a mechanism for performing batch processing at predetermined intervals, or times of day.
- Raising events that get displayed in a systems management console.

You'll find more details about this facility in "Chapter 11. Enhancing your broker domain" on page 139. The implementation details of the system programming interface are given in the *MQSeries Integrator Programming Guide*.

Solving message flow problems with the Debugger

The Control Center provides facilities that help you to validate that a message flow is performing the desired actions under all conditions, and determine the cause of unexpected processing within a message flow. As well as user tracing and trace nodes, you can use the Debugger, which is presented as an alternative screen under the Message Flows view of the Control Center.

The Debugger lets you define break points within a message flow. When a break point is encountered during message processing, control is returned to you, and you can then inspect and modify the message contents. This helps you to analyze and solve unexpected situations with message flows, such as:

- Wrongly connected nodes, such as outputs connected to incorrect inputs
- Filter nodes with incorrect conditions
- Compute nodes with incorrect logic
- Database nodes making incorrect entries into their target databases
- Incorrect messages generated by applications, or having contents that the message flow does not expect
- Feedback loops that are never exited
- User programmed plug-ins that contain errors, or are not reentrant

An example use of the Debugger is to track a single message through a message flow one step at a time. Another example is to put a breakpoint on a filter output, triggered by an erroneous message; there may be hundreds of messages generated by the input application before the bad message occurs and triggers the breakpoint.

Full details of the Debugger are provided in *MQSeries Integrator Using the Control Center*.

Chapter 5. Messages

Data and information is generated and distributed through your broker domain in the form of messages. This chapter describes the messages that MQSeries Integrator supports, and how they are interpreted by the message flows.

- “Predefined and self-defining messages”
- “Message parsers” on page 59
- “Using message templates and messages” on page 61
- “Creating additional parsers” on page 62

Predefined and self-defining messages

The format and content of each message has to be known by the process that is constructing or examining it. In MQSeries Integrator, messages are always in one of two broad categories:

- Predefined. The content of a predefined message is described by the message template.
- Self-defining. The content of a self-defining message is described by the message itself.

Predefined messages

A message can be considered in a couple of ways:

- It has a logical structure. This defines the contents of the message using a tree structure that identifies each field and its relation to other fields. For example, a message might contain three fields, in the following order:

```
AccountNumber  
AccountName  
AccountBalance
```

The applications sending and receiving messages like this understand this format, and the type of each field. For example, they might use a C structure that shows AccountNumber is an eight byte character field, AccountName is a 20 byte character field, and AccountBalance is an eight byte character field.

- It has a physical structure, also known as a wire format. Using the above example, the wire format will be a string of bytes that would look something like:

```
01234567BILLbWILLIAMSONbbbb00008907
```

where the character `b` represents a blank space

The Control Center provides the main Graphical User Interface (GUI) for message definition and management. Messages defined through the Control Center are stored in the message repository, which is created and maintained in your database, to hold all message templates in the broker domain. The message repository is managed by a component within the Configuration Manager known as the Message Repository Manager (MRM).

Each field in each message must be specified to the Control Center, using the default set of simple types, or using compound types you have defined, also through the Control Center.

Predefined messages

Before you define a message, you must define a message set to which the message belongs. Message sets keep the individual message templates linked together, and simplify their administration and distribution.

The NEONFormatter interface is available through the Control Center, as well as directly. If you have existing MQSeries Integrator Version 1 messages, or plan to create new messages using NEON formats, or both, you can use these messages by defining message flows that include the NEONRulesEvaluation, NEONTransform, and NEONMap message processing nodes. (These three nodes supersede the NEONRules and NEONFormatter nodes.) For more details about using NEON format messages, see the *MQSeries Integrator Administration Guide*. You can also use the NEONFormatter interface to create new message formats. For more information, refer to the *NEONRules and NEONFormatter Support for MQSeries Integrator User's Guide*. Note that NEON messages can be viewed, but not edited, in the Messages view of the Control Center.

If you have messages already defined within another message repository you can do one of the following:

- Export these messages to C or COBOL format and use the Control Center to import the headers or copybooks created into your MQSeries Integrator environment.
- Provide a parser that interprets these messages. For more details, see “Creating additional parsers” on page 62.

You can then use these messages with message flows you develop to support the applications that use them. Details of how to do this are described in *MQSeries Integrator Using the Control Center*.

Message templates

A message template is made up of four values contained within the message header information:

1. The message domain. This describes the source of the message definition:

MRM identifies messages defined using the Control Center

NEONMSG

identifies messages defined using the NEONMSG user interfaces

XML identifies self-defining messages, including JMS messages

BLOB identifies messages that have no defined format

NEON

identifies messages defined using the superseded NEON interface, for backwards compatibility

2. The message set. This identifies the grouping of messages within the message domain, as you have defined it. Typically a message set contains a number of related messages that provide the definitions required for a specific business task or application suite. This is equivalent to the application group (OPT_APP_GRP) in MQSeries Integrator Version 1.
3. The message type. This identifies the logical structure of the data in the message. For example, the number and location of character strings, and their relationships.

4. The message format. This describes the wire format of the message, its physical representation in the bit-stream. This attribute is only valid for messages defined through the Control Center, not for self-defining messages, or those created using the NEONFormatter. It can have one of three values:
 - XML
 - PDF¹
 - CWFxxxx (where xxxx is an arbitrary string that you assign when you create the message set in the Control Center).

Self-defining messages

A self-defining message uses the XML standard to structure its content. If the example Account message is structured using XML, it would look something like:

```
<?xml version "1.0"?>
<AccountMessage>
  <AccountNumber>01234567</AccountNumber>
  <AccountName>BILL WILLIAMSON</AccountName>
  <AccountBalance>00008907</AccountBalance>
</AccountMessage>
```

Self-defining messages can be used in any message flow on any broker. The primitive nodes provided by MQSeries Integrator support this type, except the superseded NEONRules and NEONFormatter nodes, the MQeInput and MQeOutput nodes, and the SCADAInput and SCADAOutput nodes. You do not have to define these messages using the Control Center to enable the message flows to interpret them. However, you can define them if you want a visualization of the message structure to facilitate manipulation of the messages in any of the nodes in your message flows. You do not need to assign these definitions to brokers, nor deploy them in your topology.

XML support

XML is an open messaging standard, providing a cross-platform portable mechanism for exchanging data. XML refers to a family of specifications based on a tagged message format for *metadata*. The tag language has been developed from older markup standards including GML and SGML.

XML definitions for specific business objects (for example, messages used by EDI or financial applications) are grouped using “schemas” or “*document type definitions*” (DTDs).

The XML standard is fast-growing, and is being adapted to and supported by increasing numbers of products. MQSeries Integrator’s ability to support it is therefore critical in providing comprehensive business integration. For up-to-date information about XML, and further references, see the IBM Web site

<http://www.ibm.com/developer/xml>

MQSeries messaging products support XML, and can send and receive XML structured messages. MQSeries Integrator Version 2 extends this support:

- You can use the Control Center for XML message definition. You do not have to define these messages, but if you include message processing nodes in your message flows that manipulate the message content, you will need the definition

1. The PDF format referenced here stands for “portable data format”, the native format of the GOLD messaging standard used in financial messaging applications. It is specific to the Control Center and the message repository, and has no relation to Adobe’s Portable Document Format, also known as PDF.

Self-defining messages

when you set the node properties (for example, using ESQL to change fields in a Compute node). You do not need to assign the message definitions to the brokers that will use them.

- XML is used to compose status messages that can be monitored by external systems management agents.
- Messages can be transformed from XML to any message format defined by the Control Center (XML and non-XML) by the broker.
- Message filtering, routing, and processing can be based on XML structured messages without any dictionary definition of the message.
- You can generate XML DTDs from the message repository.

Note: JMS messages are also identified by the XML domain.

How messages are processed in a message flow

The characteristics of MQSeries messages are identified by the input node of a message flow in the following ways:

- If the header immediately preceding the message body is recognized, it is examined to check for a supported value in the format field.
- If the message has an *MQRFH* or *MQRFH2* architected header, the input node checks values in that message header.

See the *MQSeries Integrator Programming Guide* for more details about the content and use of these headers.

- If the message does not have an *MQRFH* or *MQRFH2* header, the input node uses the default message template, defined as a property of the input node, to determine how the message must be parsed.

The *SCADAInput* node creates messages with *MQRFH2* headers from the messages received by the listener on the TCP/IP port.

The *MQeInput* node, used in a publish/subscribe message flow, also creates messages with *MQRFH2* headers from the messages received by the listener on the TCP/IP port.

In non publish/subscribe message flows, the *MQeInput* node action depends on the type of message received by the listener on the TCP/IP port:

- For *MQeMsgObject* messages, the *MQeInput* node creates MQSeries messages with an *MQRFH2* header, and the whole dumped-data representation of the message is copied into the message payload.
- For *MQeMbMsgObject* (message broker) messages, the *MQeInput* node creates MQSeries messages with an *MQRFH2* header. The fields defined in the *MQeMbMsgObject* class are available inside the broker; all other fields are discarded.

To get full integration across your enterprise, you probably need a variety of message templates. For example:

- Internally defined messages. You can define your own standards for messages between newly developed pieces of software. For example, messages of this type can contain XML.
- Legacy message formats. These are determined by the legacy applications themselves. They include, for example, COBOL record structures used for interacting with CICS or IMS applications. Other examples are 3270 data-streams and other forms of screen map.

- Inter-enterprise message sets (for example, EDI).
- Java message formats defined by JMS.

Your application programmers create and receive messages based on the message type, the environment the programmer is working in, and the language that the programmer is using. For example, a COBOL programmer manipulates a message as a COBOL data structure, a Notes™ programmer views it as a Notes document.

Exporting and importing MRM message sets

You can export the MRM message sets you have created in the message repository using a command provided by MQSeries Integrator. This command, **mqsimpexpmsgset**, works on whole message sets (that is, you cannot export an individual message within the message set) and generates an XML file that defines the message set. The command interacts directly with the message repository to generate the XML file.

You can reuse the exported file to populate another message repository in another broker domain, by importing it using the same MQSeries Integrator command.

The Control Center does not provide import or export functions for message sets, you must use the command. See the *MQSeries Integrator Administration Guide* for further information.

Message parsers

MQSeries Integrator can handle any message template for which a suitable parser is available. The parsers interact with the message templates stored in message dictionaries. You can extend the range of messages supported by creating your own message parsers. MQSeries Integrator provides an external interface to enable you to do this.

Default message parsers

A number of parsers are included with MQSeries Integrator. These can be considered in two broad groups, depending on the source of the message definitions.

1. Messages managed using the Control Center

When you use the Control Center to define new messages and message sets, the Control Center and Configuration Manager accept, check, and maintain the definition of these messages in the broker domain's message repository. When a message set is assigned to a broker, the information passed to the broker allows it to determine correct use of them (and therefore correct message manipulation) by the message flows.

- Record-oriented data structures

If you want to communicate with existing applications which generate messages by overlaying a data structure (typically COBOL or C) on an array of bytes, you need a definition for these that can be interpreted by the parsers.

You can achieve this by:

- Creating or purchasing a plug-in to construct and parse the messages. See “Creating additional parsers” on page 62 for further information about plug-in facilities for message parsers.

Message parsers

- Using the Control Center to import data structures created using any other method. See *MQSeries Integrator Using the Control Center* for full details of import options.
 - XML messages
You can also define messages to the message repository with a wire format of XML. Their format is specific to the Control Center, and the DTD for these messages is controlled by the message repository manager.
2. All other messages
- Messages not managed by the Control Center and Configuration Manager are also supported. These include:
- Generic XML
The broker performs simple content-based routing and manipulation on any well-formed XML message. These messages are treated as self-defining, so no schema is required. The message content can be manipulated, but no validation of the resulting message content is possible.
 - Message formats defined in the NEON dictionary
If you already have messages defined using the NEON tools, you can continue to use these formats: MQSeries Integrator correctly parses these formats. You can add new formats to your existing ones by using the NEONFormatter interface.

Any other message, for which no parser can be identified (either because the format field in the immediately preceding header is not set, or is set to an unknown value), is handled as a blob. That is, the remaining body of the message is passed through the message flow intact, and the content is left untouched.

A message flow that receives a blob message therefore can't perform content-based routing, message manipulation or message transformation. However, the message can be stored in a database, be routed according to topic, or have headers added or removed.

Parsers are also provided in the product for the following headers:

- MQCFH (the programmable command format (PCF) headers).
- MQCIH (the CICS bridge header).
- MQDLH (the DLQ header).
- MQIIH (the IMS bridge header).
- MQMD (message descriptor).
- MQMDE (message descriptor extension).
- MQRFH and MQRFH2 (rules and format headers).
- MQRMH (reference message header).
- MQSAPH (the SAP link header).
- MQWIH (the workload information header).
- SMQ_BMH (the SAP link bad message header).

The broker needs to deal with all messages in a general way, and therefore it does not handle the sequence of bytes directly but instead references *syntax elements* around which it navigates to deduce the structure of the message.

This implementation allows parsers to navigate the message tree structure, in any way. For example, a parser can access an element's parent, or its children or siblings. Other functions allow manipulation of the elements themselves, for example to set or query the values, to insert new elements into the tree or to remove elements from the tree.

Using message templates and messages

You must make the information about your message templates and messages available in the broker domain, where it is needed: that is, in any broker that hosts message flows that use these particular templates.

When you develop the topology of your broker domain using the Control Center, you make decisions about which message flows run on which brokers, and therefore need to decide at the same time which message templates are required by each broker.

If you define message templates using the Control Center, you specify the message sets that are required at each broker by assigning them to that broker, in the same way that you assign the message flows that are to be executed in that broker. You do this using the Control Center.

When you have made these decisions, you must update the repository with your changes, by checking in all the resources you have been working with. You must then request that these changes are propagated as required through the broker domain. This function, known as deployment, is handled by the Configuration Manager when you request the deploy using the Control Center.

Each message set is sent to the broker in the form of a message dictionary, which allows the broker to interpret the messages it receives. Each broker can manage a number of message dictionaries concurrently.

You can't modify dictionaries in the broker directly. However, you can delete and add message sets using the Control Center, and deploy the updated configuration, which achieves the same result.

If you create new message sets, or modify existing ones, you can assign and deploy these through the Control Center. Message flows can access new message dictionaries when the broker has implemented these changes.

If you define message templates using the NEONFormatter, you must make these available to the broker by setting values in the configuration file located by the `NN_CONFIG_FILE_PATH` environment variable. If you update these message templates, you must force the NEON nodes in the message flows to re-access the database by stopping and restarting the broker.

See *MQSeries Integrator Using the Control Center* for details of creating messages and assigning message sets to brokers.

Client access to messages

Client applications also need access to message definitions to be able to construct messages they send, and interpret messages they receive.

- If the message formats in the message repository have been imported from C or COBOL structures using the Control Center, your applications can continue to use the same C and COBOL data structures that were imported to create the message dictionary (that will be used by the brokers).
- If the messages are defined to the NEONFormatter, you must ensure the clients have access to the database in which the formats are stored (a local or remote connection is valid).

Using messages

- If the messages are self-defining XML, the client applications must construct valid messages using structures that will be understood by the recipients of the message.

Creating additional parsers

You can create additional parsers if you need to process messages (bit-streams) which for one reason or another don't fit into the categories of messages supported by the default parsers.

MQSeries Integrator provides a system programming interface, in the C language, that allows you to construct a parser to work with message processing nodes.

You can use your new parsers with existing message processing nodes (that is, those provided by MQSeries Integrator) and with your own additional plug-in message processing nodes.

You'll find further information about using this interface in "Chapter 11. Enhancing your broker domain" on page 139. Implementation details of the programming interface are in the *MQSeries Integrator Programming Guide*.

Part 3. Application planning

This part provides the information your application architects need to understand the environment that MQSeries Integrator provides for applications.

It explains the concepts introduced in “Chapter 2. MQSeries Integrator overview and concepts” on page 9, and gives more details about the implications of using the various functions of the product.

It contains the following chapters:

- “Chapter 6. Application design” on page 65
- “Chapter 7. Designing publish/subscribe applications” on page 75

The information here is an introduction to the detail in the *MQSeries Integrator Programming Guide*.

Chapter 6. Application design

This chapter introduces the main aspects of application design that you need to consider for your particular environment and applications.

This chapter covers:

- “Communication models”
- “Application programming” on page 66
- “Reusing existing applications” on page 68
- “Writing new applications” on page 69
- “MQSeries queues” on page 70
- “Message order” on page 70
- “Transaction support” on page 71
- “Security” on page 73
- “Summary” on page 73

If you are writing publish/subscribe applications, refer to “Chapter 7. Designing publish/subscribe applications” on page 75 for additional information.

Communication models

MQSeries Integrator supports two general application communication models; point-to-point and publish/subscribe. These two models, introduced in “Chapter 2. MQSeries Integrator overview and concepts” on page 9, are explored in more detail in this chapter and the next.

In point-to-point, one application sends messages to a queue associated with an input node, such as **MQInput**, of a message flow. After processing by the nodes in the message flow, the resultant message is sent directly to the receiving application’s queue by an output node, such as **MQOutput**.

In publish/subscribe, one application (the publisher) sends messages to a queue associated with an input node of a message flow that contains a **Publication** node. Another application (a subscriber) can send a subscription request to the broker, which then sends relevant publication messages to the subscriber’s queue. However, with SCADA applications the source of the message for publication is the port that the **SCADAInput** node is listening to.

A single application can also mix the two styles, if appropriate. In this case the message flow contains at least one output node and at least one publication node (in addition to one or more input nodes).

The retail scenario introduced in “Chapter 3. MQSeries Integrator: a business scenario” on page 33 is implemented using the two styles. This scenario, and others, are available by downloading the MQSeries SupportPac IC03. The Internet URL is given in “MQSeries information available on the Internet” on page 184.

This broad application support enables you to exploit your existing MQSeries, MQSeries Everyplace, and SCADA clients and applications, and to develop new applications to take advantage of the more advanced features of MQSeries Integrator. Both existing and new applications work together before and after a broker is introduced into the network.

Communication models

Point-to-point communications

Point-to-point applications exchange information with known partners. Each application is aware of the identity of the one or more applications with which it is communicating. In some cases, messages are sent from one application to another, but no response is required. These are known as *send and forget* messages or *datagrams*. In other cases, data exchange involves pairs of messages sent as requests and replies. This is called *request/response* messaging.

Your existing applications written using the point-to-point model can run unchanged in an MQSeries Integrator environment. However, you should check “Reusing existing applications” on page 68 for more detailed guidance.

You can enhance and extend your existing application function by using the facilities of the broker to include additional partners. For example, an application that handles similar data but in a different format can now participate, because the original message can be transformed by the broker into the expected format, without the sending or receiving application changing.

If you identify a message that needs additional application processing, you can create another copy of the message in the message flow, and send it to a new application developed to provide that processing. The original applications are unaware of the new action on the message and continue to work unchanged.

Publish/subscribe communications

Some applications are not tied to particular partners. They deal with data and have no specific requirements as to who is receiving that information, or where the message comes from. The publish/subscribe model allows data to be made available at any time, to whoever is interested at that time, without the sender or receiver being aware of the other.

Messages published by any one publisher can be received by any number of subscribers. Subscribers might also receive messages, on the same or different topics, from any number of publishers.

Your existing applications written using MQSeries Publish/Subscribe can run unchanged in an MQSeries Integrator environment. However, you should check “Reusing existing applications” on page 68 for more detailed guidance.

Application programming

MQSeries Integrator does not provide any new application programming interfaces. Applications can be written to the existing *Message Queue Interface* (MQI), Java Message Service (JMS), and *Application Messaging Interface* (AMI).

If you need to write SCADA applications, you should refer to the *SCADA device protocol* appendix in the *MQSeries Integrator Programming Guide*, and for MQSeries Everyplace applications refer to the MQSeries Everyplace appendix in the same book.

The MQI provides a small number of calls that allow an application to interact with other applications across an MQSeries network of queue managers. The calls support a large range of parameters that allow a rich choice of processing options for each and every message.

The JMS is a Java Messaging API developed by IBM and other messaging vendors, in partnership with Sun Microsystems. It provides a common API to access different enterprise messaging systems, such as MQSeries, through the Java programming language. Two messaging models are provided: point-to-point and publish/subscribe.

The AMI is designed to simplify the application programmer's task, by centralizing the selection of optional parameters outside the application program. It also provides support for the more advanced functions available from the message broker. The AMI is designed for general messaging applications whether a broker is involved or not.

The principal functions of the AMI are administrator-defined packets of options known as policies and services. An application specifies a service to determine the underlying messaging support required, and associates a policy with sending or receiving a message to control attributes for message processing, such as priority.

Client applications using the MQI can run on any supported MQSeries operating system, and therefore any limitations as to language or function are defined by the relevant product for that operating system.

Client applications using the JMS interface are written in the Java programming language, and are therefore restricted to the levels of JVM that are supported on the operating system in question. For further information, see the *MQSeries Using Java* book, or visit the MQSeries Web site (identified in "MQSeries information available on the Internet" on page 184).

Client applications using the AMI are restricted to the operating systems and programming languages supported by this interface. Check the current level of the *MQSeries Application Messaging Interface* book for details, or visit the MQSeries Web site (identified in "MQSeries information available on the Internet" on page 184).

Message headers

MQSeries Integrator supports applications that use different headers.

Messages begin with an MQSeries Message Descriptor, or MQMD. Defined by the MQSeries products, this precedes user or application data in every message. The MQMD contains basic control information that must travel with the message, such as:

- The message identifier
- The destination of the reply, if one is to be sent
- Reply and report options (for example, confirm on delivery report)
- The format of any following data in the message

When a message is used in an MQSeries Integrator system, it usually (but not necessarily) has one or more additional headers. The header following the MQMD is always identified in the format field within the MQMD, and itself contains another format field to identify what follows.

The additional headers can include:

- MQRFH. The Rules and Formatting header is used by MQSeries Integrator Version 1 applications and MQSeries Publish/Subscribe.
- MQRFH2. The updated version of MQRFH allows Unicode strings to be transported without translation, and it can carry numeric datatypes. The MQRFH2 header carries a description of the message contents, so that MQSeries

Application programming

Integrator can select the correct message parser when content-based processing is carried out on the message. In addition, this header contains publish/subscribe command messages. Messages created by the SCADAInput node always have MQRFH2 headers.

You are recommended to use the MQRFH2 header in all new applications written for the MQSeries Integrator environment. If you do so, you must include the MQRFH2 header immediately before the body of the message.

Reusing existing applications

Existing MQSeries applications are supported unchanged by MQSeries Integrator. The broker can be added into an existing MQSeries network, and therefore into the path taken by a message, to provide additional function, such as warehousing of message traffic. The applications that send and receive the message are not aware that the broker is now intercepting that message.

MQSeries Integrator Version 2:

- Accepts messages without MQRFH or MQRFH2 headers. If content-based processing of the message is to be carried out in a message flow, you need to describe the message contents in the properties of the input node (see *MQSeries Integrator Using the Control Center*).
- Provides the MQSeries Integrator Version 1 function via the NEONRulesEvaluation, NEONTransform, and NEONMap nodes (and the superseded NEONRules and NEONFormatter nodes) as compatible message processing nodes, ready to be included as required in any message flow defined to the broker. The graphical user interface tools for creation and management of the rules and formats used by these nodes, and the Visual Tester, are also supplied with MQSeries Integrator Version 2.

For more details of how to incorporate these nodes into message flows, refer to *MQSeries Integrator Using the Control Center*.

- Accepts publish/subscribe messages from MQSeries Publish/Subscribe using the MQRFH header, in addition to the more comprehensive MQRFH2 header used in MQSeries Integrator Version 2.

For details of the MQRFH header, see the *MQSeries Publish/Subscribe User's Guide*, and for the MQRFH2 header, see the *MQSeries Integrator Programming Guide*.

Send and forget

For simple one-way message flows, additional function is easily achieved. You can design and deploy a message flow that implements the desired functions within the broker, and use queue aliasing to redirect the original message stream to the new input queue for this new message flow.

Define the nodes that provide the new processing you require, then define the output node of the message flow to represent the original queue. This will result in a message being processed by the new message flow, and being written to the queue read by the receiving application after processing is complete.

Request/reply

MQSeries Integrator also supports request/reply applications. You can set up a message flow to process the request in whatever way you need. Somewhere within that message flow (in a database, for example), record the parameters you need from the sending application's message descriptor (MQMD). You will need the

ReplyTo queue and queue manager, and perhaps other fields such as the report options. You might find it necessary or most convenient to save the complete MQMD.

You must then update the original MQMD with the required new values. For example, insert a new *ReplyTo* queue and queue manager to represent the input node of the message flow you create to handle the responses.

When the reply is processed by this second message flow, the processing must include retrieval of the original MQMD values (such as the *ReplyTo* queue identifier recorded by the first message flow) or the entire saved MQMD to ensure the message is delivered as expected.

This technique works regardless of the number of replies expected to any request message. You have to provide the extra logic and processing within the message flows created to handle both request and reply, but this leaves the applications themselves unchanged. This can be particularly valuable if you do not own these applications, but are interacting with other departments or businesses.

If the reply message does not have to be processed in any way, you do not need to create a second message flow, and the first message flow (processing the request message) can simply propagate the original *ReplyTo* field in the message header intact.

If you have a client/server suite of applications, where multiple clients expect responses from a single server, you might find the applications need modifying to use additional techniques to match requests and replies (such as a *CorrelId*) and ensure the replies are correctly delivered.

Publish/subscribe

Publish/subscribe client applications written to the MQSeries Publish/Subscribe interface execute unchanged. You need to create and deploy a message flow that contains a **Publication** node, define the publication queue to the broker's queue manager, and specify it in the input node of the message flow.

MQSeries Integrator uses the same broker control queue as MQSeries Publish/Subscribe (SYSTEM.BROKER.CONTROL.QUEUE), therefore existing subscriber applications do not have to be changed.

For other migration considerations, check "Appendix A. Planning for migration and integration" on page 143.

Writing new applications

You can write applications that use more of the function of the MQSeries Integrator broker by adding the MQRFH2 header to some or all of your messages.

The MQRFH2 header (described in detail in the *MQSeries Integrator Programming Guide*) is used to define the message set and format for the body of the message, and to define publish/subscribe command messages. This header is extensible, allowing client applications to define fields that can be accessed and processed by customized message processing nodes. This header must immediately precede the body of the message.

New applications

If you are writing a request/reply application, you can store the *ReplyTo* queue and queue manager for the reply message (and any other options you require) in a folder contained in the MQRFH2 header, instead of using a database node as described in “Reusing existing applications” on page 68.

In some applications, it might be convenient to carry the application data in folders in the MQRFH2 header. You can create your own folders within the header. The MQRFH2 header, and suggested naming conventions for your own folders, are described in the *MQSeries Integrator Programming Guide*.

If you are writing new client applications, use the MQRFH2 header. This enables your applications to exploit all the function contained in MQSeries Integrator.

MQSeries queues

MQSeries Integrator uses a number of dedicated queues, defined by each broker, the Configuration Manager and the User Name Server, for specific functions. You can find a full list of these queues, and their purpose, in the *MQSeries Integrator Installation Guide*.

Application designers need to be aware of the system-defined queues with which they need to interact: for example, the broker control queue for publish/subscribe (SYSTEM.BROKER.CONTROL.QUEUE).

As you develop new applications, or integrate existing applications into your broker environment, you must agree on a naming convention for the queues you use for message exchange. (Input and output queues for point-to-point; input and subscriber queues for publish/subscribe.) Make sure these names do not start with the characters SYSTEM.BROKER, to avoid conflict with the system defined queues.

A subscribing application can specify a temporary dynamic queue as its subscriber queue (the queue to which publications should be sent). In this case, the broker will automatically de-register the subscription when the queue is deleted.

Message order

If message ordering is important, you can use the techniques recommended for all MQI and AMI users. See the *MQSeries Application Programming Guide* for programs written to the MQI, and *MQSeries Application Messaging Interface* for programs written to the AMI.

Note that MQSeries Everyplace and SCADA applications use a different method of message ordering — refer to the *MQSeries Integrator Programming Guide* for details.

If you have set the ‘Additional Instances’ property of a message flow to define more than one instance of that message flow, you can use the ‘Order Mode’ property of each input node within that message flow to influence the order of message processing by that node:

- If you set ‘Order Mode’ to ‘By User ID’, the node will ensure that messages from a specific user (identified by *UserIdentifier* field in the MQMD) are processed in guaranteed order. A second message from one user will not be processed by an instance of the message flow if a previous message from this user is currently being processed by another instance of the message flow.
- If you set ‘Order Mode’ to ‘By Queue Order’, the node will process a single message at a time to preserve the order in which the messages are read from the

queue. Therefore, this node behaves as though the 'Additional Instances' property of the message flow is set to zero.

Publish/subscribe

Additional considerations apply to publish/subscribe applications. For any given topic, messages are published by brokers in the same order as they are received from publishers (subject to reordering based on message priority). This normally means that each subscriber receives messages from a particular broker, on a particular topic, from a particular publisher, in the order that they are published by that publisher.

However, in common with all messages using the MQSeries transport layer, it is possible for messages, occasionally, to be delivered out of order. This could happen, for example, if a link in the network fails and subsequent messages are routed via another link.

If you need to ensure the order in which messages are received, you can use either the *SeqNum* (sequence number) or *PubTime* (publish time stamp) parameter on the **Publish** command for each published message, to calculate the order of publishing. Check the *MQSeries Integrator Programming Guide* for details of how to implement these message ordering techniques.

Transaction support

Message flows hosted by brokers might provide vital processing and data manipulation that must have full transactional integrity. That is, the message flow must complete all processing successfully, or must complete none. Any part of the processing that completed successfully (for example, the reading of the input message from the input queue) must be rolled back if there are problems that prevent later processing from completing successfully.

If the message flow processing includes interaction with an external database, the transaction can be coordinated using XA technology to assure all participants update or return to a consistent state. This external coordination support is provided by the underlying MQSeries facilities and the ODBC support provided for the database. This level of XA support is only available if the database you are using is DB2 (DB2 or Oracle on MQSeries Integrator for Sun Solaris).

MQSeries Integrator provides the required level of transactional integrity in several ways.

- You can specify that a message flow is to be *fully globally coordinated*, which means that MQSeries itself will be used as an XA Transaction Manager to coordinate the transaction associated with the message flow. The reading and writing of MQSeries messages and all interactions with capable external databases are coordinated in a single unit of work (UOW).

Fully globally coordinated transactions are only possible if the external databases are DB2 (DB2 or Oracle on MQSeries Integrator for Sun Solaris).

You must configure your external databases and MQSeries to enable this support. All actions in the message flow therefore either complete successfully, or are rolled back to the point where the original input message is restored on the input queue (or in the DB2 table for SCADA message flows).

This feature is controlled using the 'Coordinated Transaction' property of the message flow: the default is for the transaction *not* to be globally coordinated.

Transaction support

- A message flow that is not fully globally coordinated is said to be *fully broker coordinated* by default. The reading and writing of MQSeries messages and interactions with external databases are *not* coordinated within a single unit of work (UOW). However, the message flow ensures that all database transactions are committed automatically at the completion of processing a message through that flow.
- A message flow can also be *partially broker coordinated*. This means that some processing nodes will commit their operation immediately, instead of waiting until message flow completion as in a fully broker coordinated message flow. You can specify property values on the nodes that interact with databases to allow their processing to be committed immediately.
- In a fully globally coordinated or partially broker coordinated message flow, all messages subsequently sent by any output node in the same instance of the message flow are put under syncpoint, unless you set the output node properties to explicitly override this. If you do this, then the message flow is also be said to be partially broker coordinated.

You can mix these different transaction types by using different settings in multiple nodes with one message flow. The *MQSeries Integrator Administration Guide* has details of how to work with transactions in this way.

Message persistence

MQSeries messaging products provide an additional level of support for message integrity. This is *message persistence*, which defines the longevity of the message in the system. Non-persistent messages are lost in the event of system or queue manager failure. Persistent messages are always recovered if a failure occurs. SCADA applications use appropriate settings of *quality of service* instead of message persistence.

Message persistence is controlled by these factors:

- The option specified by the application putting the message to the queue (using the MQI or AMI calls)
- The default message persistence of the input queue, or the quality of service of the SCADA message
- The action taken by a message processing node in the message flow
- The option specified by the output node's persistence property
- The message persistence requested by the subscriber

When a message is read from an input queue by the input node, the default action is to use the persistence defined in the MQSeries message header (MQMD), that has been set either by the application creating the message, or by the default persistence of the input queue. The message retains this persistence throughout the message flow, unless it is changed in a subsequent message processing node.

You can override the persistence value of each message when the message flow terminates at an output node. This node has a property that allows you to specify the message persistence of each message when it is put to the output queue, either as the required value, or as a default value. If you specify default, the message takes the persistence value defined for the (one or more) queues to which the messages are written.

If a subscriber has requested persistent message delivery, and is authorized to do so by explicit or implicit (inherited) ACL, the message is delivered persistently

regardless of its existing persistence property. Also, if the user has requested non-persistent message delivery, the message is delivered non-persistent regardless of its existing persistence property.

Security

Access and authority requirements for MQSeries client applications to connect to queue managers and use MQSeries resources are unchanged by the introduction of MQSeries Integrator into your application environment.

You must therefore ensure that applications are authorized to put messages to input queues serviced by the message flow that provides the required processing, and are able to get messages from the message flow output queues.

SCADA applications have less control over access, because any device can connect using any client ID it chooses, and can therefore masquerade as an authorised ID.

For publish/subscribe applications, additional control is available to you. This is defined in “Topic-based security” on page 86.

Summary

This chapter has provided the information you require to make the following design decisions for your applications:

- What message header to use (MQRFH2 for new applications, MQRFH or no header for existing applications)
- What queues or ports to use for sending and receiving messages (they need to be set up in the message flow nodes)
- What programming interface to use (MQI, JMS, or AMI)
- What communication model to use (point-to-point, publish/subscribe, or both)
- What other features you need (transactional processing, message persistence, message ordering)

For more information about publish/subscribe applications, see “Chapter 7. Designing publish/subscribe applications” on page 75.

For information about writing the applications, having made the design decisions, see the *MQSeries Integrator Programming Guide*.

Summary

Chapter 7. Designing publish/subscribe applications

If you are using the publish/subscribe facilities of MQSeries Integrator, you need to consider the following aspects of application design in addition to those discussed in “Chapter 6. Application design” on page 65. This chapter covers:

- “How publish/subscribe applications interact with a broker”
- “Publications” on page 76
- “Subscriptions” on page 78
- “Topics” on page 82
- “Broker networks” on page 85
- “Topic-based security” on page 86
- “Summary” on page 92

How publish/subscribe applications interact with a broker

The simplest model of publish/subscribe communications involves a single broker, one application that publishes messages, and one application that subscribes to messages.

The publisher generates a message it wants to publish on a topic. The behavior of the publisher and the ways in which it can publish a message are discussed in “Publications” on page 76. “Topics” on page 82 describes topics and explains how they can be constructed.

A message flow running in the broker retrieves the publication from its input queue (read by the input node), performs any processing that is defined for publications received in that message flow, and passes the message to a publication node for distribution to a subscriber. With SCADA applications, the SCADAInput node receives the message from the SCADA port before processing by the message flow.

The publication node only knows about, and can therefore only provide messages to, an application that has registered as a subscriber. When the application registers as a subscriber, it must specify a queue on which it wants to receive messages, and a definition that restricts the messages it wants to receive. This definition is based on a combination of the topic of the message, or specific content within the message, or both. This is discussed in detail in “Subscriptions” on page 78. SCADA subscribers need to specify the topic or topics that they are interested in.

Figure 15 on page 76 shows the messages that pass between a broker and a publisher, and the broker and a subscriber.

Publications

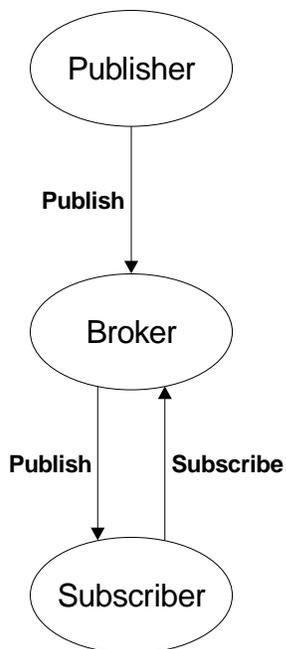


Figure 15. Publish/subscribe with a single broker

Publications

When designing a publish/subscribe system, you need to consider if publications should be retained by the broker after they have been sent to subscribers. You can also choose to publish to subscribers at your local broker only, instead of allowing publications to be propagated throughout the network of brokers. These options are described in the following sections.

Retained publications

By default, a broker discards a publication when it has sent that publication to all interested subscribers. However, a publisher can specify that it wants the broker to keep a copy of a publication, which is then called a *retained publication*. The copy can be sent by the broker to subsequent subscribers who register an interest in the topic. This means that new subscribers don't have to wait for information to be published again before they receive it.

For example, a subscriber registering a subscription to a stock price would receive the current price straightaway, without waiting for the stock price to change (and hence be republished).

The broker retains only one publication for each topic and subscription point, so the old publication is deleted when a new one arrives. See "Subscription points" on page 79 for further details about subscription points.

State and event information

Information being published can be categorized as *state information* or *event information*. This section explains these concepts, and helps you to understand why you might want to use retained publications to provide these two categories of information.

State information is information about the current *state* of something, such as the price of stock or the current score in a soccer match. When something happens (for

example, the stock price falls, or the soccer score changes), the previous state information is no longer required because it is superseded by the new information.

A subscriber usually wants to receive the current version of the state information when it starts up, and to be sent new information whenever the state changes.

Event information is information about individual *events* that occur, such as a trade in some stock or the scoring of a particular goal. Each of these events is independent of the others.

A subscriber usually wants to receive information about events when they happen.

Using retained publications

When deciding whether to use retained publications, you must consider several factors:

- What sort of information will your publications contain (state or event information)?

Event information does not usually have to be retained, but state information is often retained. However, if all the subscriptions to a topic are in place *before* any publications are made on that topic (and no new ones expected), there is no need to retain publications even for state information, because they are delivered to all the subscribers as soon as they are published.

Publications of state information might also not need to be retained if they are very frequent (for example, every second). With this frequency of publishing, any new subscriber (or a subscriber recovering from a failure) receives the current state almost immediately after it subscribes.

- Do you want to receive publications on request only?

If you use retained publications, subscribers can register using the 'Publish on Request Only' option. This means that the broker will not send any publications to that subscriber until the subscriber requests an update. The broker then sends to the subscriber the current retained publication that matches the subscription.

- Can retained publications be mixed with non-retained publications on the same topic?

This is not recommended. If you have a retained publication, and then publish a non-retained publication on the same topic, the existing retained publication is still retained (it will not be updated by the non-retained publication). If you have a subscriber that has registered with the 'Publish on Request Only' option, it will not be able to access any non-retained publications (the broker sends only the current retained publication in response to a request for an update).

- Can you have more than one application publishing retained publications on the same topic?

You are recommended not to have two or more applications publishing retained publications on the same topic. If you do and the timing is close to simultaneous, it is indeterminate which publication is retained. If the publishers use different brokers, it is possible that different retained publications for the same topic could be held at each broker.

- How will the subscriber application recover from failure?

If the publisher does not use retained publications, the subscriber application might need to store its current state locally. If the publisher does use retained publications, the subscriber can request an update to refresh its state information after a restart.

The broker continues to send publications to a registered subscriber even if that subscriber is not running. This could lead to a buildup of messages on the

Publications

subscriber queue, which can be avoided if the subscriber registers with the 'Publish on Request Only' option. The subscriber must then refresh its state periodically by requesting an update or by using a temporary dynamic queue.

- What are the performance implications of retaining publications?

The broker needs to store retained publications in a database, which reduces throughput. If the publications are very large, a considerable amount of disk space will be needed to store the retained publication of each topic. In a multi-broker environment, retained publications are stored by all other connected brokers that have a matching subscription.

The sample verification applications that are shipped with MQSeries Integrator include the Soccer Results service. This sample uses retained publications to record the latest score in each soccer match it is monitoring. The sample code illustrates the programming required to support this option.

Local and global publications

Publications can be categorized as either global or local.

Global publication

A global publication is distributed throughout the broker domain to all connected brokers. Each broker delivers a global publication to all its neighbors that have a subscriber registered with a subscription that matches the publication. Controls are in place to ensure these publications do not get into a loop.

It is possible to have more than one group of connected brokers within a single broker domain. A global publication can only be delivered to brokers that are interconnected, so its distribution is limited by the topology of your broker domain.

Local publication

Publishers can choose to restrict access to their publications to subscribers registered to the same broker as the publisher.

The publisher can specify the 'Local' option when it sends a publication. Local publications are not forwarded to other brokers.

Conference-type applications

In some cases, a publisher might also be a subscriber. For example, a group of applications can all subscribe to the same topic (such as "Conference"), and receive publications on this topic. Using the 'Other Subscribers Only' option ensures that each application will receive publications from the other applications, but not those that it has published itself.

Subscriptions

Subscriptions are supported by MQSeries Integrator in a dynamic fashion. The broker is unaware of the intention of the subscriber to register, and cannot know at any time about any subscribers other than those currently subscribed. Subscribers can register and de-register at any time, and as often as they choose.

Client applications (subscribers) issue subscription registration requests to their local broker when they want to receive published messages. All the information associated with the subscription is recorded by the broker in the subscription table. It can only be removed from this table when the subscriber de-registers, or when the subscription expires, or is deleted by the Control Center.

If the subscriber specifies a temporary dynamic queue as the queue to which publications should be sent, the broker will de-register the subscription automatically when the queue is deleted.

The subscribing application specifies the following information on the registration request:

- The topic or topics of the published messages in which it has an interest (see “Topics” on page 82).
If you specify the *multi-level wildcard* (“#”) by itself, all published messages with matching subscription points and content filters (if specified) are valid, including event publications. (For more information about the multi-level wildcard, see “The multi-level wildcard” on page 83.)
- The subscription point (see “Subscription points”) from which it wants to receive publications.
This value should match the subscription point property set for at least one publication node defined in this broker (this could be the default subscription point). If it does not match, the subscriber will not receive any publications (unless a publication node is defined subsequently with this subscription point name). For SCADA applications, the SCADA connection port is the implied subscription point.
- The content filter (see “Filters” on page 80) to be applied to the published message.
This information is optional: the subscriber does not have to include a content filter. If it does not, all published messages with matching subscription points and topics, if specified, are valid. Content filters cannot be used with SCADA messages.
- The identity of the queue (the *subscriber queue*) on which it wants to receive publications that match the criteria it has selected. An optional *CorrelId* can be specified (this is useful if several subscribers share the same queue). For SCADA applications, the SCADA port receives the publications without needing to be specified.

When the publication node receives a message, it checks through the subscription table to determine if there are any subscription requests that specify this particular node’s subscription point, that match the content, or topic, or both, of the message received.

For every match found, the node delivers the published message on the subscriber queue, using the optional *CorrelId* if specified (otherwise a fixed value is used). Each subscriber receives a single copy of each publication regardless of the number of matching subscriptions the client has. SCADA applications subscribe and publish via the SCADA port, and *CorrelId* is not applicable.

When the node has sent the publication to any subscribers that have a matching subscription, the publication is discarded (unless it is a retained publication).

Subscription points

A message flow used for publish/subscribe must contain at least one of the supplied input nodes, such as **MQInput**, and at least one **Publication** node. (Note that SCADA publish/subscribe applications also need the publication node, or exceptionally the **SCADAOutput** node.) A subscription point is the name by which a subscriber requests publications from a particular set of publication nodes. You

Subscriptions

can use the default subscription point, or set up specific subscription points, and you can have more than one publication node associated with a particular subscription point.

The default subscription point

If you define a publication node without specifying its subscription point property, it is associated with the default subscription point. A subscriber that registers a subscription without specifying a subscription point will receive publications from any such publication node (provided they match the topic and filter specified by the subscriber).

This applies to all message flows running in all brokers connected in the same network (unless the 'Local' option has been specified).

Using subscription points

If you have more than one publication node in a message flow, you can differentiate between them by specifying subscription points. These should have values that reflect the nature of the messages routed to each publication node.

For example, a message flow might apply a filter to a message for publication, and apply two different compute operations to the outputs of the **Filter** node before sending the resultant messages to separate publication nodes. In this case, the subscription point names for these publication nodes should reflect the operations carried out by the message flow. Other message flows could have publication nodes associated with either or both of these subscription points, if appropriate.

Alternatively, allow one publication node to have the default subscription point, and apply a meaningful name to the subscription point of each additional publication node. If more than one publication node in a message flow has the same subscription point property, subscribers might receive more than one copy of each publication, unless the conditions under which messages reach publication nodes are mutually exclusive.

Example

Suppose you have an application that publishes stock prices. The prices that are available from the first publication node in the message flow are in dollars. This publication node uses the default subscription point.

You can define a second path through the message flow that takes the price in dollars, and converts this using some defined conversion value, to produce the same message but with the stock price in pounds. These messages are published at a second publication node that has its subscription point property set to 'Pounds'.

You might have another message flow (running in the same broker, or a connected broker) that publishes stock prices in pounds on the same topic. Make sure it uses the 'Pounds' subscription point, and that any other message flows publishing their stock prices in dollars use the default subscription point.

Subscribers specifying the relevant topic (for example, 'stock') can then choose to receive the information in dollars or pounds, by using the default subscription point or the 'Pounds' subscription point when they subscribe.

Filters

When you register a subscription, you can specify a content-based filter to select publications according to their contents, in addition to specifying a topic and

subscription point. MQSeries Integrator needs to know how to parse the contents of the message correctly. This can be achieved in a number of ways:

- The message is a self-defining XML message.
- The message template is defined in the MQRFH2 header.
- If the message has an MQRFH header, the message set and type are taken from that header.
- Otherwise, the message is assumed to be as defined in the properties (domain, set, type and format) of the input node.

The filter itself is entered as an expression with ESQL syntax, for example:

```
Body.Name LIKE 'Smit%'
```

This means that the contents of a field called Name in the body of a publication message (that is, the publication data that follows the MQRFH2 header) will be extracted and compared to the string given in the expression. If the string in the message starts with the characters “Smit”, the expression evaluates to TRUE and so the publication will be sent to the subscriber.

The language used in the specification of filters for content-based routing forms a proper subset of the Filter node’ language. For more information about the syntax of filter expressions, see the *MQSeries Integrator Programming Guide*.

If you want to select publications using filters only, without specifying a topic, you can register a subscription with the required filter and a topic of “#” (all topics). You will receive publications only on those topics for which you have access authority. However, this subscription will result in all publications from all connected brokers being sent to the broker that is local to the subscriber. If you have set up a network of brokers, you are not advised to use this technique for performance reasons.

Local subscriptions

Subscribers can specify a local option on registration. If they do so, they are requesting that their subscription registration is not forwarded to other brokers, but held by the local broker. Any message published at this broker that matches the subscription is received by this subscriber, but messages published to other brokers are not normally available (unless the subscriber has also registered a global subscription with an overlapping topic and the same subscription point).

Retained publications

If retained publications are used, the subscriber can specify the following options when it registers a subscription.

Publish on request only

If the ‘Publish on Request Only’ option is used, the broker will not send publications to the subscriber until the subscriber sends a ‘Request Update’ message to the broker. The broker then sends any current retained publication that matches the subscription.

New publications only

Normally the broker will send the current retained publication that matches the subscription when a subscriber registers that subscription. If the subscriber uses the ‘New Publications Only’ option, the broker will wait until a new publication is received before sending it to the subscriber.

Subscriptions

Message persistence

You are recommended to send all subscription registration messages as persistent messages. All subscriptions are maintained persistently by the broker.

Brokers maintain the persistence of publications as set by the publisher, unless changed by options specified when the subscription is registered. These options are non-persistent, persistent, persistence as queue, or persistence as publisher (the default).

The system administrator decides which users are allowed to have publications sent persistently (see “Access control lists” on page 87).

Topics

A topic specifies a subject of common interest to producers and consumers of messages (publishers and subscribers). Almost any string of characters can act as a topic to describe the topic *category* of a message. However, there are three reserved characters, described in “Special characters in topics” on page 83.

Topics provide the key to the delivery of messages between publishers and subscribers. They provide an anonymous alternative to citing specific destination addresses. The broker attempts to match a topic on a published message with a list of clients who have subscribed to that topic. Topics can also be used to control which subscribers are authorized to receive publications.

You create the topics needed by your messages in a tree hierarchy, using the facilities of the Control Center. The tree can be defined before being used, and, if you choose, added to dynamically when new topics are created by client applications.

Thoughtful design of topic names and topic trees can save time and effort later for routine operations, including:

- Subscribing to multiple topics.
- Establishing security policies.
- Automatically reacting to messages on a specific topic, for example sending an alert to a manager’s pager.

Individual topics serve as elements (that is, nodes) in the topic tree. New elements are added as you define them through the Control Center, or are specified by applications, to create topic trees. Although it can be flat (linear), a topic tree usually builds from one or more root topics, adding other topics in levels of parent/child relationships to create a hierarchical naming structure.

The following figure illustrates a topic tree structure.

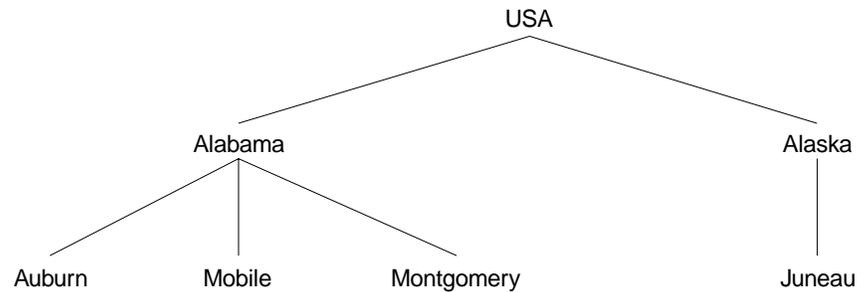


Figure 16. Example topic tree

The structure of the tree follows a format with levels of increasing granularity: “country/state/city”. Each string in the figure represents a node on the topic tree. Complete topic names aggregate nodes at one or more levels in the topic tree. Levels are separated by the “/” character (see “Special characters in topics”). Topic names fully specify the path to a specific node from the root of the tree in this format: “root/level2/level3”.

In Figure 16, the string “USA” acts as a root node, the first level of a topic name for topics in this tree. Valid topics include “USA”, “USA/Alabama” and “USA/Alabama/Montgomery”.

When you design topic names and topic trees, it is important to remember that the message broker does not interpret or attempt to derive meaning from the topic name itself. It only uses the topic name to send related messages to clients who have subscribed to that topic.

Special characters in topics

The topic of a message can contain any of the characters found in the Unicode character set. Three of these characters have a special meaning for MQSeries Integrator.

The three are the *topic level separator* “/”, the *multi-level wildcard* “#”, and the *single-level wildcard* “+”. The first of these is used to introduce structure to the topic, and can therefore be specified within the topic for that purpose. The latter two are wildcards used for subscriptions (see “Using wildcards with topics” on page 85) and cannot be used within a topic when a message is published.

Note: If you are migrating your applications from MQSeries Publish/Subscribe environment, refer to “Appendix A. Planning for migration and integration” on page 143 for further details about topics and wildcards.

The topic level separator

The topic level separator character “/” provides a hierarchical structure to the topic space. It must be used by applications to denote levels within a topic tree. The use of the topic level separator is significant when the two wildcard characters are encountered in topics specified by subscribers.

Topic hierarchy is important in administration of access control, described in “Access control lists” on page 87.

The multi-level wildcard

The multi-level wildcard character “#” is used to match any number of levels within a topic, typically an unknown number. It can be used only at the beginning

Topics

or the end of a topic (but not both). For example, you can subscribe to “USA/#”, and receive messages on topics “USA/Alabama” and “USA/Alabama/Auburn”.

The way the multi-level wildcard is implemented means it can represent zero or more levels. Therefore “USA/#” can also match the singular “USA”, where # represents zero levels. The topic level separator is meaningless in this context, because there is no level to separate.

You can only use the multi-level wildcard next to the topic level separator character unless you specify the multi-level wildcard on its own. For example, “USA#” is not valid, but “#” is.

The single-level wildcard

The single-level wildcard character “+” matches one (and only one) topic level. For example, “USA/+” matches “USA/Alabama” but not “USA/Alabama/Auburn”. Also, because the single-level wildcard matches a single level only, “USA/+” does not match “USA”.

This wildcard can be used at any level in the topic tree, and in conjunction with the multi-level wildcard. However, you can only use the single-level wildcard next to the topic level separator character unless you specify the single-level wildcard on its own. For example, “USA+” is not valid, but “+” is valid.

Note: A finer level of filtering can be provided using content filters (see “Filters” on page 80).

Topic semantics and usage

When you build an application, the topic tree design is important to the application’s communication model. The design should account for the following principles of topic name syntax and semantics:

- Topic names are **case sensitive**. For example, MQSeries Integrator recognizes “ACCOUNTS” and “Accounts” as two different topics.
- Topic names can include the space character. For example, you can define “Accounts payable” as a valid topic.
- Though not recommended, a topic level can be an empty string. For example, “a//c” is a three level topic name with an empty middle level.
- A leading “/” creates a distinct topic: “/USA” is not the same as “USA” and “/USA” will match “+/+” and “/+” but not “+”.
- For portability reasons, you should not include the null character (Unicode \x0000) in any topic.

MQSeries Integrator applies the following conditions to the construction and content of a topic tree:

- There is no limit to the levels of depth (the number of topic levels) in the tree.
- There is no limit to the length of any level name in the tree.
- There can be any number of “root” nodes (that is, any number of topic trees). These are defined below the root “”, which is the root of all root nodes. It is referred to as “topicRoot”, although there is no corresponding topic name. Applications cannot publish or subscribe to this virtual root.
- The topic trees with roots of “\$SYS” and “\$ISYS” are reserved for use by MQSeries Integrator.

If you are using topic-based security, only brokers can publish messages on these topics, and only brokers can subscribe to messages with a topic of “\$ISYS”.

regardless of the topic Access Control Lists (ACLs) defined using the Control Center. For more details about topic-based security and ACLs, see “Topic-based security” on page 86.

Using wildcards with topics

Wildcards are used only when subscribing to topics, de-registering, requesting updates, and deleting publications. Messages must always be published with a fully specified topic name.

Using wildcards in subscriptions is not difficult, but needs to be done with care. Remember that wildcards can be used at any level in the topic name string (within the restrictions already discussed). However, you are recommended to use them only at the end of a topic name. Although the single-level wildcard is accepted anywhere, the product is optimized to it being specified at the end of the string. The multi-level wildcard can only be used at the beginning or end of the string.

You should create well-formed applications that structure topics into subject trees. This allows the applications to subscribe to sub-trees by placing the multi-level wildcard “#” at the end of a topic.

You can specify more than one wildcard within a subscription, as long as their use conforms to the guidelines given. For example, “+/Alabama/#” is valid.

If you subscribe with “#”, you will receive all publications from all connected brokers. You are therefore recommended to use this type of subscription with care, to minimize the impact of workload in your broker network.

Multiple topics

It is permissible to specify more than one topic for a publication. One use of this is as follows.

Suppose an application publishes information under the topic ‘Topic 1’. The application might then be enhanced to provide additional information, which it might publish under the topic ‘Topic 1 enhanced’. If the new publications specify the original ‘Topic 1’ as well, then existing subscribers will receive both old and new publications, while subscribers who want to receive only the enhanced publications can register with ‘Topic 1 enhanced’.

Note that an application that subscribes to *both* topics will receive one copy only of each publication.

Broker networks

The interactions between a broker and its publishing and subscribing applications, described in “How publish/subscribe applications interact with a broker” on page 75, are equally valid in a broker network, in which publish/subscribe applications are interacting with any one of a number of connected brokers.

Subscriptions and published messages are propagated through the MQSeries Integrator broker domain. You can set up a network of brokers using the Control Center so that each has an explicit or implicit connection to a group of other brokers. You can have more than one group of connected brokers in the broker domain. Brokers propagate subscription registrations through each network of connected brokers, and publications are forwarded to all brokers that have matching subscriptions.

Broker networks

It doesn't matter, therefore, which broker a message is published to. Any application that has registered a subscription to a connected broker will receive publications matching that subscription.

Figure 17 illustrates a simple example of the publish and subscribe messages flowing through a network of two brokers.

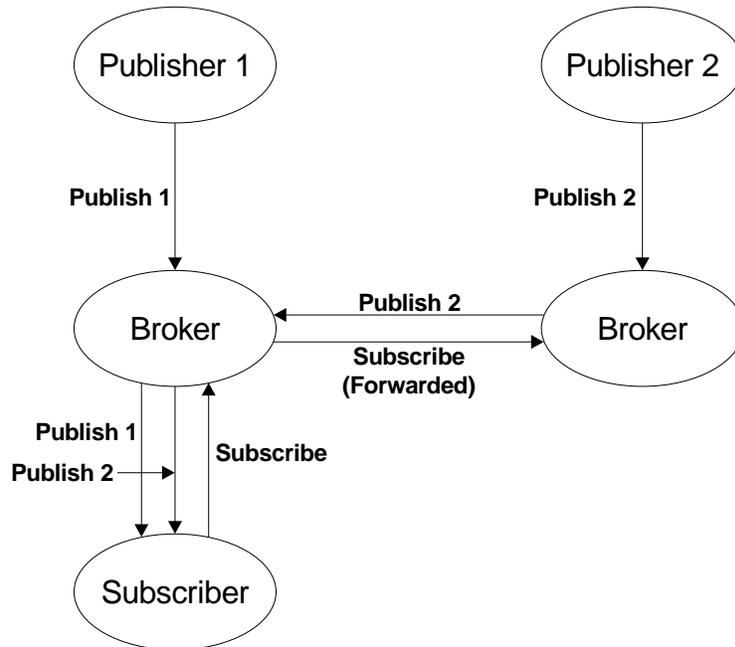


Figure 17. Publish/subscribe in a network

Each broker records subscription information from its local subscribers and information from remote subscribers forwarded by its neighbor brokers in its subscription table, which holds all the current subscription information known to that broker (for all execution groups and message flows).

Collectives

You can group your brokers in collectives. This is a way of organizing a network of brokers to get the most effective environment for publish/subscribe applications.

You can define collectives and organize your brokers using the Control Center. For more details about setting up a network with collectives, see “Supporting publish/subscribe services” on page 110.

Topic-based security

You can control access to messages on particular topics by implementing security measures governed by Access Control Lists (ACLs), which are based on the definition of principals to the underlying security control facility. A principal can be an individual user ID (for example, a logon ID), or a user group which can contain individual users.

Principals and the User Name Server

The message descriptor assigned to each message transmitted by MQSeries contains the identity of the principal that initiated the message. MQSeries sets this

identity in an operating system dependent manner, but this can be augmented at an MQSeries installation by use of standard MQSeries exits. The principal in the message descriptor is used to determine authority for the topic being published or subscribed to.

MQSeries Integrator security architecture is based on the assumption that the network is heterogeneous: although MQSeries includes a form of Windows NT domain information for client platform identification, MQSeries Integrator does not exploit this information.

The MQSeries Integrator User Name Server manages the set of principals already defined in your network, on behalf of the brokers and the Configuration Manager, for use in publish/subscribe. On Windows NT, this list of users is drawn from the domain specified on the **mqsicreateconfigmgr** command.

All brokers within the broker domain interact with the User Name Server to retrieve the total set of users and groups against which the access control lists are built and publish/subscribe requests validated.

The Configuration Manager interacts with the User Name Server and displays users/groups in ACL creation in the Topics view from the Control Center.

The User Name Server is made known to both the broker and the Configuration Manager by specifying the User Name Server queue manager on the create command for both components.

Access control lists

ACLs allow you to define, for any intersection of topic and principal, the right of that principal to publish on or subscribe to a given topic, or to request persistent delivery. You specify these definitions using the *Topics* view in the Control Center.

Access control is set explicitly on an individual topic, but can be inherited if there is no explicit ACL in place. Inheritance is from an ancestor (parent) topic, defined by the hierarchical structure of the topic tree. If none of the parent topics up to the topic root has an explicit ACL, the individual topic inherits the ACL of the topic root.

Any defined principal (user or group) known to the User Name Server can be associated with the topic in this way.

PublicGroup authorizations

In addition to the groups that you define, MQSeries Integrator provides an implicit group, "PublicGroup", to which all users automatically belong. This implicit group simplifies the specification of ACLs in a topic tree. In particular, this group is used in the specification of the ACL for the topic root. Note that the default setting of the topic root allows publish and subscribe operation for the "PublicGroup". You can view and change this ACL using the Control Center, but you cannot remove it. It determines the default permissions for the entire topic tree. You can specify ACLs for the "PublicGroup" elsewhere in the topic tree, wherever you want to define permissions for all users.

If you have a principal named "Public" defined in your existing security environment, you cannot use this for topic-based security. If you specify this principal within any ACL, it is equated to "PublicGroup" and therefore provides global access in all cases.

Topic-based security

mqbrkrs authorizations

MQSeries Integrator grants special publish/subscribe access control privileges to members of the **mqbrkrs** group, and to the corresponding **Domain mqbrkrs** global group if appropriate (see “Using Windows NT security domains” on page 123 for details).

Brokers need special privileges to perform internal publish and subscribe operations in networks where access control is enabled. When you create a broker in such a network, you must specify a user ID that belongs to the group **mqbrkrs** as the service user ID for the broker (as shown in Table 7 on page 125). The **mqbrkrs** group is given implicit privileges that allow its members to publish, subscribe and request the persistent delivery of messages on the topic root (“”). All other topics will inherit these permissions. If you attempt to configure any ACLs for the **mqbrkrs** group through the Control Center, these ACLs are ignored by MQSeries Integrator.

Resolving ACL conflicts

If the principals in your broker domain include one or more users in more than one group, it is possible that the explicit or inherited ACL values conflict.

- If the user has an explicit user ACL on the topic of interest, this always takes priority and the broker verifies the current operation on that basis.
- If the user does not have an explicit user ACL on the topic of interest, but has explicit user ACLs against an ancestor in the topic tree, the closest ancestor ACL for that user takes priority and the broker verifies the current operation on that basis.
- If there are no explicit user ACLs for the user on the topic of interest or its ancestors, the broker attempts to verify the current operation on the basis of group ACLs:
 - If the user is a member of a group that has an explicit group ACL on the topic of interest, the broker will verify the current operation on the basis of that group ACL.
 - If the user is not a member of a group that has an explicit group ACL on the topic of interest, but is a member of a group with explicit group ACLs against an ancestor in the topic tree, the closest ancestor ACL takes priority and the broker verifies the current operation on that basis.
 - If, at a particular level in the topic tree, the user ID is contained in more than one group with an explicit ACL, permission is granted if any of the specifications are positive, otherwise it is denied.

You can't associate ACLs with topics that include one or more wildcards. However, your client application access is resolved correctly when the subscription registration is made, even when that application specifies a wildcard in the topic.

ACLs and system topics

Messages that are used for internal publish and subscribe operations are published throughout the broker domain using system topics, which begin with the strings “\$SYS” and “\$ISYS”. These topics must be published and subscribed to by members of **mqbrkrs** only, with the exception of the following two scenarios:

1. If you are migrating topics from MQSeries Publish/Subscribe, you can configure ACLs on topics that begin with the string “\$SYS/STREAM” (see “MQSeries Publish/Subscribe” on page 147 for further details about migration).
2. Clients can subscribe to topics that begin with the string “\$SYS”, which allows applications that provide a management function to subscribe to the broker for administrative events.

You are recommended not to configure ACLs on topics that begin with the string “\$ISYS”. You are not prevented from doing so, but they are ignored.

Setting access control on topics

All members of the group `mqbrtpic` are permitted to define and manipulate the ACLs that define which principals are permitted to publish on and subscribe to topics. ACLs can also limit delivery of persistent messages. All defined principals (users or groups) can be associated with any topic: the permissions that can be set are shown in Table 2.

Table 2. ACL permissions

Option	Description
Publish	Permits or denies the principal to publish messages on this topic.
Subscribe	Permits or denies the principal to subscribe to messages on this topic.
Persistent	Specifies whether the principal can receive messages persistently. If the principal is not permitted, all messages are sent non-persistently. Each individual subscription indicates whether the subscriber requires persistent messages.

Persistent access control behavior is not identical to the publish and subscribe control:

- Clients that are denied Publish access have their publication messages refused. Clients that are denied Subscribe access do not receive the publication.
- The persistent access control does not deny the message to subscribers, but denies them persistence, so denied subscribers always receive messages (subject to their subscribe access control), but always have the message sent to them non-persistently, regardless of the persistence of the original message.

Inheritance of security policies

Topics are organized in a hierarchical tree. The ACL of a parent topic can be inherited by some or all of its descendent topics that do not have an explicit ACL. Therefore, it is not necessary to have an explicit ACL associated with each and every topic. Every topic has an ACL policy which is that of its parent. If all parent topics up to the root topic do not have explicit ACLs, that topic inherits the ACL of the root topic.

For example, in the topic tree in Figure 18 on page 90, the topic root is not shown but is assumed to have an ACL for “PublicGroup” that allows permission to publish, subscribe, and receive persistent publications. (The symbol “¬” means “not”.) Table 3 on page 90 summarizes the ACL for each topic in the tree shown.

Topic-based security

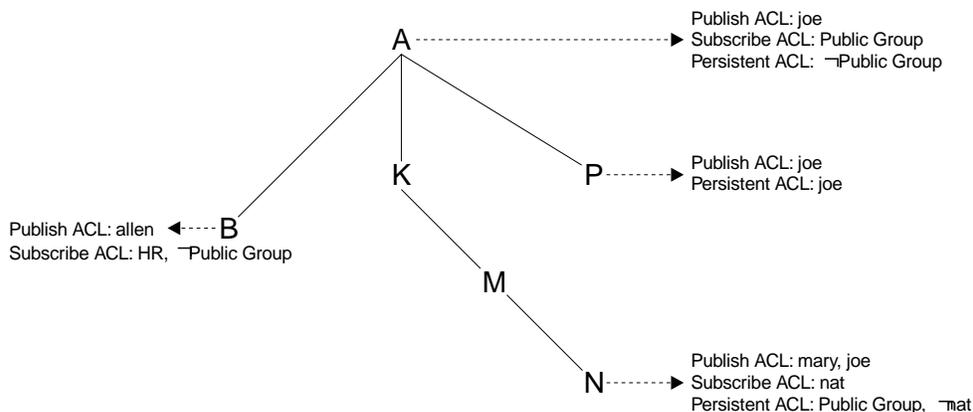


Figure 18. Inheriting ACLs in a topic tree

Table 3. The ACLs for inheritance

Topic	Publishers	Subscribers	Persistent
A	only joe	everyone	no-one
A/P	only joe	everyone	only joe
A/K	only joe	everyone	no-one
A/K/M	only joe	everyone	no-one
A/K/M/N	only mary, joe	everyone	everyone except nat
A/B	allen, joe	HR	no-one

Dynamically created topics

Topics that are not explicitly administered, but are created dynamically in response to client publish or subscribe messages, are treated in the same way as those that are administered, but have no explicit ACLs. That is, the ACLs for dynamically created topics are inherited from the closest ancestor in the topic tree that has an explicit policy. It is therefore not necessary to define leaf topics in the tree if they do not have explicit ACLs.

ACLs and wildcard topics

MQSeries Integrator does not allow you to associate an explicit security policy with a wildcard topic (for example, you cannot associate an ACL with topic “A/+”, which represents a two level hierarchy and includes “A/B”, “A/K”, and “A/P”).

However, MQSeries Integrator does guarantee correct access mediation when a client subscribes to a wildcard topic.

For example, the topic “A/+” does not (and cannot) have a security policy associated with it. Therefore, “A/+” inherits its policy from “A”. Any user can subscribe to “A/+” because the subscribe ACL includes everyone.

When a message is published on “A/P” or “A/K”, the broker delivers it to the user who subscribed to “A/+”. However, when a message is published to “A/B”, that message is only delivered to subscribers who are in the HR group.

If the system administrator changes the subscribe ACL of any topic that matches “A/+”, the broker correctly enforces the ACL when the message is delivered.

Subscribing to a wildcard topic has the semantics to deliver messages on all topics that match the wildcard, and for which the subscriber has authorization to receive that message.

ACLs and subscription resolution

The broker enforces access control through the topic of the message to be delivered. Messages are only delivered to those clients that have not had subscribe access denied, either explicitly or through inheritance. The final decision to deliver a message to a subscriber cannot be made by the broker until a specific message with a topic is being processed. A subscription can contain a wildcard, therefore the actual match against the topic namespace, and hence the topic ACLs, cannot be made at the time the subscription is received.

Activating topic ACL updates

Updates to a topic ACL do not become active until deployed and activated across the MQSeries Integrator broker domain from the Control Center. You must be a member of the group `mqbrops` to deploy ACLs.

Checking publications and subscriptions

The broker makes a number of checks on requests from publishers and subscribers.

The publisher

When a publisher application publishes a message on a topic, the broker verifies that the publisher is authorized to publish on that topic:

- If the publisher is not authorized, the broker rejects the publish request and returns a warning message to the publisher.
- If the publisher is authorized, the broker delivers the message to all authorized subscribers to the topic.

The subscriber

When a broker receives a subscription request, it verifies the following:

- The subscriber has authority to put to the subscriber queue specified in the subscription request. You must set up this authorization using MQSeries facilities: this is independent of the authorizations established for the subscriber in the Control Center.
- No other user is using the same combination of queue name, queue manager name, and correlation ID (if specified).
- The client is permitted to subscribe to the topic, according to the ACL in force for the combination of that topic and user. This can only be checked at this time if the client has not specified a wildcard in the topic for subscription.

If any of these checks fail, the broker rejects the subscription request.

When a broker is ready to deliver a publication, it checks the following:

- The subscriber is authorized to receive persistent publications, if persistent delivery on the topic has been requested.
- The client is permitted to subscribe to that topic, according to the ACL in force for the combination of that topic and user. (Any wildcard the client specified is resolved when a specific message is available: publications have fully-specified topics that do not contain any wildcards.) Messages are delivered only to those clients that have not had subscribe access denied, either explicitly or through inheritance.

Topic-based security

- If a client has subscribed by content, the broker matches the content specified, then checks the topic in the publication and consults the appropriate ACL for permission.

If any of these checks fail, the publication is not delivered to the subscriber.

Detailed information about creating and managing ACLs is provided in *MQSeries Integrator Using the Control Center*.

Summary

This chapter has provided the information you require to make the following design decisions for your publish/subscribe applications:

- The topic trees you use for publications (including the use of wildcards for subscriptions)
- The options you want to use as a publisher (retained, local, other subscribers only)
- The options you want to use as a subscriber (subscription point, filter, local, new publications only, publish on request only)
- The subscriber queues you use to receive publications (with optional correlation identifiers)
- The use of access control lists

For information about writing the applications, having made the design decisions, see the *MQSeries Integrator Programming Guide*.

Part 4. Systems planning

This part further explores the planning needed to establish the correct broker domain topology for your business.

It provides the information needed by your systems administrators to understand the infrastructure required to achieve your business purposes, and to determine how to create a new MQSeries network, or enhance and integrate MQSeries Integrator with your existing MQSeries network.

It contains the following chapters:

- “Chapter 8. System requirements” on page 95
- “Chapter 9. Planning your MQSeries Integrator network” on page 107
- “Chapter 10. Managing your MQSeries Integrator network” on page 133
- “Chapter 11. Enhancing your broker domain” on page 139

The information here is an introduction to the detail provided in the *MQSeries Integrator Administration Guide*.

Chapter 8. System requirements

This chapter summarizes the hardware and software requirements for MQSeries Integrator. It also includes information about licensing agreements and national language support.

The information provided here is an overview: for full details you must refer to the *MQSeries Integrator Installation Guide* for your operating system and to the *Readme.txt* file provided on your product CD which gives the latest and most complete information.

Summary of system requirements

MQSeries Integrator is supported on these operating systems:

- AIX
- HP-UX
- Sun Solaris
- Windows NT — this includes Windows NT Server 4.0 running on an Integrated IBM @server xSeries installed in an IBM @server iSeries 400 (AS/400)

Table 4 summarizes the installation options for all components of each of the four products; MQSeries Integrator for AIX, MQSeries Integrator for HP-UX, MQSeries Integrator for Sun Solaris, and MQSeries Integrator for Windows NT.

The key points to note are:

- You can install and configure the broker only on the operating system for which you have purchased MQSeries Integrator.
- You can install the User Name Server on either Windows NT or the operating system for which you have purchased MQSeries Integrator.
- You must install and configure the Configuration Manager and Control Center on Windows NT.

If you purchase MQSeries Integrator for AIX, MQSeries Integrator for HP-UX, or MQSeries Integrator for Sun Solaris the package includes the MQSeries Integrator for Windows NT CD from which you must install the components that run on Windows NT. Full package details are provided in “Appendix B. The product packages” on page 165.

Table 4. Summary of installation options

Product	Component	System to install on
MQSeries Integrator for AIX	Configuration Manager	Windows NT only
	Control Center	Windows NT only
	Runtime ¹	AIX only
	User Name Server ¹	AIX or Windows NT
	SDK	AIX only
	Online documentation	AIX or Windows NT

Systems summary

Table 4. Summary of installation options (continued)

Product	Component	System to install on
MQSeries Integrator for HP-UX	Configuration Manager	Windows NT only
	Control Center	Windows NT only
	Runtime ¹	HP-UX only
	User Name Server ¹	HP-UX or Windows NT
	SDK	HP-UX only
	Online documentation	HP-UX or Windows NT
MQSeries Integrator for Sun Solaris	Configuration Manager	Windows NT only
	Control Center	Windows NT only
	Runtime ¹	Sun Solaris only
	User Name Server ¹	Sun Solaris or Windows NT
	SDK	Sun Solaris only
	Online documentation	Sun Solaris or Windows NT
MQSeries Integrator for Windows NT	All components including online documentation	Windows NT only
Notes:		
1. The Runtime component on UNIX includes the broker and the User Name Server. You must install this component on UNIX. If you choose to control topic-based security on Windows NT, you can install the User Name Server component on Windows NT. A single licence entitles you to configure a single User Name Server. Therefore if you install the User Name Server component on Windows NT, you cannot configure a second User Name Server on UNIX.		

The following sections identify the requirements for the components that can be installed on each operating system.

System requirements for AIX components

This section summarizes the system requirements for the following components that can be installed on AIX:

- Runtime
- NEONRules and NEONFormatter Support
- Tivoli Interface
- SDK
- Online documentation
- NLS messages

Hardware requirements

The hardware required for AIX components of MQSeries Integrator are:

- One of the following:
 - IBM @server pSeries™ (RS/6000®)
 - IBM RS/6000 POWERserver®
 - IBM RS/6000 POWERstation
 - IBM Scalable POWERparallel®
- Any communications hardware supporting NetBIOS, SNA LU6.2, SPX, or TCP/IP
- A minimum of 512MB of RAM to support run-time operation

Disk space required

The installation requirements depend on which components you install. Full details are in the : requirements range from a minimum of 280MB to approximately 400MB.

Temporary space of approximately 150MB is required in the /tmp directory. This temporary space is freed up when installation is complete.

Software requirements

The following software products are prerequisites for operating MQSeries Integrator. If AIX, MQSeries, and the *Java Development Kit (JDK™)* are not at the correct level, installation will not continue. For optional products that you can use with MQSeries Integrator see the *MQSeries Integrator for AIX Installation Guide*.

Minimum supported levels are shown. Later compatible levels, if any, are supported unless otherwise stated. You must refer to the *MQSeries Integrator for AIX Installation Guide* and the *Readme.txt* file to check latest Corrective Service Diskette (CSD) and FixPack details for these products.

- AIX Version 4.3.3

You must ensure that your AIX system includes JDK 1.1.8. This level is supplied with the AIX CDs.

- MQSeries for AIX Version 5.1

This must be at service level CSD4 or above.

The installation program checks that you have MQSeries for AIX Version 5.1 or later installed, and that it is at the correct service level. For full details of which MQSeries Integrator component requires which MQSeries component, see the *MQSeries Integrator for AIX Installation Guide*.

The MQSeries for AIX Version 5.2 product is supplied in the MQSeries Integrator package. You do not have to install any additional CSDs if you install MQSeries from this CD. CSD6 for Version 5.1 is also supplied independently in this package: if you already have MQSeries Version 5.1 you can install this latest CSD if you choose not to install MQSeries for AIX Version 5.2.

Note: Version 5.0 is not supported at any service level.

- A database for internal broker data

The MQSeries Integrator broker requires access to a database for internal caching and for storing internal control information.

If your installation choices require a database to be present for internal broker support, the MQSeries Integrator installation program checks for a suitable database installed on this system.

If you already have a database product in the supported list below, you can use it to support MQSeries Integrator.

If the installation program detects that you have a level of database prior to those indicated here, it highlights the need to upgrade your existing license. You must upgrade your database before you can use MQSeries Integrator.

Supported databases are:

- DB2 Universal Database for AIX Version 6.1 (Enterprise Edition, Connect Enterprise Edition, or Extended Enterprise Edition)
- Oracle Version 8.1.6
- Sybase Version 12

Software requirements

Each database requires an ODBC driver: the driver for DB2 is supplied by DB2, the drivers for Oracle and Sybase are included with MQSeries Integrator.

If you do not have a suitable database installed, you can install DB2 Version 7.1, which is included in the MQSeries Integrator package.

DB2 installation requires an additional 250MB of disk storage. You will also need approximately 10MB for each set of tables you create (for the broker tables, for the configuration repository, and for message repository).

DB2 has no additional prerequisites.

This DB2 product has restricted license terms and agreements. You must only use this DB2 installation in association with your licensed use of MQSeries Integrator for message management, and only the MQSeries Integrator components can make calls to the DB2 database.

The use of a database by the MQSeries Integrator components is independent of the use of databases by your applications. You are not restricted to the databases listed here for application and data storage and retrieval. If you have a requirement for XA coordination, your choice of database can be affected. See the *MQSeries Integrator Administration Guide* for more details about how MQSeries Integrator supports transactions.

For further details of database support for brokers, see Table 5 on page 105.

System requirements for HP-UX components

This section summarizes the system requirements for the following components that can be installed on HP-UX:

- Runtime
- NEONRules and NEONFormatter Support
- SDK
- Online documentation
- NLS messages

Hardware requirements

The hardware requirements for the HP-UX components of MQSeries Integrator are:

- Any HP-UX desktop or server system
- Any communications hardware supporting NetBIOS, SNA LU6.2, SPX, or TCP/IP
- A minimum of 512MB of RAM to support runtime operation

Disk space required

The installation requirements depend on which components you install. Full details are in the *MQSeries Integrator for HP-UX Installation Guide*: requirements range from a minimum of 280MB to approximately 400MB.

Temporary space of approximately 150MB is required in the /tmp directory. This temporary space is freed up when installation is complete.

Software requirements

The following software products are prerequisites for operating MQSeries Integrator. These prerequisites are checked at installation time but do not cause

installation to fail. However, you must ensure these prerequisite products are available before you start to use MQSeries Integrator. For optional products that you can use with MQSeries Integrator see the *MQSeries Integrator for HP-UX Installation Guide*.

Minimum supported levels are shown. Later compatible levels, if any, are supported unless otherwise stated. You must refer to the *MQSeries Integrator for HP-UX Installation Guide* and the `Readme.txt` file to check latest Corrective Service Diskette (CSD) and FixPack details for these products.

- HP-UX Version 11 with HP-UX patches XSWGR1100, XSWHWCR1100, and PHSS_21906
- HP-UX Version 11 patch PHSS_22478 for Oracle support
- IBM MQSeries for HP-UX Version 5.2

The installation program checks that you have MQSeries for HP-UX Version 5.2 installed, and that it is at the correct service level. For full details of which MQSeries Integrator component requires which MQSeries component, see the *MQSeries Integrator for HP-UX Installation Guide*.

The MQSeries for HP-UX Version 5.2 product is supplied in the MQSeries Integrator package. You do not have to install any additional CSDs if you install MQSeries from this package.

Note: Version 5.0 and Version 5.1 are not supported at any service level.

- A database for internal broker data.

The MQSeries Integrator broker requires access to a database for internal caching and for storing internal control information.

If your installation choices require a database to be present for internal broker support, the MQSeries Integrator installation program checks for a suitable database installed on this system.

If you already have a database product in the supported list below, you can use it to support MQSeries Integrator.

If the installation program detects that you have a level of database prior to those indicated here, it highlights the need to upgrade your existing license. You must upgrade your database before you can use MQSeries Integrator.

Supported databases are:

- DB2 Universal Database for HP-UX Version 7.1 (Enterprise Edition, Connect Enterprise Edition, or Extended Enterprise Edition)
- Oracle Version 8.1.6

Each database requires an ODBC driver: the driver for DB2 is supplied by DB2, the drivers for Oracle and Sybase are included with MQSeries Integrator.

If you do not have a suitable database installed, you can install DB2 Version 7.1, which is included in the MQSeries Integrator package.

DB2 installation requires an additional 250MB of disk storage. You will also need approximately 10MB for each set of tables you create (for the broker tables, for the configuration repository, and for message repository).

DB2 has no additional prerequisites.

This DB2 product has restricted license terms and agreements. You must only use this DB2 installation in association with your licensed use of MQSeries

Software requirements

Integrator for message management, and only the MQSeries Integrator components can make calls to the DB2 database.

The use of a database by the MQSeries Integrator components is independent of the use of databases by your applications. You are not restricted to the databases listed here for application and data storage and retrieval. If you have a requirement for XA coordination, your choice of database can be affected. See the *MQSeries Integrator Administration Guide* for more details about how MQSeries Integrator supports transactions.

For further details of database support, see Table 5 on page 105.

System requirements for Sun Solaris components

This section summarizes the system requirements for the following components that can be installed on Sun Solaris:

- Runtime
- NEONRules and NEONFormatter Support
- Tivoli Interface
- SDK
- Online documentation
- NLS messages

Hardware requirements

The hardware requirements for the Sun Solaris components of MQSeries Integrator are:

- Any Sun SPARC or UltraSPARC desktop or server system
- Any communications hardware supporting NetBIOS, SNA LU6.2, SPX, and TCP/IP
- A minimum of 512MB of RAM to support runtime operation

Disk space required

The installation requirements depend on which components you install. Full details are in the *MQSeries Integrator for Sun Solaris Installation Guide*: requirements range from a minimum of 280MB to approximately 400MB.

Temporary space of approximately 150MB is required in the /tmp directory. This temporary space is freed up when installation is complete.

Software requirements

The following software products are prerequisites for operating MQSeries Integrator. These prerequisites are checked at installation time but do not cause installation to fail. However, you must ensure these prerequisite products are available before you start to use MQSeries Integrator. For optional products that you can use with MQSeries Integrator see the *MQSeries Integrator for Sun Solaris Installation Guide*.

Minimum supported levels are shown. Later compatible levels, if any, are supported unless otherwise stated. You must refer to the *MQSeries Integrator for Sun Solaris Installation Guide* and the Readme.txt file to check latest Corrective Service Diskette (CSD) and FixPack details for these products.

- Sun Solaris 7 with the latest Sun recommended patches
- IBM MQSeries for Sun Solaris Version 5.1

This must be at service level CSD4 or above.

The installation program checks that you have MQSeries for Sun Solaris Version 5.1 or later installed, and that it is at the correct service level. For full details of which MQSeries Integrator component requires which MQSeries component, see the *MQSeries Integrator for Sun Solaris Installation Guide*.

The MQSeries for Sun Solaris Version 5.2 product is supplied in the MQSeries Integrator package. You do not have to install any additional CSDs if you install MQSeries from this CD. CSD6 for Version 5.1 is also supplied independently in this package: if you already have MQSeries Version 5.1 you can install this latest CSD if you choose not to install MQSeries for Sun Solaris Version 5.2.

Note: Version 5.0 is not supported at any service level.

- A database for internal broker data.

The MQSeries Integrator broker requires access to a database for internal caching and for storing internal control information.

If your installation choices require a database to be present for internal broker support, the MQSeries Integrator installation program checks for a suitable database installed on this system.

If you already have a database product in the supported list below, you can use it to support MQSeries Integrator.

If the installation program detects that you have a level of database prior to those indicated here, it highlights the need to upgrade your existing license. You must upgrade your database before you can use MQSeries Integrator.

Supported databases are:

- DB2 Universal Database for Sun Solaris Version 6.1 (Enterprise Edition, Connect Enterprise Edition, or Extended Enterprise Edition)
- Oracle Version 8.1.6
- Sybase Version 12

Each database requires an ODBC driver: the driver for DB2 is supplied by DB2, the drivers for Oracle and Sybase are included with MQSeries Integrator.

If you do not have a suitable database installed, you can install DB2 Version 7.1, which is included in the MQSeries Integrator package.

DB2 installation requires an additional 250MB of disk storage. You will also need approximately 10MB for each set of tables you create (for the broker tables, for the configuration repository, and for message repository).

DB2 has no additional prerequisites.

This DB2 product has restricted license terms and agreements. You must only use this DB2 installation in association with your licensed use of MQSeries Integrator for message management, and only the MQSeries Integrator components can make calls to the DB2 database.

The use of a database by the MQSeries Integrator components is independent of the use of databases by your applications. You are not restricted to the databases listed here for application and data storage and retrieval. If you have a requirement for XA coordination, your choice of database can be affected. See the *MQSeries Integrator Administration Guide* for more details about how MQSeries Integrator supports transactions.

Software requirements

For further details of database support, see Table 5 on page 105.

System requirements for Windows NT components

This section summarizes the system requirements for the following components that can be installed on Windows NT:

- Configuration Manager
- Control Center
- Broker
- User Name Server
- SDK
- Online documentation

Hardware requirements

The hardware requirements for the Windows NT components are:

- Any Year 2000 compliant Intel Pentium® II (or above) processor-based IBM PC machine or compatible, that is explicitly compatible and fully capable of running the specified operating system, all the corresponding supporting software shown below, and any associated applications unmodified. This machine could be an Integrated IBM @server xSeries Server installed in an IBM @server iSeries 400 (AS/400).
- Any communications hardware supporting NetBIOS, SNA LU 6.2, SPX, and TCP/IP.
- If all components are installed on a single system (MQSeries Integrator for Windows NT only), a minimum of 512 megabytes (MB) of RAM are recommended to support run-time operation.
- If only the Configuration Manager and Control Center components are installed on a single system, a minimum of 300 megabytes (MB) of RAM are recommended to support runtime operation.
- If the Configuration Manager, the Control Center, and the User Name Server are installed on a single system, a minimum of 350 megabytes (MB) of RAM are recommended to support runtime operation.

Disk space required

The installation requirements depend on which components you install and how much working space you need. See the *MQSeries Integrator Installation Guide* for the appropriate operating system for full details.

If DB2 is installed by the MQSeries Integrator installation program, an additional 250MB is required.

Temporary space of 150MB (for a full installation) to 300MB (for a custom installation) is required on the operating system drive.

Software requirements

The following software products are prerequisites for operating MQSeries Integrator. These prerequisites are checked at installation time but do not cause installation to fail. However, you must ensure these prerequisite products are available before you start to use MQSeries Integrator. For optional products that you can use with MQSeries Integrator see the *MQSeries Integrator for Windows NT Installation Guide*.

Software requirements

Minimum supported levels are shown. Later compatible levels, if any, are supported unless otherwise stated. You must refer to the *MQSeries Integrator for Windows NT Installation Guide* and the *Readme.txt* file to check latest Corrective Service Diskette (CSD) and FixPack details for these products.

- Microsoft Windows NT Version 4.0, including TCP/IP, NetBIOS, and SPX, with Service Pack 5 or Service Pack 6A, either of which provides relevant Year 2000 fixes and Euro support.

Note: Service Pack 6 is not supported.

Both Windows NT Workstation and Windows NT Server products are supported. You can download Windows NT upgrades from the Microsoft Web site at:

<http://support.microsoft.com/directory/>

If you intend to run the Tour feature of the Control Center, you must install Microsoft Internet Explorer Version 5.

- IBM MQSeries for Windows NT Version 5.1.

This must be at service level CSD4 or above.

The installation program checks that you have MQSeries for Windows NT Version 5.1 or later installed, and that it is at the correct service level. For full details of which MQSeries Integrator component requires which MQSeries component, see the *MQSeries Integrator for Windows NT Installation Guide*.

The MQSeries for Windows NT Version 5.2 product is supplied in the MQSeries Integrator package. You do not have to install any additional CSDs if you install MQSeries from this CD. CSD6 for Version 5.1 is also supplied independently in this package: if you already have MQSeries Version 5.1 you can install this latest CSD if you choose not to install MQSeries Version 5.2.

Note: Version 5.0 is not supported at any service level.

MQSeries for Windows NT requires a number of other software products to install and operate a server successfully.

MQSeries for Windows NT server prerequisites are:

- Internet Explorer Version 5.

This is available from the Microsoft Web site at:

<http://www.microsoft.com>

- Active Directory Services Interface Version 2.0.

This is provided on the MQSeries CD.

- Microsoft Management Console Version 1.1.

This is provided on the MQSeries CD.

- Microsoft Data Access Component (MDAC) Version 2.5

MQSeries Integrator Version 2.0.2 uses ODBC drivers, and these require that the level of the installed Microsoft Data Access Component (MDAC) on your system is 2.5. The install program for MQSeries Integrator Version 2.0.2 checks this level and issues a warning if there is a need to update your system, however the install will not automatically upgrade your system. In order to update your system you can run the MDAC setup program from the supplemental materials CD supplied with the product. You are recommended to download the latest level of MDAC from <http://www.microsoft.com/> — note that different countries require different setup programs.

Software requirements

If you install only an MQSeries client with your MQSeries Integrator components, check the client installation details in the MQSeries Release Notes folder to determine the client's prerequisites.

- A database product.

MQSeries Integrator broker and Configuration Manager components require access to a database for internal caching and for storing internal control information. The Control Center and User Name Server do not need access to a database.

You can use any of the database products listed below:

- DB2 Universal Database for Windows NT Version 6.1 (Enterprise Edition, Connect Enterprise Edition, or Extended Enterprise Edition).

You must use this database for the Configuration Manager requirements. This database can also be used with the broker component.

- Microsoft SQL Server Version 6.5 with Service Pack 5a or Version 7 with Service Pack 1, both of which are Year 2000 compatible.

This database can be used with the broker component.

- Oracle Version 8.1.6.

This database can be used with the broker component.

- Sybase Version 12.

This database can be used with the broker component.

The installation program checks for a current DB2 installation, and determines the level installed. If DB2 is not present, or needs to be upgraded, installation asks you if you want to install DB2 7.1 from the MQSeries Integrator Version 2.0.2 product package. You can cancel this if you are using another database, or plan to install a suitable database product after you have installed MQSeries Integrator.

Each database requires an ODBC driver: the drivers for DB2 and SQL Server are supplied by the database product, the drivers for Oracle and Sybase are included with MQSeries Integrator.

If you do not have a suitable database installed, the MQSeries Integrator installation program launches the installation program for DB2 Version 7.1, which is included in the MQSeries Integrator package.

DB2 installation requires an additional 420MB of disk storage. You will also need approximately 10MB for each set of tables you create (for the broker tables, for the configuration repository, and for message repository).

DB2 has no additional prerequisites.

This DB2 product has restricted license terms and agreements. You must only use this DB2 installation in association with your licensed use of MQSeries Integrator for message management, and only the MQSeries Integrator components can make calls to the DB2 database.

The use of a database by the MQSeries Integrator components is independent of the use of databases by your applications. You are not restricted to the databases listed here for application and data storage and retrieval. If you have a requirement for XA coordination, your choice of database can be affected. See the *MQSeries Integrator Administration Guide* for more details about how MQSeries Integrator supports transactions.

For further details of database support, see Table 5.

Database support

Table 5 summarizes the supported versions of databases that you can use for the broker database and for user databases accessed in message flow nodes on each operating system.

Table 5. 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.

Client requirements

You can run MQSeries Integrator applications on all platforms for which MQSeries provides a client.

License information

Under the terms of the MQSeries Integrator Version 2.0.2 license agreement, you can install one instance of each component at any one time on any one system, with the exception of the Control Center. You can install the Control Center on multiple systems provided that each Control Center is interacting with the same single Configuration Manager. You can create multiple brokers on a single system.

For details of which component can be installed on which operating system, see Table 4 on page 95.

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

National language support

- 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.

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.

For further information about changing language settings, refer to the *MQSeries Integrator Administration Guide*

Chapter 9. Planning your MQSeries Integrator network

This chapter provides information about how you plan a network of MQSeries Integrator resources that supports your business processes. It discusses:

- “Planning MQSeries Integrator resources”
- “Designing the MQSeries infrastructure” on page 114
- “Planning database resources” on page 118
- “Planning security” on page 120
- “Planning for data conversion” on page 131

Planning MQSeries Integrator resources

When you plan an MQSeries Integrator network, you must consider what components you will install, and where, and how you will organize and use them together. The information here helps you to do that, by explaining the initial considerations and by identifying decisions you must make.

Note: You must configure your broker domain subject to your license agreement, described in “License information” on page 105.

The following areas are discussed:

- “Naming conventions”
- “Broker domain basics” on page 109
- “Client applications” on page 113

Naming conventions

When you plan a new MQSeries Integrator network, one of your first tasks must be to establish a convention for naming the resources that you will create within this network. There are three aspects to this:

- “MQSeries Integrator resources”
- “MQSeries resources” on page 108
- “Database resources” on page 109

MQSeries Integrator resources

A naming convention for MQSeries Integrator resources throughout your network ensures that names are unique, and that users creating new resources can be confident of not introducing duplication or confusion.

The resources you must create and name within an MQSeries Integrator network are:

- **Brokers.** When you create a broker, you give it a name that must be unique within your broker domain. You must use the same name for the same broker when you create it on the system in which it is installed (using the command **mqsicreatebroker**) and when you create a reference to that broker in the broker domain topology in the Control Center. The latter is a representation of the physical broker (created by **mqsicreatebroker**) in the configuration repository, and this single name links the two. Broker names are case-sensitive except on Windows NT.
- **Execution groups.** Each execution group name must be unique within a broker.

MQSeries Integrator network

- Message flows and message processing nodes. Each message processing node must be unique within the message flow it is assigned to. For example, if you include two MQOutput nodes to a single message flow, you must provide a unique name for each.

Message flow names must be unique within the broker domain. Any reference to that name within the broker domain is always to the same message flow. You can therefore assign the same message flow to many brokers.

- Message sets and messages. Each message name must be unique within the message set to which it belongs.

Message set names must be unique within the broker domain. Any reference to that name within the broker domain is always to the same message set. You can therefore assign the same message set to many brokers.

The Configuration Manager and User Name Server are not allocated names when you create them. They are identified only by the name of the MQSeries queue manager that hosts the services they provide.

There are a few restrictions for naming resources: see the *MQSeries Integrator Administration Guide* for details.

MQSeries resources

All MQSeries Integrator resources have dependencies on MQSeries services and objects. You must therefore also consider what conventions you will adopt for MQSeries object names. If you already have an MQSeries naming convention, you are recommended to use a compatible extension of this convention for MQSeries Integrator resources.

When you create a broker or a Configuration Manager, you must specify a queue manager name. This queue manager is created for you if it does not already exist. Because the broker and Configuration Manager each use a unique set of MQSeries queues, they can share one queue manager, if appropriate. However, every broker must have a dedicated queue manager. Note that the MQSeries Everyplace queue manager name specified must be unique across MQSeries Everyplace and MQSeries, within MQSeries Integrator.

If you set up a User Name Server in your broker domain, this also uses a unique set of MQSeries queues. The User Name Server can therefore also share a queue manager with a broker, or the Configuration Manager, or both.

You must ensure that every queue manager name is unique within your network of interconnected queue managers, whether or not every queue manager is in your MQSeries Integrator network. This ensures that each queue manager can unambiguously identify the target queue manager to which any given message must be sent, and that MQSeries Integrator applications can also interact with basic MQSeries applications.

MQSeries supports a number of objects defined to queue managers. These objects (queues, channels, and processes) also have naming conventions and restrictions, that are defined in the *MQSeries Command Reference*. In summary, the restrictions are:

- All names must be a maximum of 48 characters in length (channels have a maximum of 20 characters).
- The name of each object must be unique within its type (for example, queue or channel).

- Names for all objects starting with the characters “SYSTEM.” are reserved for use by IBM.

For full details of all restrictions and recommendations, you must refer to the *MQSeries Planning Guide* and to *MQSeries System Administration*.

Database resources

You must consider the naming conventions you use for databases, both for databases you create for MQSeries Integrator product use (for broker tables, the configuration repository, and the message repository), and for databases you create for application use.

Your configuration and message repositories are owned and managed by the Configuration Manager: because there is only one Configuration Manager you should not find any conflict with names. Database tables used for brokers can be unique and local to the broker, or can be *shared* because the rows of the tables specific to each individual broker incorporate the name of the broker. You might need to align the naming of all of these databases with other databases in use in your broker domain.

For details of the database tables created for MQSeries Integrator use, see the *MQSeries Integrator Installation Guide* for your product.

You must also ensure that the databases used for application data (accessed through message flows) are uniquely named throughout your network, so there is no opportunity for confusion or error.

Broker domain basics

This section covers the following topics:

- “General guidelines”
- “Supporting publish/subscribe services” on page 110

General guidelines

Before you start planning a full deployment of MQSeries Integrator, you must understand a few basic rules and recommendations:

- You must install and initialize a single Configuration Manager within your broker domain. This component controls and maintains all configuration and administration information for all the components, and the resources defined to those components in the configuration repository. It also manages the message repository that contains all definitions created through the Control Center. The Configuration Manager therefore defines the scope of the broker domain. It is in constant contact with all other components created and deployed in the broker domain.

When you create the Configuration Manager, you specify the security domain that is used to check users’ authority to complete tasks in the broker domain. For a discussion of security in the broker domain, see “Planning security” on page 120.

- You must install at least one Control Center. This provides your only means of viewing and managing the configuration and message repositories maintained by the Configuration Manager. The Control Center is a central point of control for the business processes of your broker domain, enabling you to create and modify messages and message flows, and assign and deploy these resources to the brokers.

MQSeries Integrator network

The Control Center does not control system administration aspects of the broker domain. System administration (for example, creation and activation of a broker) is supported by a set of commands.

- You must install and initialize at least one broker. The broker supports the services (defined as message flows acting on messages) that are required by your applications. You must also use the Control Center to define this broker to the Configuration Manager (using the same name in both places), and deploy the broker domain topology, to register and activate this broker in the broker domain. Deployment initiates the communications between the Configuration Manager and the broker.

The *MQSeries Integrator Installation Guide* illustrates a step-by-step approach to setting up a very simple broker domain configuration using the components listed above. It also illustrates how you can expand that simple broker domain by creating a User Name Server to employ topic-based security (this is discussed in “Supporting publish/subscribe services”).

The configuration tasks for establishing a broker domain are supported by a set of commands you can enter at the command prompt. A subset of these commands (to create, modify, and delete) are also available through a graphical administrative interface, the MQSeries Integrator Command Assistant.

For example, you can create a broker using the command **mqscreatebroker**, delete it using the command **mqsdeletebroker**, and modify its properties using the command **mqschangebroker**. For full details of all these commands, and the Command Assistant, see the *MQSeries Integrator Administration Guide*.

Supporting publish/subscribe services

If your applications exploit publish/subscribe services, there are two additional considerations for planning your broker domain:

- “Setting up collectives”.
- “Employing topic-based security” on page 112.

These are both optional: you can support publish/subscribe services without implementing either of these options.

Setting up collectives: A collective is a set of one or more brokers that are directly connected to each other. A single broker can belong to only one collective. Brokers within one collective can exist on the same physical system, or on separate systems.

A collective provides these benefits:

- Messages destined for a specific broker in the same collective are transported directly to that broker and do not need to pass through any intermediate broker. This improves broker performance and optimizes inter-broker publish/subscribe traffic, relative to a hierarchical tree configuration.
- If your clients are geographically dispersed, you can set up a collective in each location, and connect the collectives (by joining a single broker in each collective) to optimize the flow of publications and subscription registrations through the network.
- You can group clients according to the shared topics they publish and subscribe to.

MQSeries Integrator network

Clients that share common topics can connect to brokers within a collective. The common publications will be transported efficiently within the collective, because they will not pass through any brokers that do not have clients with an interest in those common topics.

- A client can connect to its nearest broker, to improve its own performance. The broker receives all messages that match the subscription registration of the client from all brokers within the collective.

Client application performance is also enhanced for any other service requested from this broker, or the broker's queue manager. A single client application can use both publish/subscribe and point-to-point messaging.

- The number of clients per broker can be reduced by adding more brokers to the collective to share workload within that collective.

Figure 19 illustrates one way of connecting your publish/subscribe brokers in collectives.

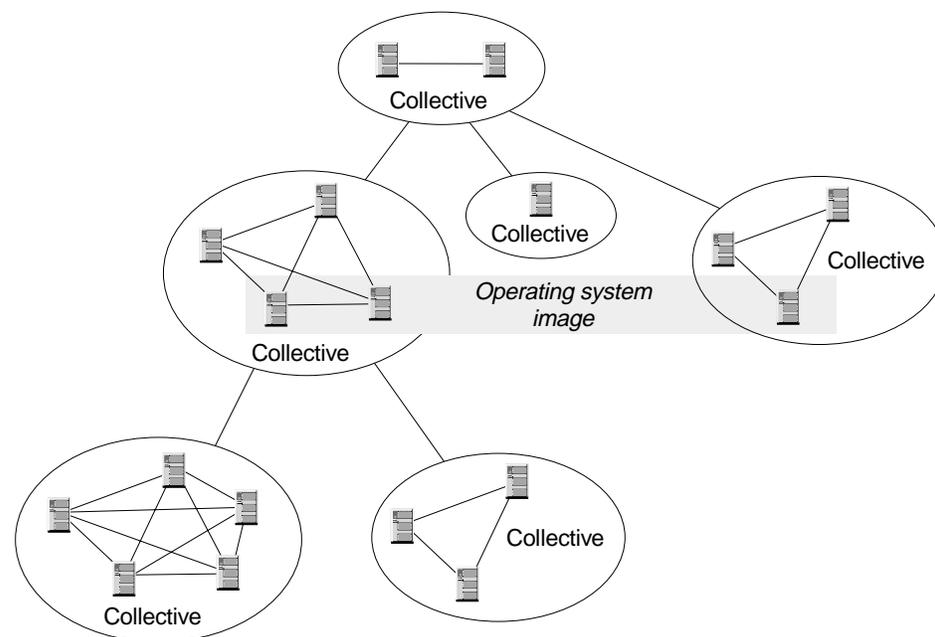


Figure 19. Collectives with a broker domain

When you create a collective, the Control Center ensures that the connections you make to other collectives and brokers are valid. You are prevented from making connections that would cause messages to cycle forever within the network. You are also prevented from deploying a collective of brokers that does not have the required MQSeries connections already defined.

Each broker in the collective maintains a list of its neighbors. A neighbor is a broker in the same collective, or a broker outside its own collective to which it has an explicit connection (that is, for which it is acting as a gateway). The complete list of neighboring brokers forms a broker's neighborhood.

Any broker with at least one deployed execution group can receive publications and subscription registrations, and receive and pass on publications from or to its neighbors, even if you have not assigned and deployed any message flow containing a publication node to that broker.

MQSeries Integrator network

Employing topic-based security: You can choose to limit application access to particular messages. For example, if a client application publishes messages containing sensitive company finance information, or personnel details, you might want to restrict who has access to those messages.

If you want to restrict message access in this way, you must:

- Install and configure a User Name Server to provide information on the principals valid in your broker domain.

When you create the User Name Server, you specify the security domain that is used to check users' authority to publish on and subscribe to specific topics, and their authority to request persistent delivery. For a discussion of security in the broker domain, see "Planning security" on page 120.

- Ensure that a topic is associated with every message that is to be restricted (either specified explicitly in the message by the publisher, or associated with the message by the input node when it gets the message from the input queue).
- Create Access Control Lists (ACLs), through the Control Center to associate principals with topics.

You are strongly recommended to configure a single User Name Server in your broker domain. However, there are circumstances in which it is appropriate to consider creating more than one (subject to your license agreement):

- Performance

If you have a large number of brokers in your broker domain, the requests they send to the User Name Server can be handled more quickly. You could also benefit if your broker domain configuration is complex, and brokers can interact more efficiently (in terms of network traffic) if more than one User Name Server is installed.

- Resilience

Although no standby mechanism is provided by MQSeries Integrator, you might want to be able to redirect requests to a second User Name Server if a system error occurs on the system of your first User Name Server.

If you do have more than one User Name Server, and more than one is active at once, you must ensure that all of them are able to reference a single source of principal definitions.

You must also ensure that each User Name Server is associated with a unique MQSeries queue manager, to ensure that the User Name Server associated with the Configuration Manager and each broker can be identified, and that there is no conflict in the User Name Server's use of MQSeries fixed name queues.

For more details of administering User Name Servers in your broker domain, see the *MQSeries Integrator Administration Guide*.

How publications and subscriptions flow through the network: When a client registers a subscription, the broker registers a matching subscription with its neighbors. This is called a "proxy subscription". If an identical subscription has already been registered, the broker does not register again: only one proxy subscription will be in effect at any one time. Likewise, when a client de-registers a subscription from a broker, the broker de-registers the proxy subscription from its neighbors, if the client is the last (or only) client for which the broker is holding the proxy.

Content-based filters are not included in proxy subscriptions. Therefore a superset of messages might be received by the broker to which a subscriber that specified a content filter is registered, but will not be passed on to that subscriber by its local broker unless there is a content match.

All proxy subscriptions are made with the *PersistenceAsPublisher* option. This results in messages being delivered to neighboring brokers with the persistence specified by the publisher. Client subscription persistence options only take effect at the local broker (that is, the broker with which the clients have registered).

Therefore a subscriber that requests persistent delivery always receives a persistent message for matching publications. However, the message could be delivered through the broker network as a non-persistent message if this was specified by the publisher. If a problem occurs during the transmission of a message between publisher and subscriber, it is therefore possible that the subscriber will never get the message despite specifying persistent delivery as an option on subscription registration.

Client applications

MQSeries Integrator client applications are applications that use the services provided by the message flows deployed within one or more brokers in the broker domain.

These applications can use one of two techniques for gaining access to the broker's services:

- An application can use an MQSeries client connection. You can use all of the MQSeries clients supported by MQSeries Version 5.1 or later, giving you the freedom to connect applications running in a wide variety of environments into your broker domain. An application running on the same system as the queue manager to which it connects can also use a client connection.
- An application running on the same system as a broker can use a local connection to the queue manager that hosts that broker.

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.

The Control Center application

The Control Center is a special MQSeries Integrator client application. It uses an MQSeries client for Java connection over TCP/IP to the broker that hosts the Configuration Manager, regardless of whether the Configuration Manager is on the same system as the Control Center or a different system.

When the Configuration Manager is created, the required server connection channel is defined. This allows any number of Control Center clients to connect to the Configuration Manager's queue manager. When you invoke the Control Center, it dynamically creates the client connection channel to complete the connection with the Configuration Manager.

Designing the MQSeries infrastructure

MQSeries Integrator depends on the MQSeries transport services to support internally generated communications between components. Some of these resources are created for you, when you create MQSeries Integrator components that depend on them. Others depend on the exact setup of your broker domain, and you must therefore create these resources yourself.

Communications between MQSeries Integrator components are protocol-independent, with the exception of the connection between every instance of the Control Center and the Configuration Manager. This must be a TCP/IP connection, as must connections to the MQSeries Everyplace and SCADA nodes. Other connections can use any of the protocols supported by the MQSeries messaging product for the operating system for your MQSeries Integrator product.

Except for MQSeries Everyplace and SCADA applications, applications that use broker services must also use MQSeries to send and receive all messages. The resources required by your applications (queues and client connection and server connection channels) are application specific, and you must therefore create these resources yourself.

The information here concentrates on the specific requirements that MQSeries Integrator imposes on an MQSeries network. For a full description of designing and connecting an MQSeries network, see *MQSeries Intercommunication*, which covers the basics, such as setting up transmission queues and channels, in detail.

For more specific details of how to implement the MQSeries infrastructure for your MQSeries Integrator broker domain, see the *MQSeries Integrator Administration Guide*.

This section includes the following information:

- “MQSeries resources for brokers”
- “MQSeries resources for the Configuration Manager” on page 115
- “MQSeries resources for the User Name Server” on page 116
- “MQSeries resources for the Control Center” on page 116
- “MQSeries resources for client applications” on page 116
- “MQSeries clusters” on page 117

MQSeries resources for brokers

Each broker depends on a number of MQSeries resources, some of which are always required, others are dependent on the broker domain setup:

1. Each broker must be associated with a queue manager to host its services. You must specify a queue manager name when you create the broker. If this queue manager does not exist, it is created for you.

The broker cannot share a queue manager with any other broker. However, it can share a queue manager with the Configuration Manager, or the User Name Server, or both, subject to the installation options in effect for your MQSeries Integrator product. See Table 4 on page 95 for a summary of the options available.

The broker and its queue manager can share the same name, subject to naming restrictions for both products.

2. Each broker must have a number of fixed-name queues on its queue manager. These allow it to exchange information with other components in the broker

domain. These queues are defined for you when the broker is created. The use of these fixed-name queues dictates that each broker to be hosted by a unique queue manager.

3. Each broker must communicate with the Configuration Manager. If the broker and the Configuration Manager do not share a queue manager, you must define the channels and transmission queues that support communications between the two queue managers.
4. If you have included a User Name Server in your broker domain, each broker must communicate with it. If the broker and the User Name Server do not share a queue manager, you must define transmission queues and channels that support two-way communications between the two queue managers.
5. The broker's queue manager must have a listener to receive messages from other components that do not share its queue manager, and from clients on other physical systems. You must create a listener for every protocol used for connections to the broker. If any connection uses the TCP/IP protocol, you must decide which port the listener must listen on.
6. If the broker is connected to other brokers, either in a collective, or to communicate with another collective, the queue manager needs transmission queues and channel definitions to support two-way communications with each of the other brokers' queue managers.

MQSeries resources for the Configuration Manager

The Configuration Manager depends on a number of MQSeries resources, some of which must be available, others are dependent on the broker domain setup:

1. The Configuration Manager must be associated with a queue manager to host its services. You must specify a queue manager name when you create the Configuration Manager. If this queue manager does not exist, it is created for you. The Configuration Manager can share a queue manager with a broker, or the User Name Server, or both, subject to the installation options in effect for your MQSeries Integrator product. See Table 4 on page 95 for a summary of the options available.
2. The Configuration Manager must have a number of fixed-name queues on its queue manager. These allow it to exchange information with other components in the broker domain. These queues are defined for you when the Configuration Manager is created.
3. The Configuration Manager must communicate with every broker in the broker domain. You must define transmission queues and channels to support two-way communications between the Configuration Manager and every broker except the one (if defined) that shares its queue manager.
4. If you have included a User Name Server in your broker domain, the Configuration Manager must communicate with it. If the Configuration Manager and the User Name Server do not share a queue manager, you must define transmission queues and channels to support two-way communications between the two queue managers.
5. The Configuration Manager's queue manager must have a listener to receive messages from the Control Center, and from other components and clients that do not share its queue manager. You must create a listener for every protocol used for the inter component connections. If the connection is TCP/IP you must also decide which port the listener must listen on: no other listener must be active on this port.
6. The Configuration Manager's queue manager must have a server connection. This is defined for you when the Configuration Manager is created. Every Control Center client can use this single definition.

MQSeries resources for the User Name Server

The User Name Server depends on a number of MQSeries resources, some of which must be available, others are dependent on the broker domain setup:

1. The User Name Server must be associated with a queue manager to host its services. You must specify a queue manager name when you create the User Name Server. If this queue manager does not exist, it is created for you.
The User Name Server can share a queue manager with a broker, or the Configuration Manager, or both, subject to the installation options in effect for your MQSeries Integrator product. See Table 4 on page 95 for a summary of the options available.
2. The User Name Server must have a number of fixed-name queues on its queue manager. These allow it to exchange information with other components in the broker domain. These queues are defined for you when the User Name Server is created.
3. The User Name Server must communicate with the Configuration Manager. If the two do not share a queue manager, you must define the transmission queues and channels to support two-way communications between the two queue managers.
4. The User Name Server must communicate with every broker in the broker domain. You must define transmission queues and channels to support two-way communications between the User Name Server and every broker except the one (if defined) that shares its queue manager.
5. The User Name Server's queue manager must have a listener to receive messages from other components that do not share its queue manager. You must create a listener for every protocol used for connections to the User Name Server. If you create a TCP/IP listener, you must also decide which port it must listen on.

MQSeries resources for the Control Center

The Control Center depends on a number of required MQSeries resources:

1. The fixed-name queues defined by the Configuration Manager (described above).
2. The Configuration Managers's listener (described above).
3. The server connection defined to the Configuration Manager's queue manager (described above). This is always defined as a TCP/IP connection and cannot be changed.
4. The client connection. This is dynamically created when you initialize the Control Center.

All necessary resources are defined and created for you, and you do not have to take any additional action to enable the Control Center.

MQSeries resources for client applications

SCADA applications do not require any local MQSeries resources. MQSeries Everyplace applications require an MQSeries Everyplace client on the local system.

A client application 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 or MQSeries Everyplace 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 or MQSeries Everyplace 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 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.

MQSeries applications can only get messages from queues owned by the queue manager to which it is connected, although MQSeries Everyplace supports a "remote get" function. 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 (either local, or using an MQSeries or MQSeries Everyplace 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).

MQSeries clusters

When you design the MQSeries network underlying your MQSeries Integrator broker domain, you must consider whether it is desirable to use clustering. Queue manager clusters bring two significant benefits:

- **Reduced system administration.** Clusters need fewer definitions to establish a network, and allow you to set up and change your network more quickly and easily.
- **Increased availability and workload balancing.** In addition to simpler administration, you can benefit by defining instances of the same queue on more than one queue manager, thus distributing workload through the cluster.

You can use clusters with MQSeries Integrator, but must consider the following points:

- **For broker, Configuration Manager, and User Name Server administration:**
If you define the queue managers that support your brokers, the Configuration Manager, and the User Name Server to a cluster, you can benefit from the simplified administration provided by MQSeries clusters. You might find this particularly relevant for the brokers in a collective, which must all have MQSeries interconnections.
- **For SYSTEM.BROKER queues:**
The SYSTEM.BROKER queues are defined for you when you create MQSeries Integrator components, and are not defined as cluster queues. You must not change this attribute.

- For message flow input queues:
 - If you define an input queue as a cluster queue, you must consider the implications for the order of messages or the segments of a segmented message. The implications are the same as they are for any MQSeries cluster queue. In particular, the application must ensure that if it is sending segmented messages then all segments will be processed by the same target queue, and therefore by the same instance of the message flow at the same broker.
- For message flow output queues:
 - MQSeries Integrator always specifies MQOO_BIND_AS_Q_DEF when it opens a queue for output. If you expect segmented messages to be put to an output queue, or want a series of messages to be handled by the same process, you must specify DEFBIND(OPEN) when you define that queue. This ensures that all segments of a single message, or all messages within a sequence, are put to the same target queue and are processed by the same instance of the receiving application.
 - If you create your own output nodes, you are recommended to specify MQOO_BIND_AS_Q_DEF when you open the output queue, and DEFBIND(OPEN) when you define the queue, if you need to guarantee message order, or ensure a single target for segmented messages.
- For publish/subscribe:
 - If the target queue for a publication is a cluster queue, you must deploy the publish/subscribe message flow to all the brokers on queue managers in the cluster. However, the cluster does not provide any of the failover function to the broker domain topology and function. If a broker to which a message is published, or a subscriber registers, is unavailable, the distribution of the publication or registration will not be taken over by another broker.
 - When a client registers a subscription with a broker running on a queue manager that is a member of a cluster, the broker forwards a proxy registration to its neighbors within the broker domain: the registration details are not advertised to other members of the cluster.

For a fuller understanding and discussion of clusters, and the implications of using cluster queues, see the *MQSeries Queue Manager Clusters* book.

Planning database resources

MQSeries Integrator requires a number of databases to contain control and configuration information. You must create these immediately after installation, because they are populated when you create your MQSeries Integrator components and resources.

Under normal circumstances, you do not need to be aware of the nature of these databases, or how the various components make use of, or update, the information they contain. However, it is useful to understand these basics:

- “Database requirements”
- “Databases and code pages” on page 119
- “Database locations” on page 120
- “Database backup and recovery” on page 120

Database requirements

MQSeries Integrator uses three sets of tables within databases. You can choose to create these in a single database if appropriate (subject to the installation options in Table 4 on page 95), or you can create a different database for each set.

The three sets of tables required by MQSeries Integrator are:

1. The configuration repository. This set of tables is managed by the Configuration Manager. It contains all configuration information for your broker domain. When you create and modify the resources in your broker domain using the Control Center (for example, if you create message flows), the changes you make are initially stored in your local system. You must deploy these changes for those updates to be processed by the Configuration Manager and reflected in the configuration repository.

The Configuration Manager is the only component that accesses this database. You can view and manage the data in this repository using the Control Center, which interacts with the Configuration Manager on your behalf.

You must create this database using DB2.

2. The broker database (also known as the broker's local persistent storage). This contains control information used by the brokers in maintaining their state and other internal information. The database contains one set of tables: the rows within each table include the broker name to ensure the integrity of the data.

When you make changes to the broker's environment, and deploy those changes, the Configuration Manager sends messages to the broker to update its local persistent store. For example, if you assign and deploy a new message flow to the broker, the data is updated.

You can create the broker database to hold this information using the following database products:

- IBM DB2 Universal Database
- Microsoft SQL Server (on Windows NT only)
- Oracle
- Sybase (not on HP-UX)

You can use a separate database for each broker if you choose. For more information about supported databases, see Table 5 on page 105.

3. The message repository. This set of tables is managed by the Configuration Manager. It contains all the message and message set definitions you have created using the Control Center and deployed in your broker domain. If you import externally defined message definitions using the Control Center, these are also stored in this repository.

You must create this database using DB2.

This repository does **not** contain definitions for messages created using the NEONFormatter user interface. For information on the database requirements for NEON message formats, see "Appendix A. Planning for migration and integration" on page 143.

MQSeries Integrator uses ODBC to connect to the message repository and the broker databases: ODBC drivers for DB2 and SQL Server are provided with the database products, ODBC drivers for Oracle and Sybase are provided by MQSeries Integrator. The Configuration Manager uses JDBC to connect to the configuration repository, see Figure 1 on page 10.

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

Databases

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.

ESQL 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 is running, and then by the database manager to the code page in which the database was created. You must ensure that these three code pages are compatible to avoid data conversion problems.

Database locations

The databases used by the product components can be located on any system that is accessible by the component that creates and maintains the tables within them.

You can set up a local database for each component if you choose, or you can set up a central database on a shared server, and set up remote access to that server for each and every system hosting a component that requires that access.

There are advantages and disadvantages to local and remote database usage. You must refer to the documentation supporting the database you are using for MQSeries Integrator to determine the best options for your specific environment.

Note: The User Name Server has no requirement for access to any of these databases.

Database backup and recovery

You must include the databases used by MQSeries Integrator in your regular database backup routines to ensure that the data critical to the operation of your broker domain is secure and recoverable in the event of system or disk storage failure.

For more details of the databases and tables, see the *MQSeries Integrator Installation Guide* for your product. For more information about recovery procedures, see the *MQSeries Integrator Administration Guide*.

Planning security

An important part of planning your broker domain is considering the security controls that are available, and the levels of security you want to implement for those controls.

MQSeries Integrator exploits MQSeries and the operating system facilities to control security of components and tasks:

- Topic-based security.
The MQSeries Integrator User Name Server interacts with the operating system security system to control user and group access to publications and subscriptions.
- Operational control of components.
MQSeries Integrator uses the operating system access control.
- Operational roles used in the Control Center.
MQSeries Integrator uses Windows NT access control. (The Control Center runs on Windows NT only.)

You must review the following information to understand the implications for your configuration. The following sections describe the controls that are available, and how they affect the operation of your broker domain:

- “Security and principals”
- “Operational security” on page 127
- “Control Center security” on page 128
- “Application security” on page 129
- “Message flow security” on page 129

You should also review the *Setting up security* section in the *MQSeries Integrator Administration Guide* for further information on implementing a security setup.

Security and principals

Security control of MQSeries Integrator components, resources, and tasks depends on the definition of users and groups of users (principals) to the security subsystem of the operating system (the Windows NT User Manager or the UNIX user/group database).

The MQSeries Integrator local groups are:

- **mqbrkrs**
- **mqbrasgn**
- **mqbrdevt**
- **mqbropts**
- **mqbrtpic**

The requirement for and creation of these groups differs on each operating system:

- On Windows NT, MQSeries Integrator creates all these groups on the system on which it is installed.
- On AIX systems, MQSeries Integrator creates the local group **mqbrkrs** on the system on which a component is installed.
- On HP-UX systems, you must create the local group **mqbrkrs** yourself, before you install MQSeries Integrator components.
- On Sun Solaris systems, you must create the local group **mqbrkrs** yourself, before you install MQSeries Integrator components.
- The groups other than **mqbrkrs** are used to control Control Center tasks and configuration repository access, and therefore **are not required on UNIX systems**.

You must assign users (or other groups) to the local groups to allow them to perform specific tasks. These assignments are summarized in Table 7 on page 125 and Table 6 on page 125.

Security

The local groups provide the following authorities:

- **mqbrkrs**
Users in this group are authorized as service user IDs for the brokers, the Configuration Manager, and the User Name Server. (Service user IDs have other authority requirements, detailed in Table 7 on page 125.)
- **mqbrdevt**
Users in this group are permitted to perform the following tasks in the Control Center:
 - Design messages, message sets, and message flows.
- **mqbrasgn**
Users in this group are permitted to perform the following tasks in the Control Center:
 - Manage execution groups within brokers.
 - View messages and message flows.
 - Assign message flows to execution groups.
 - Assign message sets to brokers.
- **mqbrops**
Users in this group are permitted to perform the following tasks in the Control Center:
 - Create brokers. (This creates a reference within the Control Center and the configuration repository to a broker you have created on the system on which it is to execute. This reference must have the same name as the physical broker).
 - Deploy, start, and stop message flows.
 - Start and stop trace activity on message flows.
 - Manage and deploy the broker domain topology, including collectives.
 - View the whole deployed system, including messages, message flows, and subscriptions.
 - Deploy topics.
 - View logs that report on the deployment activity.
- **mqbrtpic**
Users in this group are permitted to perform the following tasks in the Control Center:
 - Manage topics, and the access controls lists for the topic tree.
 - Deploy topics.
 - View the logs that report on that deployment activity.

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 are recommended to ensure that all the principals you define for MQSeries Integrator task authorizations are limited to eight characters 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 is used by MQSeries Integrator), but you must only use these longer names if you are sure you will not later include a UNIX system in your MQSeries Integrator network.

MQSeries Integrator has three primary sets of principals:

- Operational user IDs. These are defined as:
 - The user IDs that configure and manage MQSeries Integrator components.
 - The service user IDs under which the major components (broker, Configuration Manager, and User Name Server) operate. You must specify a service user ID when you create each component.

For more details about operational user IDs, see “Operational security” on page 127.

- Control Center user IDs. You must assign these to the MQSeries Integrator groups according to the set of tasks they will undertake. These groups are checked by the Configuration Manager, and must be defined to the security domain that you specify when you create the Configuration Manager. The user IDs you assign to these groups must be defined to the same security domain. For more details about Control Center user IDs, see “Control Center security” on page 128.
- Application user IDs. Users that participate in publish/subscribe must be assigned to groups that you create to control topic-based security. These groups and users must be defined to the security domain that you specify when you create the User Name Server. If you create the User Name Server on Windows NT, you are recommended to specify the same security domain as the one you specify when you create the Configuration Manager, but you are not forced to do this.

For more details about application user IDs, see “Application security” on page 129.

Using Windows NT security domains

MQSeries Integrator draws principals from either a Windows NT local account security domain, or a Windows NT primary domain, or a Windows NT trusted domain. For more information about Windows NT security domains, refer to 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.

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 you plan to use MQSeries Integrator within a primary or trusted security domain, global groups are created in your primary or trusted security domain controller during installation. The global groups, that mirror the local groups, are:

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

Security

(These groups are not used by MQSeries for any purpose, therefore the 12 character restriction does not apply.)

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 they appear in the list above. So, for example, from the Windows NT User Manager for domains select **User->New User** and enter **Domain mqbrasgn** in the username 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 on a primary domain controller, it is recommended that you install on the domain controller first so that the domain groups are created for you, and they will then be automatically added to the local groups created during install 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.
- 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 is authorized for the tasks this user ID is to perform, so that it is authorized globally.

For further details of how to implement security in the Windows NT environment, see the *MQSeries Integrator Administration Guide*.

Using UNIX security domains

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

Summary of authorizations

The authorizations required for the major tasks in both Windows NT and UNIX environments are summarized here. For further details, refer to the *MQSeries Integrator Administration Guide*.

Table 6 on page 125 summarizes the authorization and security requirements for some of the major tasks in the UNIX environments.

Table 7 summarizes the authorization and security requirements for some of the major tasks in the Windows NT environment.

Table 6. Summary of authorization in the UNIX environments

User is...	UNIX domain
Creating broker, User Name Server	<ul style="list-style-type: none"> Member of mqbrkrs The broker or User Name Server will run under the service user ID specified on the create command in most situations. This service user ID must be used to issue the create command itself. However when 'root' is used to issue the create command it can nominate any user to run the broker or User Name Server.
Installing	User must be a superuser
Uninstalling	User must be a superuser
Changing broker, User Name Server	User that the broker or User Name Server runs as (the service user ID), or 'root'
Deleting broker, User Name Server	User that the broker or User Name Server runs as (the service user ID), or 'root'
Starting and stopping broker, User Name Server	Member of mqbrkrs
Listing broker, User Name Server	Member of mqbrkrs
Changing, displaying, retrieving trace information	Member of mqbrkrs
Running User Name Server (service user ID)	<ul style="list-style-type: none"> Member of mqbrkrs The broker or User Name Server will run under the service user ID specified in the create command
Running Configuration Manager (service user ID)	Not applicable on UNIX systems
Running broker (not MQSeries trusted appl) (service user ID)	<ul style="list-style-type: none"> Member of mqbrkrs The broker or User Name Server will run under the service user ID specified in the create command
Running broker (MQSeries trusted appl) (service user ID)	<ul style="list-style-type: none"> Service user ID must be mqm mqm must be a member of mqbrkrs
Clearing, joining, listing MQSeries publish/subscribe brokers	Member of mqbrkrs
Running Control Center	Not applicable on UNIX systems
Running publish/subscribe applications	Any user, subject to MQSeries Integrator topic and MQSeries queue access control

Table 7 summarizes the security requirements for the major tasks. It illustrates what group memberships are 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.

Table 7. Summary of authorizations in the Windows NT environment

User is...	Local domain (SALONE)	Primary Domain (PRIMARY)	Trusted domain (TRUSTED)
Installing	• Member of Administrators	Not applicable.	Not applicable.
Uninstalling	• Member of Administrators	Not applicable.	Not applicable.

Security

Table 7. Summary of authorizations in the Windows NT environment (continued)

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
Starting broker, Configuration Manager, User Name Server	<ul style="list-style-type: none"> • Member of Administrators 	Not applicable.	Not applicable.
Running User Name Server (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 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\Domain mqm (see note 2)
Running broker (service user ID) (see note 5)	<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.

Notes:

- If you are running in a primary domain, you can also:
 - 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**.
- If you are running in a trusted domain, you can also:
 - 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**.
- 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.
- If a valid user ID is defined in the domain used by the Configuration Manager (for example, PRIMARY\User4) an identical user ID defined in a different domain (for example, DOMAIN2\User4) will be able to access the Control Center with the authorities of PRIMARY\User4.
- The broker can be run as an MQSeries trusted application. If it is, security requirements are changed. See the *MQSeries Integrator Administration Guide* for full details.

Operational security

When you create and activate your broker domain, there are two aspects of security that control the authorizations of users to perform these tasks:

- Configurational security, that controls the right of users to configure and manage MQSeries Integrator resources using the supplied commands.
- Run-time security, that controls the right of users to execute processes as service user IDs.

For a full definition of the commands that support these tasks and the authority required to invoke each one, see the *MQSeries Integrator Administration Guide*.

For a better understanding of MQSeries and database resource security for MQSeries Integrator components, see the *MQSeries Integrator Administration Guide*. For further information on securing MQSeries resources used by MQSeries Integrator components, refer to the section *Securing MQSeries resources* in the *MQSeries Integrator Administration Guide*. For further details of MQSeries security, refer to *MQSeries Planning* and *MQSeries System Administration*. For further details of database security, refer to the documentation for the database you are using.

Configurational security

MQSeries Integrator provides a set of configuration and operation commands that support system administration tasks that are not available through the Control Center.

The authorizations required by the user invoking these commands varies, depending on the task the command performs. These tasks are:

- Creating, changing, and deleting broker, the Configuration Manager, and the User Name Server
- Starting, stopping, listing, and tracing brokers, the Configuration Manager, and the User Name Server

The authorizations required for a subset of these commands is illustrated in Table 7 on page 125 and Table 6 on page 125. You can find a more complete summary of authorizations in the *MQSeries Integrator Administration Guide*.

Run-time security

When you start the broker, Configuration Manager, and User Name Server components on Windows NT, they are started up as Windows NT services running under the user ID that you specify as the service user ID when you create that component. When you start the broker or the User Name Server components on UNIX, they are started as normal processes running under the service user ID.

The authorizations required by these user IDs are illustrated in Table 7 on page 125 and Table 6 on page 125. You can find a more complete summary of authorizations in the *MQSeries Integrator Administration Guide*.

You must also use the MQSeries facilities to authorize the broker service user IDs to access the message flow input and output queues. Typically, this needs to be set for `get` and `inq` for input queues, and `put` and `setall` for output queues. See *MQSeries System Administration* for more information about setting queue access authorities.

Database security

The service user IDs for the brokers and the User Name Server must also be authorized to access databases:

- The Configuration Manager service user ID must be authorized for create and update tasks on the database in which both configuration and message repositories are defined. (This might be one or two databases: both must be DB2.)
- Each broker service user ID must be authorized for create and update tasks on the database that contains the broker internal tables.
- Each broker service user ID must also be authorized for the appropriate access for every database referenced and accessed by a message processing node in any deployed message flow.

Control Center security

All users can invoke the Control Center: there is no initial check when the program is invoked. However, in order to perform Control Center tasks, a user must choose the role they want to assume during this session. The role maps to a Windows NT group, and you must therefore define and configure the user and groups to meet your requirements, using the guidelines that are summarized in Table 7 on page 125. The domain that users are drawn from is set on the create command for the Configuration Manager – see the *MQSeries Integrator Administration Guide* for details of the command syntax. This configuration is independent of the implementation of topic-based security and the installation of a User Name Server.

To select a specific role, the user must choose one of the following from the *File->Preferences* dialog (User's role pane):

1. **Message flow and message set developer**
This role equates to the permissions of the **mqbrdevt** group members.
2. **Message flow and message set assigner**
This role equates to the permissions of the **mqbrasgn** group members.
3. **Operational domain controller**
This role equates to the permissions of the **mqbrops** group members.
4. **Topic security administrator**
This role equates to the permissions of the **mqbrtpic** group members.
5. **All roles**
This role combines all four roles, authorizing the user to perform all tasks.

The role determines what the user can view within the Control Center, and therefore limits the tasks that are available to that user. However, the authorization of that user to perform a given task is not checked until the request is processed by the Configuration Manager. To be able to perform any action, therefore, a user must be defined to the security domain specified when you created the Configuration Manager.

The Control Center passes the request in a message to the queue **SYSTEM.BROKER.CONFIG.QUEUE**: the Configuration Manager sends responses to the queue **SYSTEM.BROKER.CONFIG.REPLY** (both queues are defined to the Configuration Manager's queue manager).

All groups in the Configuration Manager's security domain have get and put authority to both queues. On receipt of the message, the Configuration Manager checks that the user ID is in the group that is authorized to complete the specific

task. Therefore you are recommended to encourage Control Center users to assume the role that corresponds to their authorization.

Additional authorizations required by users of the Control Center are summarized in Table 7 on page 125. For more details of the roles defined, and the facilities of the Control Center, see *MQSeries Integrator Using the Control Center*.

The IBMMQSI2 superuser

A *superuser* user ID is recognized by the Control Center and 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) 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 “Primitive node types” on page 50) 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 check out the existing primitive nodes, import the replacement nodes, and check them in to the configuration repository.

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

MQSeries authorizations

The Control Center connects to the Configuration Manager using an MQSeries client/server connection. For details of the security available for this connection, see the *MQSeries Clients* book (the chapter entitled “Setting up MQSeries client security”).

Application security

You need to consider application security in two areas:

- “Message flow security”.
- “Topic-based security” on page 130.

Message flow security

When you deploy a message flow on one or more brokers, applications can start to feed messages into the message flow by putting messages to the queue that is identified as the input queue. You set up the association between the input node and the queue by setting the queue name as a property of the node.

Similarly, applications access queues to receive messages placed on those queues by output or Publication nodes, when the message flow has completed processing for those messages.

The user IDs under which applications are executing must therefore be authorized to write to, or read from, the queues used by the message flow the applications are interacting with.

Security

You must also authorize every subscriber (that is, every application making a subscription registration) to put messages to the queue `SYSTEM.BROKER.CONTROL.QUEUE`. (This does not apply to MQSeries Everyplace applications.)

You must use the facilities provided by MQSeries to restrict which users are permitted to have “put” or “get” access to the queues. (This does not apply to MQSeries Everyplace applications.) For more details of applying security to MQSeries resources, see *MQSeries System Administration*.

Note that SCADA applications do not support message flow security. The `SCADAInput` node accepts all messages that the listener detects at the connected port.

Topic-based security

If you have applications that use the publish/subscribe services of a broker, you have the option of applying an additional level of security to the topics on which messages are published and subscribed. This additional security, known as topic-based security, is managed by the User Name Server. The User Name Server and the benefits of topic-based security are discussed in “Employing topic-based security” on page 112.

If you want to take advantage of topic-based security in your MQSeries Integrator broker domain, you must create, or update, your brokers and the Configuration Manager to recognize the User Name Server. You can identify the User Name Server to the brokers and the Configuration Manager by specifying the User Name Server’s queue manager name as the `-s` parameter on the commands **`mqsicreatebroker`**, **`mqsicreateconfigmgr`**, **`mqsichangebroker`**, and **`mqsichangeconfigmgr`**.

If you have already created the Configuration Manager and one or more brokers, you must stop them (using **`mqsistop`**) before you make these changes. You can then restart the Configuration Manager and the brokers, and start the User Name Server, using **`mqsistart`**. These steps are illustrated in the *MQSeries Integrator Installation Guide*.

When you have configured your broker domain components to incorporate the User Name Server, you can implement topic-based security by setting up Access Control Lists (ACLs) from the *Topics* view of the Control Center. ACLs are lists of principals, and are assigned to topics to control which principals can publish, subscribe, and request persistent delivery on those topics.

The principals you can include in an ACL are notified to the Control Center by the Configuration Manager, which requests the information from the User Name Server.

- If you created the User Name Server on Windows NT, it extracts principal information from the domain server of the security domain that you specified when you created the User Name Server. You must therefore define all users and groups required by your implementation of topic security to the security domain specified when you created the User Name Server.
- If you created the User Name Server on UNIX, it extracts principal information from the user/group database. You must therefore define all users and groups required by your implementation to the database accessed by the User Name Server.

When a publisher publishes a message to a broker, or a match for a published message for a particular subscriber is found, the broker checks its local copy of principal and ACL information to determine if the user request is authorized by an ACL for the specified topic.

After the broker has determined that a client has the authority to receive a particular publication, it makes a further check as to whether the client is authorized to request persistent delivery on this topic. If the client has requested persistent delivery, but is not authorized to do so, the broker does make the message available to the client, but non-persistently.

For more details on how to implement topic security, see *MQSeries Integrator Using the Control Center*, and for more detailed information on aspects of topic security, see “Topic-based security” on page 86.

Planning for data conversion

If you are using a network of systems that use different methods for storing numeric values, or you need to communicate between users who view data in different code pages, you need to consider how to implement data conversion.

- Numeric order

For numeric and encoding aspects, you must consider:

- Big Endian versus Little Endian

- Encoding values in MQSeries (field Encoding in the MQMD)

Encoding values are system specific. For example, Windows NT usually has an encoding of 546, hexadecimal value X'00000222'. The three final hexadecimal digits identify:

1. The float number format

This value can be 1 (IEEE format byte order normal), 2 (IEEE format byte order reversed) or 3 (System/390[®] format byte order normal).

2. The packed decimal number format

This value can be 1 (byte order normal) or 2 (byte order reversed).

3. The hexadecimal number format

This value can be 1 (byte order normal) or 2 (byte order reversed).

The bit order within a byte is never reversed. Byte order normal means that the least significant digit occupies the highest address.

Systems that process numbers in normal byte order are Big Endian (System/390, AS/400, and UNIX). Systems that process numbers in reversed byte order are Little Endian (mainly PCs).

For further details about numeric order, see Appendix D, Machine Encodings, in the *MQSeries Application Programming Reference*.

- Code page conversions

Code page conversion might be required for any of the following reasons:

- ASCII versus EBCDIC

- National languages

- Operating system specific code pages

For more information about code page support in MQSeries, see the *MQSeries Application Programming Reference* book.

Data conversion

When you use MQSeries Integrator, you can use the data conversion facilities of MQSeries, or MQSeries Integrator, or both.

- If you use MQSeries facilities, you must ensure that the input nodes within your message flows are set up for conversion. That is, you must set the Convert property to yes, and enter the required encoding and CCSIDs (coded character set identifiers).

Headers and message body are converted according to the MQMD values, and other header format names. You might have to set up data conversion exits to convert the body of your messages.

When you use MQSeries facilities, the whole message is converted to the specified encoding and CCSID.

For more detail about data conversion using MQSeries facilities, see Appendix F, Data Conversion, in the *MQSeries Application Programming Reference*.

- If you use MQSeries Integrator facilities, you must define your messages in the message repository (using the Control Center), or use self-defining messages. You can then use the Compute node to define encoding and CCSIDs. The predefined elements of the messages are converted according to their type and Custom Wire Format characteristics. You do not need MQSeries data conversion exits.
 - String data is converted according to the CCSID setting.
 - Integer and Float Extended Decimal types are converted according to the CCSID setting.
 - Integer and Float (other physical data types) are converted according to the Encoding setting.
 - Binary and Boolean data is not converted.

MQSeries Integrator can also convert those MQSeries headers for which parsers are provided (listed in “Default message parsers” on page 59).

When you use MQSeries Integrator facilities, the whole message is not converted to the specified encoding and CCSID: you can specify a different encoding, or CCSID, or both, in each header to perform a different conversion for the following part of the message. The encoding and CCSID in the last header therefore defines the values for the message body.

For an example of data conversion using MQSeries Integrator facilities, see the *MQSeries Integrator ESQL Reference* book.

Chapter 10. Managing your MQSeries Integrator network

This chapter provides the information you need to understand how to manage your MQSeries Integrator network, when you have planned and created it.

It covers the following topics:

- “Managing broker domain components”
- “Monitoring and analysis” on page 134

Managing broker domain 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:

1. Starting and stopping components
 - a. Start a component. You can use the command **mqsistart** to start up the instances of broker, Configuration Manager and User Name Server created by command. You must identify which component is to be started as the first parameter on the command. If appropriate, the associated queue manager is also started.
 - b. Stop a component. The command **mqsistop** 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.
2. Viewing and modifying components
 - a. List components or subcomponents available on a system. 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
If you specify a broker name as a parameter on the command, it returns a list of the broker’s execution groups. If you specify a broker name and identify an execution group, it returns the message flows within that execution group.
 - b. Change parameters of a component. If you need to update the parameters currently set for a component, use the **mqsichangebroker**, **mqsichangeconfigmgr**, or **mqsichangeusername** command. These set the newly specified value for each operand included on the command, and leave all others unchanged.

The change commands listed, like the create and delete commands discussed in “Planning MQSeries Integrator resources” on page 107, can be invoked using the Command Assistant.

For full details of all these commands, and the use of the Command Assistant, see the *MQSeries Integrator Administration Guide*.

For more information about managing the MQSeries resources associated with these MQSeries Integrator components, see *MQSeries System Administration*, *MQSeries Clients*, and *MQSeries Intercommunication*.

Managing the network

Managing application and business processes

The Control Center provides all the facilities for managing application and business processes. You can use the Control Center to:

- Define your broker domain, using the *Topology* view:
 - Add new brokers and collectives.
 - Remove a broker or collective.
 - Change the connectivity between brokers and collectives.
- Work with message flows, using the *Message Flow* and *Assignments* views:
 - Create new message flows using existing node types.
 - Assign message flows to execution groups in brokers.
 - Remove message flows from execution groups.
 - Solve message flow problems using the *Debugger*, an alternative screen under *Message Flow*.
- Organize your messages, from the *Messages* and *Assignments* views:
 - Define new message templates and message sets.
 - Update message templates.
 - Assign message sets to brokers.
 - Delete messages or message sets.
- Control your publish/subscribe network, in the *Topics* and *Subscriptions* views:
 - Define your topics.
 - Ensure authorizations are valid and complete.
 - Examine the subscriptions currently active.
- Manage your broker domain, using the *Topology* and *Operations* views:
 - Deploy assigned resources to brokers.
 - Check on the status of the latest resources deployed.
 - Check on broker status.
 - Switch on problem diagnosis tools.
- Monitor the success of deployments by viewing responses in the *Log* view.
- Access the NEON GUIs (rules, formatter, and tester) in the *Messages* view.

For further information, and details of how to complete the tasks outlined here, see *MQSeries Integrator Using the Control Center*.

Monitoring and analysis

When you have completed initial configuration and activation of your MQSeries Integrator network, you need to be sure that it is running as efficiently as possible, and that it is behaving as you want and expect.

The following topics describe how you can monitor your broker domain, and analyze its activities to achieve these goals:

- “Problem determination”
- “Managing workload and performance” on page 137
- “System management” on page 138

Problem determination

When your broker domain is configured and activated, you might want to view further information about how its operation is progressing, or you might need to detect why it is not behaving as you expect.

MQSeries Integrator provides commands and facilities that help you understand what is happening in your broker domain, and allow you to generate and review more information when you need to. It provides two major sources of information:

- Traces generated by components
- Messages generated by commands

These facilities are fully described in the *MQSeries Integrator Administration Guide*.

You can also use information generated by other products used by MQSeries Integrator (MQSeries, the databases, and ODBC) to help resolve problems.

To solve problems at the message flow level, see “Solving message flow problems with the Debugger” on page 54 where the Debugger facility of the Control Center is described.

Traces

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

Every level of additional tracing will affect the performance of your system.

Local error log messages: MQSeries Integrator writes some events to local logs supported by the operating system in which the errors are generated.

The logs used are:

- The UNIX syslog. You can extract readable syslog content to a file to view the entries recorded. For further information on how to use the syslog, see the *MQSeries Integrator Administration Guide*.
- The Windows NT event log (Application View).

You can access the records in this log using the Windows NT Event Viewer service.

Although you cannot select whether MQSeries Integrator takes the action to write these events to the Application event log, you can control the activity of the event log itself, at the operating system level.

Records in the local log are written by all product components to record significant events. For example, a record is written when you stop and start brokers, the Configuration Manager, or the User Name Server. If an interaction with a database fails, this is also recorded. In some situations (for example, when you start the Configuration Manager), you are advised to view this information to ensure that the action you have taken completes successfully. You can also use the contents of this log for reference and error information when you are developing and running message flows.

The local logs are of interest to your local operations department because they provide initial information about failures and unexpected behaviors. The information in these logs might also be requested to support the service trace information generated at the request of your IBM Support Center.

Optional traces: Optional traces are provided by MQSeries Integrator:

- User tracing. You can trace brokers, execution groups, and message flows. You can use this facility when you are looking at problems or unexpected behavior exhibited by your message flows.

Monitoring the network

- **Service tracing.** You can activate a more comprehensive broker trace, and start tracing for the Configuration Manager, User Name Server, and Control Center, and for the command utility programs (for example, **mqscreatebroker**). You are recommended to use these traces only when directed to do so by your IBM Support Center. If you encounter a problem that you have to report to IBM for resolution, you are likely to be given instructions to create and access the service logs to provide supporting information.

Controlling user trace: Four commands are provided to activate optional traces, and to access and review the contents of the logs generated. These commands are:

- **mqsichangetrace:** to activate and deactivate trace, or to change trace settings (for example, trace logfile size).
- **mqsireporttrace:** to report the current trace settings.
- **mqsireadlog:** to access and retrieve log file contents in XML format.
- **mqsiformatlog:** to format an XML log file (generated by **mqsireadlog**) for easier interpretation.

For details of these commands, their usage, and other problem determination techniques, see the *MQSeries Integrator Administration Guide*.

The Control Center also has an interface to start and stop tracing for execution groups and message flows on specific brokers. You can use this method as an alternative to the commands provided.

For example, if you do not have command line access on the system on which the broker is running, the Control Center communicates with the remote broker to achieve the same actions. The options available through this interface are a subset of the support provided by the commands invoked on the command line on the broker's local system. However, you must have local access to be able to extract the trace output from the system on which it is generated.

For details of trace options in the Control Center see *MQSeries Integrator Using the Control Center*.

Tracing message flows: When you create a message flow, you can include a **Trace** node. You can use the trace node to record additional information about the message being processed. The information generated is written to the standard trace logs or to a separate file.

Monitoring Control Center deployment: The Control Center displays additional activity records in its *Log* view. These records provide information about the success or failure of the actions taken by the user of the Control Center. For example, if you deploy a message flow to a broker, a series of records are displayed for you to check the progress of that deployment.

For more details about these options, see *MQSeries Integrator Using the Control Center*.

Messages

When you invoke any of the commands that MQSeries Integrator supplies (for example, **mqscreatebroker** or **mqsistart**), responses are returned in the form of messages. These messages have the prefix BIP and a numeric value. Some messages are also generated by the installation and un-installation programs, and by the Control Center. You can check the full meaning of these messages, and the actions you can take, in the *MQSeries Integrator Messages* book.

For more information about MQSeries Integrator messages, see the *MQSeries Integrator Administration Guide*.

Information available from other sources

In addition to MQSeries Integrator trace, you can refer to:

- The database messages and logs
You can determine additional information about MQSeries Integrator's use of databases from the messages issued by the database products, and from log information generated by database trace activity.
- MQSeries messages and logs
You can access trace information generated by MQSeries in its log files. You can also gain further information from MQSeries messages when these are returned by MQSeries Integrator activities.
- MQSeries events
You can control the generation of event messages by MQSeries queue managers in response to specific conditions. For example, you can request an event is generated when a queue becomes full.
- ODBC traces
You can initiate trace for ODBC activity. On Windows NT, you must select the Trace tab of the ODBC function available in the Control Panel. On UNIX, you must modify the `.odbc.ini` file to activate the trace.

You can find more information about these additional sources in the *MQSeries Integrator Administration Guide*.

Managing workload and performance

When you have configured and activated your broker domain, its performance will depend very heavily on the level of activity it is supporting.

There are several areas you can consider in making best use of the resources you have defined. These are:

- "Using MQSeries trusted applications"
- "Tuning message flow performance" on page 138

Using MQSeries trusted applications

When you create the broker using the `mqsicreatebroker` command, you can configure it to run as an "MQSeries trusted application". 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.

This does **not** affect the operation of any MQSeries channel agents or listeners. If you want to run these as trusted applications, you must follow the guidance in *MQSeries Intercommunication*, in the section entitled "Running channels and listeners as trusted applications".

You must be aware that MQSeries places a number of restrictions on the operation of a trusted MQSeries application. If you want to enable a broker as a trusted application, you must first review these restrictions for applicability to your own environment. They are documented in the *MQSeries Application Programming Guide*, in the section entitled "Connecting to a queue manager using the MQCONN call".

Monitoring the network

You must also consider:

- MQSeries trusted applications must run with an effective user ID and group ID of **mqm**. You must therefore have created the broker to run under this user ID.
- You must be careful if you are deploying plug-in nodes, or parsers, or both. Because the trusted application (the broker) is running in the same operating system process as the queue manager, an ill-behaved plug-in could compromise the integrity of the queue manager.

You are therefore recommended to develop all plug-in components with full consideration of the restrictions. You are also advised to test plug-in components in a non-trusted environment before deploying them in a trusted broker.

Tuning message flow performance

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

For more details of these properties, see *MQSeries Integrator Using the Control Center* and the Control Center online help.

System management

MQSeries Integrator uses architected messages to publish events related to the status, and change in status, of the brokers. These messages are published using the reserved topic root `$$SYS` in code page 1208.

The format of these messages, constructed in XML, is detailed in the *MQSeries Integrator Administration Guide*. The messages cover configuration changes, state changes, error notifications, and detailed subscription and topic information (for example, a subscription registration).

You can develop or buy in system management adapters or customized administrative applications. These subscribe to the system management topics generated by MQSeries Integrator to receive information on the broker domain activity.

Chapter 11. Enhancing your broker domain

This chapter discusses advanced options that extend the basic functions of the broker and other components, and hence allow you to enhance your broker domain.

Details of implementing the advanced functions discussed here are provided in the *MQSeries Integrator Programming Guide*.

The topics covered are:

- “General guidance for writing plug-ins”
- “Writing your own message processing node types”
- “Writing your own parsers” on page 140

General guidance for writing plug-ins

MQSeries Integrator provides support for you to extend your system by writing components which plug in to the framework provided by the product. The “plug-ins” supported are message processing node types and message parsers. The guidelines you need to understand and follow are mostly the same for both plug-in types. The common considerations are discussed here, followed by sections which indicate the special considerations for each plug-in type in turn.

A plug-in, or broker extension, must be written in the C programming language. It must be distributed as a shared library. The file type of the shared library must be set to the value required by the operating system on which it will run (these values are given in the *MQSeries Integrator Programming Guide*).

If you plan to program using either of the supplied plug-in interfaces, you must install the “Samples and SDK” optional component on at least one system. The SDK provides the required header files and contains samples that you can modify to your own requirements.

You can use your new node types or parsers on more than one operating system, if you make them platform independent. You can achieve this by using the ANSI standard C programming language, and avoiding any use of operating system dependent functions.

Refer to the *MQSeries Integrator Programming Guide* for further information on:

- The programming interface for both plug-in types, including all the calls and parameters
- How to create the icon, *signature*, and help files for the message processing node type using the Plug-in SmartGuide in the Control Center
- How to build the required components for each interface
- The content of the supplied sample files

Writing your own message processing node types

You can create your own message processing node types to complement the primitive node types provided by MQSeries Integrator.

Writing message processing nodes

You might want to do this, for example:

- If your messages need additional transformation not provided by the primitive nodes. For example, you might need a currency converter node.
- If you want to write messages into a flat file on the local system for later processing by another application or utility program.

You can use your new node types with existing primitive node types to create message flows to achieve the processing your messages require.

MQSeries Integrator does not support plug-in input nodes. The **MQInput** node must be used as the input node (or nodes) for most message flows, the **MQeInput** node for MQSeries Everyplace message flows, and the **SCADAInput** node for SCADA message flows. You can augment the function provided by every other node.

Writing your own parsers

Message parsers are invoked by the processes within a broker to interpret the bit-stream forming a message and its header (or headers). MQSeries Integrator provides a number of message parsers that handle a wide range of messages and headers, and cover the majority of formats that are expected to be processed within a broker domain. These default parsers are described in “Message parsers” on page 59.

However, you might need to use messages that are not covered by these default parsers. To allow for this possibility, MQSeries Integrator provides an external interface that enables you to supply your own parsers. These can be invoked by the broker processes whenever a message of this new type is received, and can work in the broker alongside the default parsers.

When you define a message, one of its attributes is the message domain. This is the value that tells the broker which parser must be invoked to interpret the bit-stream.

Part 5. Appendixes

Appendix A. Planning for migration and integration

This chapter helps you plan for migration to MQSeries Integrator Version 2.0.2 from compatible IBM offerings. It gives you an overview of the tasks involved, and provides references to the detailed information you need to complete these tasks.

Refer to the section giving details for your existing product:

- “MQSeries Integrator Version 1”
- “MQSeries Publish/Subscribe” on page 147

If you are migrating from MQSeries Integrator Version 2.0 or Version 2.0.1 to Version 2.0.2, see “Release to release migration” on page 31.

MQSeries Integrator Version 1

Migration to MQSeries Integrator Version 2.0.2 is supported from the following products:

- MQSeries Integrator Version 1.0²
- MQSeries Integrator Version 1.1

Migration information can be found in the *MQSeries Integrator Administration Guide*.

The tasks you must plan for fall into two broad categories:

- “Installation”:

This identifies tasks you must complete before and immediately after installation of MQSeries Integrator Version 2.0.2.

These tasks are fully described in the *MQSeries Integrator Installation Guide* for your product.

- “Run-time” on page 144:

This identifies tasks you must complete during normal operation to enable the continued use of your Version 1 resources.

These tasks are fully described in either the *MQSeries Integrator Administration Guide*, or in *MQSeries Integrator Using the Control Center*.

Installation

You should consider these areas when you plan the installation of MQSeries Integrator Version 2.0.2:

- “Backing up configuration files”
- “Preserving your MQSeries Integrator Version 1 rules and formats” on page 144
- “Uninstallation of MQSeries Integrator Version 1” on page 144

Backing up configuration files

MQSeries Integrator Version 1 uses a number of configuration files to control various aspects of its operation. Some of these files are reused by MQSeries Integrator Version 2.0.2, and can be updated in some circumstances.

2. It is also possible to upgrade from NEON’s MQIntegrator product. The tasks required are identical to those specified for migrating from MQSeries Integrator Version 1.0. However, the presence of this product is not detected by the MQSeries Integrator Version 2.0.2 installation program.

MQSeries Integrator Version 1

You are therefore advised, but not forced, to backup your MQSeries Integrator Version 1 configuration files.

For details of configuration files, see the *MQSeries Integrator Installation Guide* for your product.

Preserving your MQSeries Integrator Version 1 rules and formats

All the rules and formats you have defined in MQSeries Integrator Version 1 can be reused by MQSeries Integrator Version 2.0.2. The message processing nodes NEONRulesEvaluation, NEONTransform, and NEONMap (that supersede the NEONRules and NEONFormatter nodes) provide the NEONRules and NEONFormatter function in MQSeries Integrator Version 2.0.2 and can reproduce the MQSeries Integrator Version 1.1 behavior.

If you have rules and formats defined by any previous version of MQSeries Integrator (including Version 1.1 and Version 2.0.1) that you want to reuse, you must export this data from your previous version (using the tools supplied with that version) then import it into MQSeries Integrator Version 2.0.2 (using the tools supplied with V2.0.2). This will convert the data into a format suitable for use with MQSeries Integrator V2.0.2.

Uninstallation of MQSeries Integrator Version 1

You must remove all previous versions of MQSeries Integrator (versions 1.0, 1.1, 2.0 and 2.0.1) before you install Version 2.0.2.

Run-time

You must consider these operational aspects when planning your migration from MQSeries Integrator Version 1 to MQSeries Integrator Version 2.0.2. These are:

- “NEON rules and formats”
- “Setting up a message flow which emulates the functionality of the Version 1 Rules engine” on page 145

NEON rules and formats

MQSeries Integrator Version 2.0.2 provides message parsers that interpret the NEON message formats, and these are used by any message processing node that detects a NEON message has been received. Therefore interpretation of messages in NEON formats can be provided to any message processing node, not just the NEONRulesEvaluation, NEONTransform, and NEONMap nodes (which supersede the NEONRules and NEONFormatter nodes).

Update of message content is provided by the NEONTransform and NEONMap nodes (which supersede the NEONFormatter node) and the Compute node.

Access to the database containing existing definitions is defined by the *neonreg.dat* configuration file (it was *MQSIruleng.mpf* in MQSeries Integrator Version 1). The MQSeries Integrator Version 2.0.2 code accesses the configuration file by interrogating the environment variable NN_CONFIG_FILE_PATH. You must set this variable to point to the default file supplied by MQSeries Integrator Version 2.0.2.

Note that:

- You can modify the version given in the examples directory
- You can modify the version created by the NEONRules and NEONFormatter Support component when the setup program *inst_db* is run
- On Windows NT you must restart your system to enable these changes

You can encrypt the *neonreg.dat* file to protect the password. See the *NEONRules and NEONFormatter Support for MQSeries Integrator System Management Guide* for more details.

You must be aware that the **NEONRules** and **NEONFormats** defined by the NEONRules and NEONFormatter GUI tools are not distributed automatically to all brokers that need them, as those defined by the MQSeries Integrator Version 2.0.2 Control Center are. You must configure your system so that every broker running a message flow that accesses your **NEONRules** and **NEONFormats** has access to the database that contains these definitions. It should also be noted that only one rules and formatter database can be accessed per machine. This means that if two brokers are installed in the same machine they must both access the same rules and formatter database — there can only be one *neonreg.dat* on a particular machine.

MQSeries Integrator Version 2.0.2 provides full support for MQRFH headers, as well as MQRFH2 headers. If you are developing new applications, you are recommended to use the new MQRFH2, which offers superior function.

For further details of these tasks, see the *MQSeries Integrator Administration Guide*.

Enhancing existing rules and formats: MQSeries Integrator Version 2.0.2 provides support for you to continue to develop new and modify existing rules and formats. It does this by installing the **NEONRules** and **NEONFormats** graphical utility programs.

You can therefore continue to maintain existing data, and can add new definitions to your existing set. Refer to the *NEONRules and NEONFormatter Support for MQSeries Integrator User's Guide* for information on using these user interfaces.

The NEON Formats are represented in the Control Center under the Message Sets tab. This is so that NEON Formats can be used as Inputs or Outputs for the purpose of defining field mappings in the Compute or Database properties panels. Although the NEON Formats may be viewed from the Message Sets tab, they may not be edited, and you are recommended to use the NEONFormatter GUI (which may be launched from the Control Center) for viewing as well as editing existing NEON Formats or creating new ones.

Setting up a message flow which emulates the functionality of the Version 1 Rules engine

To provide continued support for your existing MQSeries Integrator Version 1 applications, you must deploy an MQSeries Integrator Version 2.0.2 message flow that emulates the function of the MQSeries Integrator Version 1 product daemon.

Default MQSeries Integrator Version 1 message flows are provided for your use. The message flow for Version 2.0.2 includes the **NEONRulesEvaluation** message processing node (which supersedes the **NEONRules** node)³. The default message flow for Version 2.0.1 and earlier includes the **NEONRules** message processing node. In addition, they include:

- An **MQInput** node, that reads input messages from an input queue and delivers them to the **NEONRulesEvaluation** or the **NEONRules** node.

3. The message flows only emulate the function of an unmodified MQSeries Integrator Version 1.1 daemon. If you have modified the daemon in your MQSeries Integrator Version 1.1 product, these message flows will not provide identical function. You must also modify these message flows to recreate the modifications you have made to the daemon.

MQSeries Integrator Version 1

- Three **MQOutput** nodes, that handle messages for *Output*, *Failure*, and *NoHit* processing.

Before you deploy either default message flow, you must edit the node properties of the MQInput and MQOutput nodes to align with your MQSeries Integrator Version 1 use of queues.

You must also ensure that any broker to which you assign this message flow is able to access the database in which your formats and rules are defined.

You can also use NEON format messages with other message processing nodes within a message flow. You must define a message flow with the message processing nodes providing the function your message processing requires. The nodes detect the presence of a NEON header and invoke the NEON parser to interpret the message.

If you want to change the content of the message, you must use the NEONTransform node (which supersedes the NEONFormatter node) or the NEONMap node. You can also use the Compute node to write NEON format messages.

You can also modify the default message flow supplied to include additional function. For example, you can cause all messages to be stored in a warehouse by adding a **Warehouse** node into the message flow prior to the NEONRulesEvaluation node.

If you include the NEONRulesEvaluation message processing node in your message flow, you can continue to use existing subscriptions with that message flow. You can also continue to use the NEONRules user interface to modify existing and create new subscriptions. Or you could replace the node handling messages destined for the *NoHit* queue with one that updates the message and returns it to the originator.

MQSeries Integrator Using the Control Center provides details on how to define, modify, assign, and deploy message flows.

You can increase the throughput of NEON messages by assigning the same message flow to multiple execution groups on a single broker, or to multiple brokers, or both. MQSeries Integrator Version 2.0.2 implements synchronization controls around the NEON message processing nodes to ensure the integrity of the multiple flows.

User exits: You can continue to use your existing MQSeries Integrator Version 1.1 user exits with MQSeries Integrator Version 2.0.2 message processing nodes. The source of your exit programs can be used unchanged. However, you must rebuild them to use the new dynamic link interface that is required by the MQSeries Integrator Version 2.0.2 modules that provide the MQSeries Integrator Version 2.0.2 function. For further information see the *NEONRules and NEONFormatter Support for MQSeries Integrator User's Guide* and the *MQSeries Integrator Administration Guide*.

If you are migrating from MQSeries Integrator Version 1.0, your user exits must be modified to be compatible with MQSeries Integrator Version 1.1 before they can be used with MQSeries Integrator Version 2.0.2.

MQSeries Publish/Subscribe

MQSeries Publish/Subscribe is supported software that provides publish/subscribe application support for MQSeries applications. It is available from the IBM Web site, and can be installed on several MQSeries Messaging products servers, including:

- AIX
- HP-UX
- Sun Solaris
- Windows NT

You can find latest details of this product, including how to download the product code, on the following web site:

<http://www.ibm.com/software/ts/mqseries/txppacs/ma0c.html>

If you plan to create a heterogeneous network including MQSeries Integrator brokers and MQSeries Publish/Subscribe brokers, you must ensure your systems have the appropriate level of MQSeries to run your brokers.

- 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 CSD3 on Windows NT, or CSD1 on AIX and Sun Solaris platforms.

Note: CSD6 is supplied with the MQSeries Integrator Version 2.0.2 package: you can apply this service to provide the MQSeries upgrade required for the MQSeries Publish/Subscribe brokers.

- MQSeries Version 5.2. This is the minimum level for the HP-UX platform.

If you do not upgrade MQSeries to these specified service levels, 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.

Scenarios for migration and integration

If you are already using MQSeries Publish/Subscribe, you can take advantage of the improved message processing function provided by MQSeries Integrator by integrating your two networks of brokers and creating a heterogeneous network.

You can also migrate individual MQSeries Publish/Subscribe brokers to create replacement MQSeries Integrator brokers, with support for their client applications intact.

These two possibilities offer you a number of advantages:

- Publications from within the MQSeries Publish/Subscribe network can be targeted by MQSeries Integrator subscribers. This includes messages originating in environments not yet supported by MQSeries Integrator.
- Message flows can be created and deployed on MQSeries Integrator brokers to:
 - Analyze the information that is flowing around your enterprise.
 - Invoke additional business logic dependent upon the content of the publications.

MQSeries Publish/Subscribe

- Consolidate the information within your enterprise in the form of new publications, that can then be republished as a series of additional topics available to both MQSeries Integrator and MQSeries Publish/Subscribe clients.

There are three possible scenarios for exploiting the two networks:

1. You can choose to have two independent broker networks, and therefore have two separate broker domains for publish/subscribe applications. This scenario is described in “Scenario 1: running two independent broker networks” on page 160.
2. You can connect the two networks to allow publications and subscriptions to flow throughout the integrated network. Further details are provided in “Scenario 2: creating and operating a heterogeneous network” on page 161.
3. You can selectively and gradually migrate individual brokers from MQSeries Publish/Subscribe to MQSeries Integrator Version 2.0.2. For more guidance on this option, see “Scenario 3: migrating MQSeries Publish/Subscribe brokers” on page 162.

Before you can make this choice, and create your migration plan, you must be aware of the differences in the two products, described in “Product differences”.

Product differences

There are differences in the support provided by the two products that you must consider when you plan how you will integrate your two networks. These are discussed in the following sections:

- “Message formats”
- “Streams” on page 151
- “Stream authority” on page 154
- “Topics” on page 156
- “Wildcards” on page 156
- “Default topic routing” on page 157
- “Retained publications” on page 157
- “Metatopics” on page 157
- “Subscription points” on page 158
- “Content-based filtering” on page 159
- “Throughput” on page 159

Message formats

You are recommended to use the MQRFH2 header for new client applications developed for the MQSeries Integrator broker. These applications can then access all of the function provided by MQSeries Integrator.

Existing MQSeries Publish/Subscribe applications using the MQRFH header are also supported by MQSeries Integrator, but function is limited to that provided by MQSeries Publish/Subscribe. MQSeries Publish/Subscribe does not support the MQRFH2 format. Clients connected to MQSeries Publish/Subscribe brokers must use the MQRFH format.

However, client applications that need to communicate with one another using publish/subscribe can do so regardless of the format of the messages they are using: MQSeries Integrator provides automatic conversion to ensure the subscriber receives the message in the desired format.

Table 8 shows the mapping between equivalent fields in the MQRFH and MQRFH2 headers.

Table 8. MQRFH and MQRFH2 mapping

MQRFH field name	MQRFH2 field name
MQPSCCommand	Command
MQPSDelOpts	DelOpt
MQSPSubOpts	PubOpt
MQSPSubTime	PubTime
MQPSQMgrName	QMgrName
MQPSQName	QName
MQPSRegOpts	RegOpt
MQPSSeqNum	SeqNum
MQPSTopic	Topic

All the MQRFH2 fields shown are contained in a <psc> folder.

Field names that are not included in Table 8 do not have a common meaning, or are only valid in one header or the other. Field names which are not recognized, or not appropriate to the other format, are not copied. For example, the following name-value area of an MQRFH:

```
MQPSCCommand Publish
MQSPSubOpts RetainPub
MQPSStreamName SAMPLE.BROKER.RESULTS.STREAM
MQPSTopic "Sport/Soccer/State/LatestScore/Team1 Team2"
```

is converted to this MQRFH2 folder:

```
<psc>
<Command>Publish</Command>
<PubOpt>RetainPub</PubOpt>
<Topic>Sport/Soccer/State/LatestScore/Team1 Team2</Topic>
</psc>
```

Using these mapping rules, MQSeries Integrator ensures that MQRFH2 publications can still be received by MQRFH subscribers, and MQRFH publications can be received by MQRFH2 subscribers.

Content-filters can also be specified by MQRFH2 subscribers even if the topic that they are subscribing to is one published in MQRFH format by an MQSeries Publish/Subscribe client, although there is some limit to compatibility. For more information, see “Content-based filtering” on page 159.

Table 9 on page 150 summarizes the valid options for clients using the different message formats.

MQSeries Publish/Subscribe

Table 9. Summary of message option support

Message	Option Name	Option Value	Support	
All requests (client to broker)	MQPSCCommand	DeletePub	yes	
		DeregPub	yes ¹	
		DeregSub	yes	
		Publish	yes	
		RegPub	yes ¹	
		RegSub	yes	
		ReqUpdate	yes	
		MQMD.Format	MQFMT_PCF MQFMT_RF_HEADER	no yes
		MQMD.Report	MQRO_PAN MQRO_NAN	yes yes
		MQMD.MsgType	MQMT_REQUEST MQMT_DATAGRAM	yes yes
MQMD.MsgId		yes		
MQMD.CorrelId		yes ⁴		
MQMD.ReplyToQ		yes		
MQMD.ReplyToQMgr		yes		
MQPSSStreamName		prefixed on topic ³		
MQPSTopic		yes		
All requests except Delete Publication	MQPSQMgrName		yes	
	MQPSQName		yes	
	MQPSRegOpts	CorrelAsId	yes	
Delete Publication	MQPSDelOpts	Local	yes ⁵	
Deregister Publisher ¹	MQPSRegOpts	DeregAll	yes	
Deregister Subscriber	MQPSRegOpts	DeregAll	yes	
Publish	MQMD fields	As specified by MQPS ²	yes	
		MQPSRegOpts	Anon	yes ⁷
			Local	yes ⁵
	DirectReq		yes ¹	
	MQPSPubOpts	NoReg	yes ¹	
		RetainPub	yes (set by publisher)	
		IsRetainedPub	yes (set by broker)	
		OtherSubsOnly	yes	
	MQPSPubTime		yes	
MQPSSeqNum		yes		
MQPSStringData ¹		yes		
MQPSIntData ¹		yes		
Register Publisher ¹	MQPSRegOpts	Anon	yes ⁷	
		Local	yes ⁵	
		DirectReq	yes ¹	
Register Subscriber	MQPSRegOpts	Anon	yes ⁷	
		Local	yes ⁵	
		NewPubsOnly	yes	
		PubOnReqOnly	yes	
		InclStreamName	no ³	
		InformIfRet	yes	

Table 9. Summary of message option support (continued)

Message	Option Name	Option Value	Support
All responses (broker to client)	MQPSCompCode		new values added ⁶
	MQPSReason		new values added ⁶
	MQPSReasonText		values may added ⁶
	MQPSCommand		command to which this is a response
Notes: <ol style="list-style-type: none"> 1. This option is supported for migration purposes. 2. MQPS is MQSeries Publish/Subscribe. 3. The stream name parameter is effectively prefixed on the topic. The stream name can be deduced from the queue name if the property <i>implicitStreamNaming</i> of the Publication node is set (see “Streams”). 4. The client identity is determined as the concatenation of the queue manager name, the queue name, and optionally the correlation id (when the correlation ID as identity option is set). The application identifier is thus “MQPSQMgrName:MQPSQName[:correlId]”. The default values specified by MQSeries Publish/Subscribe are used if these values are not present in a message. 5. The behavior of this option differs. See the <i>MQSeries Integrator Programming Guide</i> for an explanation of this option. 6. New values have been added. See the <i>MQSeries Integrator Programming Guide</i> for details. 7. Ignored by MQSeries Integrator Version 2.0.2. 			

Special rules also apply for MQRFH2 subscribers if the information is being published on an MQSeries Publish/Subscribe stream other than the default, SYSTEM.BROKER.DEFAULT.STREAM. These rules are summarized in Table 10 on page 152.

Streams

MQSeries Publish/Subscribe primarily use streams as a means to partition the topic name space. Sets of related topics could be grouped together into separate streams allowing different security controls to be applied, and the publishing workload of the broker to be better balanced.

MQSeries Integrator provides more flexible controls to achieve both of these behaviors. The concept of a stream is only supported for MQRFH application compatibility.

Stream names now only have the partitioning effect on the topic name space. MQSeries Integrator provides more flexible security controls that allow authorization to be applied to an individual topic level. Also, the publishing workload of the broker can be more easily controlled by creating additional instances of publication message flows either serving the same or different input queues.

MQSeries Integrator still allows MQRFH client applications to specify an MQPSSStreamName command parameter in their subscriptions and publications. However, the stream name is only used to modify the topic in order to preserve the partitioning characteristic of MQSeries Publish/Subscribe.

When the stream-name associated with a message is set to something other than SYSTEM.BROKER.DEFAULT.STREAM, the message is processed as if the topic (or topics) mentioned within the message had been prefixed with the string “\$SYS/STREAM/<streamname>/”. That is, a subscription to Topic1 that specifies a stream name of StreamX is processed as if the subscription had been made to topic “\$SYS/STREAM/StreamX/Topic1”.

MQRFH2 publishing and subscribing applications can still target stream-related topics, even though they themselves are not allowed to specify a stream name in

MQSeries Publish/Subscribe

the messages they send to the MQSeries Integrator broker. To do this, they must prefix the topics with the appropriate stream prefix.

For example, an MQRFH2 subscriber must specify topic “\$SYS/STREAM/STOCK.STREAM/IBM/Latest” in order to subscribe to topic “IBM/Latest” that is published on stream STOCK.STREAM within the MQSeries Publish/Subscribe network.

MQSeries Publish/Subscribe only allows a stream-related publication to be sent to a queue with the same name as the stream. However, MQSeries Integrator allows publishing clients to send their publications to any input queue in a message flow. MQRFH applications choosing explicitly to specify a stream name parameter within a publication can send it to any publication queue being serviced by the MQSeries Integrator broker. The queue no longer needs to have the same name as the stream. However, this behavior could affect the order in which publications are received, and you must consider the importance of ordering for your applications. For more details about ordering, see “Throughput” on page 159.

Each Publication node has an *Implicit Stream Naming* property that defaults to *true*. This default option results in behavior identical to that in MQSeries Publish/Subscribe when an MQRFH publication does not contain an explicit stream name. If this property is *false*, and the publication contains no explicit stream name, SYSTEM.BROKER.DEFAULT.STREAM is assumed.

Table 10 summarizes the options available to both MQRFH and MQRFH2 client applications publishing messages to either the default stream, or a specific MQSeries Publish/Subscribe stream. An example stream name of *StreamX* is used to illustrate the options.

Table 10. MQRFH and MQRFH2 client application options

	MQRFH publisher		MQRFH2 publisher	
	default stream	StreamX	default stream	StreamX
MQRFH subscriber	S1,P1	S2,P2	S1,P3	S2,P4
MQRFH2 subscriber	S3,P1	S4,P2	S3,P3	S4,P4

Subscriber notes:

- S1** Subscriber subscribes either without a stream name or with stream name “SYSTEM.BROKER.DEFAULT.STREAM”.
- S2** Subscriber subscribes with stream name “StreamX”.
- S3** Subscriber subscribes on topic without adding “\$SYS/STREAM/<streamname>/”.
- S4** Subscriber subscribes prefixes topic with “\$SYS/STREAM/StreamX/”.

Publisher notes:

- P1** Publisher publishes on any queue specifying stream name “SYSTEM.BROKER.DEFAULT.STREAM”. or publishes without specifying a stream name on any queue with the *Implicit Stream Naming* property set to false.
- P2** Publisher publishes on any queue specifying stream name “StreamX”, or publishes without specifying a stream name on queue “StreamX” with the *Implicit Stream Naming* property set to true.
- P3** Publisher publishes on any queue without adding the prefix “\$SYS/STREAM/<Stream>/” to the topic.
- P4** Publisher publishes on any queue and adds the prefix “\$SYS/STREAM/StreamX/” to the topic.

MQSeries Publish/Subscribe

Note: The “`$$SYS/STREAM/<streamname>/`” prefix is removed from all topics in an MQRFH2 publication when it is delivered to an MQRFH subscriber.

Streams and neighbor brokers: In an MQSeries Publish/Subscribe network it is not mandatory for all brokers to support the same set of streams as its neighbors. If a broker does not support a stream that is supported by one of its neighboring brokers, publications associated with the uncommon stream are simply not available to clients at that broker.

When an MQSeries Integrator broker joins the network, it acts as if it supports all the streams of its neighboring MQSeries Publish/Subscribe broker. This means that clients of the MQSeries Integrator broker are able to target publications for any stream supported by any of its MQSeries Publish/Subscribe neighbors.

However, to make these publications available, you must define the stream queues, and define and deploy the message flows that will support them, to the MQSeries Integrator broker.

The effects of adding an MQSeries Integrator broker into a multi-stream MQSeries Publish/Subscribe environment are illustrated by the example in Figure 20. The MQSeries Integrator broker, NEWBROKER, has been used to join MQSeries Publish/Subscribe brokers, BROKERA, and BROKERB.

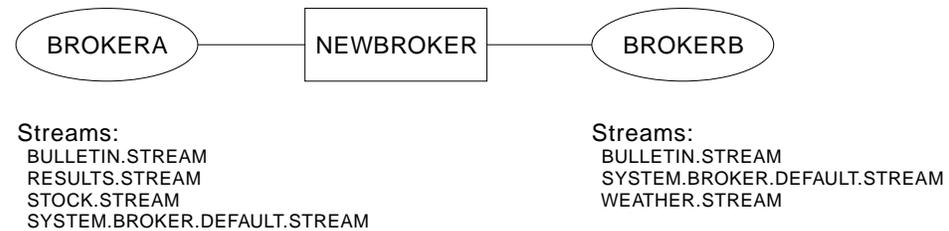


Figure 20. A heterogeneous network

The default stream queue `SYSTEM.BROKER.DEFAULT.STREAM` is always supported by every broker in an MQSeries Publish/Subscribe network, and must be defined at every MQSeries Integrator broker in a heterogeneous network. You must also define and deploy a message flow at each broker to service this queue.

When an MQSeries Integrator broker is integrated into an MQSeries Publish/Subscribe network, and links two or more MQSeries Publish/Subscribe brokers that share common streams, you must define the common stream queues, and define and deploy the message flows that service them, to the MQSeries Integrator broker.

For example, the MQSeries Integrator broker NEWBROKER shown in Figure 20 must have a stream queue defined for `BULLETIN.STREAM`. It must also have a message flow defined and deployed to provide a publication service for that queue.

You only need to define stream queues and associated message flows to the MQSeries Integrator broker for the other streams shown in Figure 20 if it is possible that one of its MQSeries Publish/Subscribe neighbors will send a message to one of these queues. A message will be sent if one of the following occurs:

1. A subscription to a publication on one of these streams is registered by a client of the MQSeries Integrator broker.

MQSeries Publish/Subscribe

2. A *DeletePublication* command for the stream is issued by a client anywhere within the broker network.

If you are unsure if the above cases might occur, you are recommended to create stream queues and message flows in the MQSeries Integrator broker for every stream that is supported by an MQSeries Publish/Subscribe neighbor. If you do not do this, you might see the following results:

- Messages sent from MQSeries Publish/Subscribe brokers will be put to the dead-letter queue (DLQ) of the MQSeries Integrator broker if the stream queue does not exist on that broker.
- Messages will build up on stream queues on the MQSeries Integrator broker if the stream queue exists but there is no message flow deployed to service it.

Streams and migration: When an MQSeries Publish/Subscribe broker is migrated to an MQSeries Integrator broker (using the **migmqbrk** command), the streams supported at the time of the migration are replicated exactly in the MQSeries Integrator broker: no subsequent changes can be made (that is, no streams can be added or removed from this replicated set). The migration is not complete until you have created and deployed message flows that process all of these streams.

Stream authority

In MQSeries Publish/Subscribe, all publish and subscribe authority checks are performed against the stream queue. Publishing applications need authority to put messages to the stream queue. The MQSeries Publish/Subscribe broker also checks the authority of subscribing applications which require browse authority on the stream queue. A subscribing application also needs to have put authority for the queue that it nominated to receive its publications.

The same check is made by MQSeries Integrator brokers, but the subscribe authority (browse) is no longer checked. Instead, MQSeries Integrator provides a more granular security model in which both publish and subscribe access can be defined in a hierarchical manner right down to an individual topic level. You can implement this model by creating Access Control Lists (ACLs) using the Control Center. For more information about ACLs, refer to *MQSeries Integrator Using the Control Center*.

Before you migrate an MQSeries Publish/Subscribe broker to a replacement MQSeries Integrator broker, or migrate your MQSeries Publish/Subscribe applications to run on MQSeries Integrator, you must consider the security implications:

- Publishing applications are subject to the same checks even if your broker is not running with topic security enabled, because the authority to put a message to the stream or publication queue continues to be checked by MQSeries.
However, stream publications can be processed by MQSeries Integrator on any input queue, because publishers no longer need to put to a queue with the same name as the stream. You are therefore recommended to set up equivalent ACLs for all streams using their corresponding topic level qualifiers
- The MQSeries Integrator broker does not check that subscribing applications have browse authority on the stream queue. Instead, MQSeries Integrator models streams by prefixing all topics that aren't part of the default stream with a unique prefix, `$$SYS/STREAM/<streamname>/`. This maintains the partitioning characteristics of streams and allows stream-specific ACLs to be set up. Topics in the default stream are not altered by the broker, therefore the root topic can be used to specify authorities for default stream topics.

Figure 21 illustrates the stream authorities that are required. This example assumes that you have updated the default ACL on the topic root for principal PublicGroup with authority for publish, subscribe, and persistent delivery all set to deny.

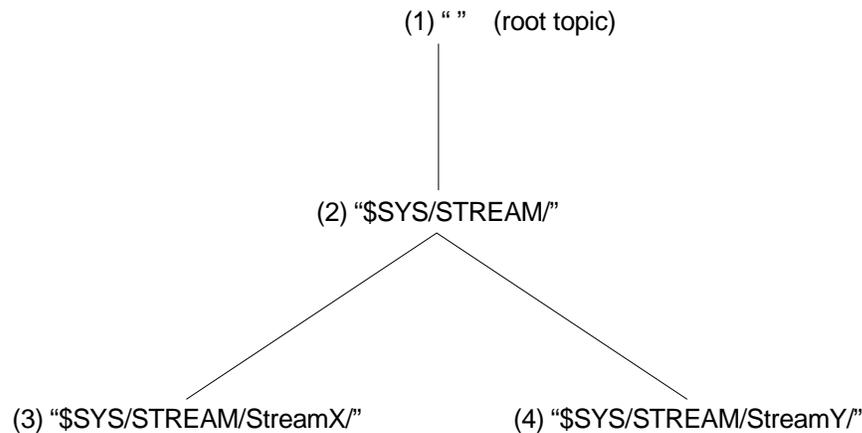


Figure 21. Stream authorities

Using this example, assume that the following groups are defined:

- PDefault: the group of users authorized to publish on the default stream
- SDefault: the group of users authorized to subscribe to the default stream
- PStreamX: the group of users authorized to publish on StreamX
- SStreamX: the group of users authorized to subscribe to StreamX
- PStreamY: the group of users authorized to publish on StreamY
- SStreamY: the group of users authorized to subscribe to StreamY

You must grant and deny authorities by setting up ACLs as follows:

1. PDefault must be granted publish authority on the root, SDefault must be granted subscribe authority on the root.
2. PDefault must be denied publish authority on \$SYS/STREAM/, SDefault must be denied subscribe authority on \$SYS/STREAM/.

These settings ensure that publishers and subscribers on the default stream are unable to publish on or subscribe to other streams automatically (that is, without an explicit ACL that overrides that setting).

3. PStreamX must be granted publish authority on \$SYS/STREAM/StreamX/, SStreamX must be granted subscribe authority on \$SYS/STREAM/StreamX/.

These settings override any setting on parent topics and limit publish and subscribe activity to users within these specific groups.

4. PStreamY must be granted publish authority on \$SYS/STREAM/StreamY/, SStreamY must be granted subscribe authority on \$SYS/STREAM/StreamY/.

These settings override any setting on parent topics and limit publish and subscribe activity to users within these specific groups.

If you wanted to set up exceptions to this situation, you can do so by introducing an ACL at the appropriate point. For example, if you wanted to grant authority to publishers to the default stream (PDefault) to publish on StreamX, you must create an explicit ACL at point (3) to grant that authority, thus overriding the denial at point (2). In this scenario, users in PDefault could still not publish on StreamY.

MQSeries Publish/Subscribe

Topics

In MQSeries Publish/Subscribe, all publications must be tagged with an arbitrary character string called a topic. This defines the subject matter of the publication. MQSeries Publish/Subscribe recommends, though does not enforce, that topic strings are structured into a number of fields or levels using the forward slash, “/”, as a delimiter.

MQSeries Integrator publications also have an associated topic, and the topic structure is delimited by the forward slash character. Therefore, if your existing applications follow the MQSeries Publish/Subscribe recommendation, they are better positioned to exploit the function provided by MQSeries Integrator, which allows the structure of the topic to be externalized.

MQSeries Integrator allows you to control users’ authority to publish on, and subscribe to, any topic at any level within the topic structure.

Wildcards

Wildcards can be used by subscribing applications to broaden the scope of publications they register an interest in. By specifying a wildcard, the subscriber is specifying a general pattern of the topics they are interested in, rather than an explicit topic.

This function is provided by both MQSeries Publish/Subscribe and MQSeries Integrator. However, MQSeries Integrator provides a different set of wildcards that allow a more extensive and flexible use of wildcards by subscribers.

- MQSeries Publish/Subscribe wildcards:
 - An asterisk (*) matches zero or more characters.
 - A question mark (?) matches exactly one character.
 - The percent sign (%) can be used as an escape character to use an “*”, a “?”, or a “%” character within a topic.
- MQSeries Integrator wildcards:

The wildcard characters are used to match specific levels within the structured topic. The characters used are:

 - The multi-level wildcard (the character #), that matches any number of levels at the start or end of the topic.
 - The single-level wildcard (the character +), that matches a single level within the topic.

The full range of function of the MQSeries Integrator wildcards are only available to MQRFH2 clients. Subscriptions made by MQRFH clients to MQSeries Integrator brokers for topics that contain either of the MQSeries Integrator wildcards are rejected with the MQRCCF_TOPIC_ERROR reason code.

Applications using MQRFH and connecting to MQSeries Publish/Subscribe brokers in a heterogeneous network are therefore recommended not to publish on, or subscribe to, topics containing either the multi-level wildcard (#) or single-level wildcard (+) characters. MQSeries Publish/Subscribe brokers do not police this: if your applications specify the MQSeries Integrator wildcards in topics when they publish or register a subscription in a heterogeneous broker network, these publications and subscriptions are ignored by MQSeries Integrator brokers within the network. You are therefore strongly advised to review and if necessary change the topics being used within an MQSeries Publish/Subscribe implementation before adding an MQSeries Integrator broker to the network.

MQSeries Publish/Subscribe

When applications that use MQRFH2 use the MQSeries Integrator wildcards to target multiple publications from within the MQSeries Publish/Subscribe network, wildcard mapping is performed. In most cases, the broker replaces both the multi-level wildcard and single-level wildcard characters with an asterisk. This does not provide an exact match for either of the MQSeries Integrator wildcards, but ensures a superset of the required publications are sent to the MQSeries Integrator broker. The MQSeries Integrator broker evaluates the “#” and “+” wildcards to match the correct publications.

For example, the topic “employee/+/development” is propagated as “employee/*/development” to an MQSeries Publish/Subscribe neighbor. This might cause redundant publications to be sent to the MQSeries Integrator broker from its MQSeries Publish/Subscribe neighbor. However, none of these would be sent to the original client when the MQSeries Integrator evaluates the original subscription. The exception to this is a subscription to the topic “+” which is never propagated: it cannot be represented as an “*” because this is the topic that is propagated if a subscription to topic “#” is made at the MQSeries Integrator broker.

You are recommended not to specify the MQSeries Publish/Subscribe wildcard characters in MQRFH2 client subscriptions. If you do specify one or more, they are assumed by MQSeries Integrator to be part of the topic, and are therefore prefixed by the escape character (%) before the subscription is sent on to an MQSeries Publish/Subscribe neighbor.

For example, if your MQRFH2 client subscribes with a topic of “USA/Alaska*/Juneau?”, this is modified and passed to an MQSeries Publish/Subscribe broker neighbor as “USA/Alaska%*/Juneau?”.

If an application using MQRFH connects to an MQSeries Integrator broker, MQSeries Integrator emulates the behavior of the MQSeries Publish/Subscribe wildcard characters * and ? using a mixture of its own wildcard characters and filter expressions. Existing MQRFH applications that subscribe to an MQSeries Integrator broker therefore receive the same publications as they would receive if they subscribe to an MQSeries Publish/Subscribe broker.

Default topic routing

In MQSeries Integrator, the *Topic* property of the MQInput node can be used to route messages that do not contain publish/subscribe parameters. This feature does not apply to MQRFH subscribers.

MQRFH subscribers expect to receive publications, with a well-formed MQRFH header, from both MQSeries Publish/Subscribe and MQSeries Integrator clients. In the latter case, the original MQRFH2 header is converted as described in Table 8 on page 149. However, if the message does not contain publish/subscribe information in either an MQRFH or an MQRFH2 header, the default topic will not be used to send publications to an MQRFH subscriber.

Retained publications

In MQSeries Publish/Subscribe, retained publications are published as non-persistent messages and are therefore automatically deleted when the broker’s queue manager is restarted. In MQSeries Integrator, retained publications are persistent and are preserved across queue manager restarts.

Metatopics

MQSeries Publish/Subscribe brokers provide information about publishers and subscribers via a special set of topics called metatopics. These topics start with the

MQSeries Publish/Subscribe

“MQ/S/” or “MQ/SA/” prefix, and are subscribed to by two categories of applications, administration programs and clients.

MQSeries Integrator does not provide equivalent metatopics, and therefore any existing program (administration or client) that subscribes to MQSeries Publish/Subscribe metatopics cannot work with an MQSeries Integrator broker. However, MQSeries Integrator does publish information about subscription events using its own set of system topics. These are described in the *MQSeries Integrator Administration Guide*.

The following considerations apply for the two categories of application in the MQSeries Integrator environment:

- Administration programs such as the **amqpsd** sample use the MQSeries Publish/Subscribe metatopics to display subscription information. This information is provided by MQSeries Integrator in the Control Center, which provides an interface to view and delete subscriptions throughout the broker network.
- Applications use messages published on MQSeries Publish/Subscribe metatopics, for example, to request information about their own current subscriptions.

A client program can subscribe to MQSeries Integrator system topics and process the event publications. MQSeries Integrator does not provide a topic that reports all of the current subscriptions for a particular topic or client, but does publish whenever subscriptions are added or removed. This information is published as event information not state information (MQSeries Publish/Subscribe metatopics are published as state information). For more information about event and state publications, see “State and event information” on page 76.

Subscription points

Subscription points are a feature provided by MQSeries Integrator that can be used to make information associated with a particular topic available in a number of different formats.

For example, stock prices might be published with a default currency of dollars, but might be required by subscribers in a number of other currencies.

This can be achieved by defining additional paths through the message flow that take each publication and convert the dollar stock price into another currency, for example sterling, before it is passed to its Publication node.

Each additional currency must be associated with a different subscription point and therefore a Publication node. The original publication in dollars is associated with the default subscription point.

Subscribers can then subscribe to stock prices using a combination of topic and the subscription point that provides the data in the correct currency.

Subscription points are not supported by MQSeries Publish/Subscribe. You must therefore consider their use in a heterogeneous network carefully. In particular, publications can only pass between MQSeries Integrator and MQSeries Publish/Subscribe brokers on the default subscription point.

Also, all topics published in an MQSeries Publish/Subscribe broker domain are on the default subscription point. These topics are only available to MQRFH2

subscribers that subscribe to the topics without specifying a subscription point (that is, they are using the default subscription point).

Similarly, clients at MQSeries Publish/Subscribe brokers can only subscribe to topics that are published on the default subscription point at MQSeries Integrator brokers (at Publication nodes that do not have a subscription point set).

Content-based filtering

MQSeries Integrator supports content-based filtering of publications. This is a powerful and flexible option for publish/subscribe application suites. This option significantly enhances the ability of the MQRFH2 subscriber to restrict the messages they wish to receive.

When an MQRFH2 client registers a subscription with the local broker, it can specify a filter to be applied to the content of fields within each publication message.

An MQRFH2 subscriber can subscribe to MQRFH publications within the MQSeries Publish/Subscribe part of a mixed broker network based upon the restrictions mentioned in this chapter. All MQRFH publications are converted to MQRFH2 format by the broker before delivery to the MQRFH2 client (see Table 8 on page 149 for conversion details).

An MQRFH2 subscriber can also request some very restricted content-based filtering to be performed on the MQRFH publications they are subscribing to. This can only be done if the body of the publication is in a format that can be parsed by the broker: that is, it can be interpreted by one of the broker's default parsers (described in "Message parsers" on page 59). For example, messages in XML or MQPCF format can be processed in this way.

If you want to make full use of content-based filtering, you must convert publications into MQRFH2 format. This enables all messages defined in the message repository to be interpreted by the brokers parsers. MQRFH clients are not able to specify a content filter.

For more details about message formats, their construction, and the message repository, see *MQSeries Integrator Using the Control Center*.

Throughput

In MQSeries Publish/Subscribe a single thread processes publications on each of the stream queues. This guaranteed the order in which publications were processed from the queue. When you consider throughput for publications in an MQSeries Integrator broker domain, you must also consider the importance of the order in which messages are published. The techniques to increase throughput do not necessarily guarantee order.

MQSeries Integrator supports two options that increase throughput:

1. You can configure the message flow with additional threads by setting the *Additional Instances* property of the MQInput node. This property instructs the broker to schedule additional threads to read messages from the input queue, thus allowing publications from that queue to be processed concurrently by the broker. You must also ensure that the stream (input) queue has the share attribute set (MQSeries Publish/Subscribe required stream queues to have noshare set).

If multiple threads process messages from a single queue, publications are not guaranteed to be delivered to subscribers in the order in which they are placed

MQSeries Publish/Subscribe

on the input queue. Therefore MQSeries Integrator provides a simple ordering facility that can be used to allow concurrent processing of publications whilst still maintaining some sequence:

You can set the *Order Mode* property of the MQInput node to the value *By User ID*. This will ensure the order of delivery of publications sent to the broker by a given user. When this property is set, the processing of messages that carry a given *UserIdentifier* field in the MQMD will be held up if any other thread servicing that message flow is currently processing a message that carries the same *UserIdentifier*.

The benefits of running additional instances of the message flow will be negated if all publishing applications are running under the same user ID. This might be the case for publishing applications connected to a queue manager remote to the broker's queue manager. Messages from these remote publishers arrive at the broker via a channel that might have been set up to insert the channel program's user ID in place of the originating client's user ID. Refer to the *MQSeries Intercommunications* book for more information on how to set the PUTAUT channel attribute to change the default channel behavior.

2. You can configure one or more additional message flows (not instances) that read publications from different queues. You must also update some of your publishing applications to publish to the new queue (or queues). This has the effect of splitting the stream, and therefore spreading the workload.

If you choose to increase throughput using this method, you must consider the impact this has on the order in which publications are delivered. In particular you must ensure that the publisher applications are split with respect to the topics they are publishing on to ensure that order can be maintained per topic, if this is important. If your applications publish to different queues (message flows) on the same topic order cannot be guaranteed.

If you update the publisher applications to send publications to a new queue which has a different name to the stream on which they are publishing, you must also update these applications to explicitly include the stream name within their publications using the MQPSSStreamName parameter.

Publishing applications that specify a stream parameter do not need to be modified, as this parameter takes precedence. However, if publishing applications do not specify the stream parameter, the behavior is determined by the setting of the *Implicit Stream Naming* property of the publication node in the message flow:

- If the property is set to *false*, the default stream is assumed.
- If the property is set to *true*, the stream name is assumed to be the same as the name of the stream input queue.

Scenario 1: running two independent broker networks

If you already have an MQSeries Publish/Subscribe broker network, you can continue to use this network unchanged. The introduction of MQSeries Integrator Version 2.0.2 to your environment, and the creation of brokers in that broker domain, does not affect your MQSeries Publish/Subscribe broker domain until you take specific action to connect the two 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 2.0.2 network, without any interaction.

Your existing applications can continue to work unchanged. However, there can be no interchange of publications in this scenario.

You must be aware that a single queue manager cannot support both an MQSeries Publish/Subscribe broker and an MQSeries Integrator Version 2.0.2 broker. If you have brokers of both types on the same system, each broker must have its own dedicated queue manager.

You can implement this scenario while you assess the new product and the extra functions contained within the publish/subscribe support. It also lets you plan for the extent of integration or migration, or both, that you require, without affecting your current environment.

Scenario 2: creating and operating a heterogeneous network

When you have operated two separate networks for a while, and understand the benefits that MQSeries Integrator Version 2.0.2 provides, you can take the next step of setting up an integrated network with a mix of MQSeries Publish/Subscribe and MQSeries Integrator brokers.

A heterogeneous network enables publications and subscriptions to be propagated through one logical network, made up of two physical networks.

Applications registered with all brokers (MQSeries Integrator Version 2.0.2 and MQSeries Publish/Subscribe) are not aware that there is a heterogeneous network, and, subject to authorizations being in place and the product differences addressed, can publish and subscribe freely.

One of the advantages of creating and operating a heterogeneous network is that it allows you to integrate MQSeries Publish/Subscribe brokers running on operating systems where you currently do not have MQSeries Integrator Version 2.0.2 installed. You can integrate them with new MQSeries Integrator Version 2.0.2 brokers, or with those migrated from MQSeries Publish/Subscribe brokers on the same operating system, or both.

You also create and operate a heterogeneous network while you implement migration, because you are not required to migrate your whole MQSeries Publish/Subscribe broker network in one step. See “Scenario 3: migrating MQSeries Publish/Subscribe brokers” on page 162 for details about migrating individual brokers.

To achieve a heterogeneous network, you must:

- Select the brokers that are to join the two networks together.

The hierarchical structure of the MQSeries Publish/Subscribe network, with a single root broker (node) and a number of leaf nodes, allows you to integrate the two networks in two ways:

- You can add a single MQSeries Integrator Version 2.0.2 broker to the MQSeries Publish/Subscribe network as a root node. The MQSeries Publish/Subscribe hierarchy results in the heaviest workload at the root node. If you add an MQSeries Integrator Version 2.0.2 broker as a new root, all MQSeries Publish/Subscribe message traffic will be processed by this node.
- You can add one or more MQSeries Integrator Version 2.0.2 brokers to the MQSeries Publish/Subscribe network as leaf nodes. This option minimizes the additional workload placed on the MQSeries Integrator Version 2.0.2 broker.

MQSeries Publish/Subscribe

- Establish message flows that provide the publish/subscribe services required in the MQSeries Integrator broker.
The choices you have for implementing these message flows have already been discussed in “Throughput” on page 159.

Details of how you implement these actions are described in the *MQSeries Integrator Administration Guide*.

Scenario 3: migrating MQSeries Publish/Subscribe brokers

This third scenario describes the planning you must do when you decide to migrate your MQSeries Publish/Subscribe brokers. This is likely to be the final stage of your adoption of MQSeries Integrator into your current MQSeries Publish/Subscribe environment.

The action of migrating an MQSeries Publish/Subscribe broker to an MQSeries Integrator broker *replaces* the broker. This is a final step, from which it is difficult to return.

You must therefore ensure you have considered the move carefully, and have taken any actions or decisions necessary to ensure a smooth transition.

You are advised to consider the following:

- The order in which you migrate the brokers
You do not have to migrate all the brokers in the network at once. You can migrate brokers one at a time, thus creating an intermediate state in which the network consists of a mixture of MQSeries Publish/Subscribe and MQSeries Integrator brokers.
In fact, a mixed network of this nature might be the final state of the network, because you cannot migrate brokers that have been created on an operating system not supported by MQSeries Integrator.
If you have a choice of which brokers to migrate first, you are advised to migrate leaf nodes first. These brokers have a single relation in the network (a parent) and their migration is therefore easier to plan and implement.
- The place of each broker in the network
Each broker you migrate has at least one neighbor, its parent. You are advised to quiesce client applications on all related brokers, and stop the brokers, in addition to the one you are migrating.
- The use of collectives in the MQSeries Integrator network
A collective removes a single point of failure, and therefore increases the resilience of every individual node in the publish/subscribe network. For more information about using collectives, see “Supporting publish/subscribe services” on page 110.

Table 11 on page 164 identifies the areas of potential incompatibility due to the upgraded behavior of MQSeries Integrator. It provides some hints as to when, and how, you might need to make changes to your client applications, or the topics they use.

If you do make changes, you must test your changes for correctness by running the changed items in your MQSeries Publish/Subscribe network for a reasonable period of time before migrating to MQSeries Integrator.

Migration checklist

When you have identified the MQSeries Publish/Subscribe broker or brokers that you want to migrate to MQSeries Integrator, you must work through the items presented in Table 11 on page 164 to ensure your migration is transparent to your client applications.

You need an in-depth knowledge of both the broker, and the client applications that are using it, to determine exactly which items affect your environment.

You will find the MQSeries Publish/Subscribe sample administration program, **amqpsd**, which reports on the state of an MQSeries Publish/Subscribe broker, helps you to identify some of the problem areas listed here. Refer to the *MQSeries Publish/Subscribe User's Guide* for full details of the operation of this program.

MQSeries Publish/Subscribe

Table 11. Migration inhibitors checklist

Item	Suggested discovery	Suggested resolution	Chkd
Topics			
No topics contain the # or the + character	Check full output from amqspsd .	Redesign topics ¹	
No applications are subscribing to metatopics	Check full output from amqspsd for subscription to topics starting with either "MQ/S/" or "MQ/SA/"	No equivalent MQSeries Integrator functions ²	
Streams			
No user-defined topics have been added to the administration stream	Check topics returned in the output from amqspsd filtered by administration stream	Move subscriptions and publications to existing or new stream ³	
Common streams shared between broker and its relations do not need to change ⁴	No new common streams are needed in the future	After migration, remove the MQSeries Integrator broker from the MQSeries Publish/Subscribe network and add it (rejoin it) again	
Capacity			
Is the broker running near to full capacity	Any reported instances of messages building up on the control queue or any of the stream queues	After migration, create additional message flow instances to spread the workload ⁵	
Message formats			
No publishing applications are using MQPCF messages ⁶	Check publishing applications	Change applications to use MQRFH format	
User exit			
No routing exit is being used	Check for the presence of the <i>routingexit</i> configuration parameter	No equivalent MQSeries Integrator functions ⁷	
<p>Notes:</p> <ol style="list-style-type: none"> 1. If the topics being used by your publisher and subscriber applications need to be redesigned, this might involve more than simply changing the affected client applications. Subscriptions and retained publications that reference the invalid topics need to be removed. Also brokers need to be stopped so that all processing on the affected topics is suitably quiesced in the entire broker network, prior to deploying the modified publisher and subscriber applications. 2. This issue is discussed in "Metatopics" on page 157. MQSeries Publish/Subscribe and MQSeries Integrator do not provide fully compatible function for metatopics. 3. If the administration stream (stream queue SYSTEM.BROKER.ADMIN.STREAM) has been used for convenience by client applications, these topics need to be moved to another stream supported by all brokers in the network. No subscriptions or retained publications are migrated on this stream. 4. If the broker is part of multibroker network, MQSeries Integrator brokers will not respond to stream support changes at neighboring MQSeries Publish/Subscribe brokers. If you require the replacement broker to support other streams, the MQSeries Integrator broker must be removed from the MQSeries Publish/Subscribe network, and added again. 5. MQSeries Publish/Subscribe and MQSeries Integrator have different operational characteristics that make it difficult to compare their performance directly. In particular, MQSeries Integrator stores its persistent data within a database. You are advised to model your broker's current workload with an MQSeries Integrator broker prior to migration. MQSeries Integrator throughput can be increased in two ways: see "Throughput" on page 159 for details. 6. MQSeries Integrator brokers only accept publications made in MQRFH or MQRFH2 format. The migmqbrk command does not export MQPCF retained publications to the replacement MQSeries Integrator broker. 7. If only a small majority of publications need to be processed by the user exit, an additional MQSeries Publish/Subscribe broker could be created to host affected subscribers prior to migration. The subscribing applications themselves do not need to be moved to the new broker, but their subscriptions do need to be rerouted. The user exit code can then run at the new broker which would not be migrated. 			

Appendix B. The product packages

The following sections provide a summary of the contents of the packages for MQSeries Integrator for AIX, MQSeries Integrator for HP-UX, MQSeries Integrator for Sun Solaris, and MQSeries Integrator for Windows NT. For exact details, you must refer to the product `Readme.txt` file, and to the *MQSeries Integrator Installation Guide* for you product.

The MQSeries Integrator for AIX package

The MQSeries Integrator for AIX package includes the following CD-ROMs:

1. MQSeries Integrator for AIX V2.0.2

This primary-product CD-ROM provides the product in all available national languages. It also includes the installable documentation package, the NEONRules and NEONFormatter Support, and the Tivoli interface support files and documentation. For more information, see the `Readme.txt`.

2. MQSeries Integrator for Windows NT V2.0.2

3. MQSeries Integrator for AIX V2.0.2 'Supplement', which includes:

- MQSeries for AIX Version 5.1 CSD6

This CSD is provided to enable you to upgrade an existing installation of MQSeries for AIX Version 5.1 that does not have at least CSD4, should you choose not to upgrade to Version 5.2.

- The MQSeries Integrator Version 2.0.2 documentation PDF package that can be viewed without installation. You are recommended to install the PDF package from the primary product CD, but you might choose to refer to the product library before installation.

- MQSeries for Windows NT Version 5.1 CSD6

This CSD is provided to enable you to upgrade an existing installation of MQSeries for Window NT Version 5.1 that does not have at least CSD4, should you choose not to upgrade to Version 5.2.

- Any additional product service updates required for any product supplied in this package are included on this CD. Up-to-date details of the service levels required are included in the MQSeries Integrator Version 2.0.2 `Readme.txt` file on the primary product CD.

4. MQSeries Integrator for AIX V2.0.2 'DB2 for AIX 7.1'

This is supplied for specific use with MQSeries Integrator. If you do not already have a suitable database to use, you must install this product before you install MQSeries Integrator.

5. MQSeries Integrator for Windows NT V2.0.2 'DB2 for Windows 7.1' (English and EMEA Languages)

This is supplied for specific use with MQSeries Integrator

6. MQSeries Integrator for Windows NT V2.0.2 'DB2 for Windows 7.1' (English and AP Languages)

This is supplied for specific use with MQSeries Integrator

7. MQSeries for AIX V5.2 (Server)

The MQSeries Integrator installation program checks that you have the appropriate components of MQSeries installed on your system. The Runtime

Product package for AIX

component requires the MQSeries for AIX server at V5.1 or later. Other components have no MQSeries dependency.

If any MQSeries component is required for MQSeries Integrator installation, and you do not have the correct level already installed, you must install this CD. If you already have Version 5.0 or Version 5.1, you can use these CDs to migrate to Version 5.2.

8. MQSeries for Windows NT V5.2 (Server)

The installation program checks that you have the appropriate components of MQSeries installed on your system. Some MQSeries Integrator components require MQSeries for Windows NT server at V5.1 or later, the Control Center and the Configuration Manager require MQSeries classes for Java support. A few components have no MQSeries dependency.

If any MQSeries component is required for MQSeries Integrator installation, and you do not have the correct level already installed, you must install this CD. If you already have Version 5.0 or Version 5.1, you can use these CDs to migrate to Version 5.2.

9. MQSeries V5.2 Clients

MQSeries Clients for all platforms in all available national languages are included on this CD.

The following hardcopy installation books are supplied:

- *MQSeries Integrator for AIX Installation Guide*
- *MQSeries for AIX Quick Beginnings*
- *MQSeries for Windows NT and Windows 2000 Quick Beginnings*

The MQSeries Integrator for HP-UX package

The MQSeries Integrator for HP-UX package includes the following CD-ROMs:

1. MQSeries Integrator for HP-UX V2.0.2

This primary-product CD-ROM provides the product in all available national languages. It also includes the installable documentation package and the NEONRules and NEONFormatter Support. For more information, see the `Readme.txt`.

2. MQSeries Integrator for Windows NT V2.0.2

3. MQSeries Integrator for HP-UX V2.0.2 'Supplement', which includes:

- The MQSeries Integrator Version 2.0.2 documentation PDF package that can be viewed without installation. You are recommended to install the PDF package from the primary product CD, but you might choose to refer to the product library before installation.
- MQSeries for Windows NT Version 5.1 CSD6
This CSD is provided to enable you to upgrade an existing installation of MQSeries for Window NT Version 5.1, should you choose not to upgrade to Version 5.2.
- Any additional product service updates required for any product supplied in this package are included on this CD. Up-to-date details of the service levels required are included in the MQSeries Integrator Version 2.0.2 `Readme.txt` file on the primary product CD.

4. MQSeries Integrator for HP-UX V2.0.2 'DB2 for HP-UX 7.1'

This is supplied for specific use with MQSeries Integrator. If you do not already have a suitable database to use, you must install DB2 from this CD (before or after MQSeries Integrator installation).

5. MQSeries Integrator for Windows NT V2.0.2 'DB2 for Windows 7.1' (English and EMEA Languages)

This is supplied for specific use with MQSeries Integrator

6. MQSeries Integrator for Windows NT V2.0.2 'DB2 for Windows 7.1' (English and AP Languages)

This is supplied for specific use with MQSeries Integrator

7. MQSeries for HP-UX V5.2 (HP 11 Server)

The MQSeries Integrator installation program checks that you have the appropriate components of MQSeries installed on your system. The runtime component requires the MQSeries for HP-UX V5.2 server. Other components have no MQSeries dependency.

If any MQSeries component is required for MQSeries Integrator installation, and you do not have the correct level already installed, you must install this CD.

8. MQSeries for Windows NT V5.2 (Server)

The installation program checks that you have the appropriate components of MQSeries installed on your system. Some MQSeries Integrator components require MQSeries for Windows NT server at V5.1 or later, the Control Center and the Configuration Manager require MQSeries classes for Java support. A few components have no MQSeries dependency.

If any MQSeries component is required for MQSeries Integrator installation, and you do not have the correct level already installed, you must install this CD. If you already have Version 5.0 or Version 5.1, you can use these CDs to migrate to Version 5.2.

9. MQSeries V5.2 Clients

MQSeries Clients for all platforms in all available national languages are included on this CD.

The following hardcopy installation books are supplied:

- *MQSeries Integrator for HP-UX Installation Guide*
- *MQSeries for HP-UX Quick Beginnings*
- *MQSeries for Windows NT and Windows 2000 Quick Beginnings*

The MQSeries Integrator for Sun Solaris package

The MQSeries Integrator for Sun Solaris package includes the following CD-ROMs:

1. MQSeries Integrator for Sun Solaris V2.0.2

This primary product CD-ROM provides the product in all available national languages. It also includes the installable documentation package, the NEONRules and NEONFormatter Support, and the Tivoli interface support files and documentation. For more information, see the `Readme.txt`.

2. MQSeries Integrator for Windows NT V2.0.2

3. MQSeries Integrator for Sun Solaris V2.0.2 'Supplement', which includes:

- MQSeries for Sun Solaris Version 5.1 CSD6

This CSD is provided to enable you to upgrade an existing installation of MQSeries for Sun Solaris Version 5.1 that does not have at least CSD4, should you choose not to upgrade to Version 5.2.

- The MQSeries Integrator Version 2.0.2 documentation PDF package that can be viewed without installation. You are recommended to install the PDF package from the primary product CD, but you might choose to refer to the product library before installation.

Product package for Sun Solaris

- MQSeries for Windows NT Version 5.1 CSD6
This CSD is provided to enable you to upgrade an existing installation of MQSeries for Window NT Version 5.1, should you choose not to upgrade to Version 5.2.
 - Any additional product service updates required for any product supplied in this package are included on this CD. Up-to-date details of the service levels required are included in the MQSeries Integrator Version 2.0.2 Readme.txt file on the primary product CD.
4. MQSeries Integrator for Sun Solaris V2.0.2 'DB2 for Sun Solaris 7.1'
This is supplied for specific use with MQSeries Integrator. If you do not already have a suitable database to use, you must install DB2 from this CD (before or after MQSeries Integrator installation).
 5. MQSeries Integrator for Windows NT V2.0.2 'DB2 for Windows 7.1' (English and EMEA Languages)
This is supplied for specific use with MQSeries Integrator
 6. MQSeries Integrator for Windows NT V2.0.2 'DB2 for Windows 7.1' (English and AP Languages)
This is supplied for specific use with MQSeries Integrator
 7. MQSeries for Sun Solaris V5.2 (Server)
The MQSeries Integrator installation program checks that you have the appropriate components of MQSeries installed on your system. The runtime component requires the MQSeries for Sun Solaris server at V5.1 or later. Other components have no MQSeries dependency.
If any MQSeries component is required for MQSeries Integrator installation, and you do not have the correct level already installed, you must install this CD. If you already have Version 5.0 or Version 5.1, you can use these CDs to upgrade to Version 5.2.
 8. MQSeries for Windows NT V5.2 (Server)
The installation program checks that you have the appropriate components of MQSeries installed on your system. Some MQSeries Integrator components require MQSeries for Windows NT server at V5.1 or later, the Control Center and the Configuration Manager require the MQSeries Client for Java. A few components have no MQSeries dependency.
If any MQSeries component is required for MQSeries Integrator installation, and you do not have the correct level already installed, you must install this CD. If you already have Version 5.0 or Version 5.1, you can use these CDs to upgrade to Version 5.2.
 9. MQSeries V5.2 Clients
MQSeries Clients for all platforms in all available national languages are included on this CD.

The following hardcopy installation books are supplied:

- *MQSeries Integrator for Sun Solaris Installation Guide*
- *MQSeries for Sun Solaris Quick Beginnings*
- *MQSeries for Windows NT and Windows 2000 Quick Beginnings*

The MQSeries Integrator for Windows NT package

The MQSeries Integrator for Windows NT package includes the following CD-ROMs:

1. MQSeries Integrator for Windows NT V2.0.2

Product package for Windows NT

This primary-product CD-ROM provides the product in all available national languages. It also includes the installable documentation package and the NEONRules and NEONFormatter Support, and the Tivoli interface support files and documentation. For more information, see the Readme.txt.

2. MQSeries Integrator for Windows NT V2.0.2 'DB2 for Windows 7.1' (English and EMEA Languages)

This is supplied for specific use with MQSeries Integrator. If you do not have a suitable version of DB2 installed, the MQSeries Integrator Version 2.0.2 install process will offer to install DB2 for you.

3. MQSeries Integrator for Windows NT V2.0.2 'DB2 for Windows 7.1' (English and AP Languages)

This is supplied for specific use with MQSeries Integrator. If you do not have a suitable version of DB2 installed, the MQSeries Integrator Version 2.0.2 install process will offer to install DB2 for you.

4. MQSeries Integrator for Windows NT V2.0.2 'Supplement', which includes:

- MQSeries for Windows NT, V5.1 CSD6

This CSD is provided to enable you to upgrade an existing installation of MQSeries for Windows NT Version 5.1 that does not have at least CSD4, should you choose not to upgrade to Version 5.2.

- The MQSeries Integrator Version 2.0.2 documentation PDF package that can be viewed without installation. You are recommended to install the PDF package from the primary product CD, but you might choose to refer to the product library before installation.
- Any additional product service updates required for any product supplied in this package are included on this CD. Up-to-date details of the service levels required are included in the MQSeries Integrator Version 2.0.2 Readme.txt file on the primary product CD.

5. MQSeries for Windows NT V5.2 (Server)

The MQSeries Integrator installation program checks that you have the appropriate components of MQSeries installed on your system. Some MQSeries Integrator components require MQSeries for Windows NT server at V5.1 or later. Version 5.2 is supplied primarily for those customers who do not have an existing MQSeries installation. However, you can also use this CD to upgrade an existing MQSeries Version 5.1 installation to Version 5.2. If you do this upgrade, refer to the migration instructions supplied with MQSeries Version 5.2.

6. MQSeries V5.2 Clients

MQSeries Clients for all platforms in all available national languages are included on this CD.

The following hardcopy installation books are supplied:

- *MQSeries Integrator for Windows NT Installation Guide*
- *MQSeries for Windows NT and Windows 2000 Quick Beginnings*

Product package for Windows NT

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	AS/400	CICS
DB2	DB2 Universal Database	IBM
IBMLink	IMS/ESA	iSeries
MQSeries	POWERparallel	POWERserver
pSeries	RS/6000	SupportPac
System/390	TXSeries	WebSphere
xSeries	400	@server

Lotus, Notes, and Domino are trademarks of Lotus Development Corporation in the United States, other countries, or both.

Tivoli is a trademark of Tivoli Systems Inc. in the United States, other countries, or both.

Pentium is a trademark of Intel Corporation in the United States, other countries, or both.

Java, JDBC, and JDK are registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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
- NEONMSG⁴, 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.

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.

4. 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 184). 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 `mqsidocs`. 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 `mqsidocs`. This launches Acrobat Reader and opens the PDF package.
- On Sun Solaris, invoke `pkgadd -d` and select `mqsi-docs` from the menu. After installation, run the command `mqsidocs`. 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 `mqsi_root/bin/book/pdf/<locale>` (on UNIX) or `mqsi_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, `<mqsi_root>/docs/`
- On Windows NT, `<mqsi_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 12.

Table 12. 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 library references

The following MQSeries product publications 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.

For AIX installation tasks you might need:

- *MQSeries for AIX Quick Beginnings*, GC33-1867

For HP-UX installation tasks you might need:

- *MQSeries for HP-UX Quick Beginnings*, GC33-1869

For Sun Solaris tasks you might need:

- *MQSeries for Sun Solaris Quick Beginnings*, GC33-1870

For Windows NT installation tasks you might need:

- *MQSeries for Windows NT and Windows 2000 Quick Beginnings*, GC33-1871

For planning and configuration tasks you might need:

- *MQSeries Intercommunication*, SC33-1872
- *MQSeries System Administration*, SC33-1873
- *MQSeries Queue Manager Clusters*, SC34-5349
- *MQSeries MQSC Command Reference*, SC33-1369
- *MQSeries for AIX Quick Beginnings*, GC33-1632
- *MQSeries Messages*, GC33-1876

For application programming tasks you might need:

- *MQSeries Application Programming Reference*, SC33-1673
- *MQSeries Application Programming Guide*, SC33-0807
- *MQSeries Application Messaging Interface*, SC34-5604
- *MQSeries Using Java*, SC34-5456

For migrating from MQSeries Integrator Version 1 you might need:

- *MQSeries Integrator Version 1.1 Installation and Configuration Guide*, GC34-5503

MQSeries Publish/Subscribe publications

If you have installed MQSeries Publish/Subscribe and plan to migrate brokers to MQSeries Integrator Version 2, or to establish a mixed network, refer to the following publication:

- *IBM MQSeries Publish/Subscribe User's Guide*, GC34-5269

You can download this book and the MQSeries Publish/Subscribe Product Extension package from the MQSeries Web site.

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:

- *IBM MQSeries Workflow Concepts and Architecture*, GH12-6285

For a complete list of MQSeries Workflow publications, refer to the information on the MQSeries Web site.

MQSeries library

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

A

- access control list 28
 - activating 91
 - checking 91
 - deployment 91
 - dynamic topics 90
 - explicit 87
 - inheritance 87, 89
 - permissions 89
 - persistent delivery 89
 - PublicGroup 87
 - resolution of conflicts 88
 - setting 89
 - settings 87
 - subscription resolution 91
 - system topics 88
 - wildcards 90
- AIX installation
 - delivery media 165
 - disk space requirements 97
 - hardware requirements 96
 - software requirements 97
 - system requirements 96
- Application Messaging Interface 23, 66
- applications
 - communication models 65
 - design 65
 - existing 68
 - MQSeries resources 116
 - point-to-point 65, 66
 - programming interfaces 66
 - publish/subscribe 65, 66, 69
 - request/reply 68
 - security 73
 - send and forget 68
 - transaction support 71
 - writing new 69
- Assignment view 16

B

- broker 11
 - associating message flows 16
 - associating message sets 19
 - broker tables 11
 - client connections 24
 - collective 12
 - connections for publish/subscribe 12
 - creating a reference 11
 - execution group 48
 - message flows 12
 - message sets 12
 - MQSeries resources 114
 - publish/subscribe interactions 75
 - system management interface 13
- broker domain
 - advanced options 139
 - basics 109
 - business processes 134
 - changing components 133

- broker domain (*continued*)
 - client applications 113
 - Control Center 113
 - listing components 133
 - managing components 133
 - monitoring 134
 - MQSeries infrastructure 114
 - performance 137
 - planning 107
 - plug-ins 139
 - starting components 133
 - stopping components 133
 - supporting publish/subscribe 110
 - system management 138
 - workload 137
- business integration 4
 - MQSeries family 4
- business process rules 3, 13
- business scenario 33, 37
 - business data 35
 - business needs 36
 - finance flow 37
 - implementation 38
 - MQSeries Integrator solution 36
 - partner flow 37
 - retail 33
 - stock flow 37

C

- change management
 - SupportPac IC04 21
- CICS 3
- clusters, MQSeries 117
- code page, database 119
- code page support 105
- collective 12, 110
- commit 14
- communication models
 - point-to-point 3, 23
 - publish/subscribe 3, 23
- complementary offerings
 - IBM Business Partners 184
 - Partner Offerings 184
- Configuration Manager 9
 - configuration repository 9, 10
 - message repository 10
 - Message Repository Manager (MRM) 55
 - MQSeries resources 115
- configuration repository 9
- Control Center 20
 - assigning 21
 - Assignment view 16
 - change management 21
 - check in 21
 - check out 20
 - connection to Configuration Manager 20
 - Debugger 54
 - deploying 21

- Control Center 20 (*continued*)
 - export 21
 - import 21
 - MQSeries authority 129
 - MQSeries resources 116
 - roles 128
 - security 128
 - superuser 129
 - updating 20

D

- data conversion 131
 - code pages 131
 - numeric order 131
- database 30, 118
 - backup and recovery 120
 - code page support 119
 - DB2 dependency 30
 - location 120
 - ODBC drivers 105, 119
 - ODBC drivers (AIX) 98
 - ODBC drivers (HP-UX) 99
 - ODBC drivers (Sun Solaris) 101
 - ODBC drivers (Windows NT) 104
 - requirements 118
 - support 105
- DB2 106
- Debugger
 - solving problems using 54
- domain 17
- dynamic routing 44
 - message flow 44
- dynamic topics
 - ACLs 90

E

- error handling 134
 - use of local logs 135
- error handling in message flows 47
- events 13
- execution group 48
- export message set 59
- exporting resource definitions 21

F

- finance flow
 - business scenario 37

H

- header
 - message 58
- Help 22
- HP-UX installation
 - disk space requirements 98
 - software requirements 98

- HP-UX installation
 - delivery media 166
 - hardware requirements 98
 - system requirements 98

I

- IBM Business Partners 184
- IBMMQSI2 129
- IBMPrimitive nodes
 - check 52
 - compute 51
 - database 52
 - extract 51
 - filter 52
 - FlowOrder 50
 - Label 50
 - MQeInput 50
 - MQeOutput 50
 - MQInput 50
 - MQOutput 50
 - MQReply 50
 - NEONFormatter 51
 - NEONMap 52
 - NEONRules 51
 - NEONRulesEvaluation 51
 - NEONTransform 51
 - publication 51
 - reset content descriptor 52
 - RouteToLabel 51
 - SCADAInput 51
 - SCADAOutput 51
 - throw 53
 - trace 53
 - trycatch 52
 - warehouse 52
- import message set 59
- importing resource definitions 21
- IMS/ESA 3
- information on the Internet
 - complementary offerings 184
 - MQSeries family libraries 184
 - MQSeries products 184
 - MQSeries SupportPacs 184
- input terminal 14
- inter-enterprise message sets 59
- internally defined messages 58

L

- legacy message formats 58
- license agreement
 - DB2 on AIX 98
 - DB2 on HP-UX 99
 - DB2 on Sun Solaris 101
 - DB2 on Windows NT 104
 - MQSeries Integrator 105
- Lotus Domino 3

M

- message 55
 - client access 61
 - Control Center definition 18, 55
 - creating with SmartGuide 18
 - dictionaries 61

- message 55 (*continued*)
 - domain 17
 - headers 67
 - importing 18
 - message flows 41
 - message set 18
 - MQRFH and MQRFH2 mapping 149
 - NEON definition 56
 - NEON definitions 18
 - order 70
 - parsing 19, 59, 140
 - persistence 72, 82
 - predefined 18, 55
 - processing in message flow 58
 - self-defining 19, 57
 - set 17
 - template 17, 56
 - using messages 61
 - using templates 61
 - wire format 57
 - within transactions 71
 - XML 57
- message dictionary
 - using with clients 61
- message flow 41
 - assigning to brokers 48
 - business process rules 13
 - common node characteristics 45
 - contents 41
 - creating 14
 - default 53
 - definition 41
 - dynamic routing 44
 - enriching message content 45
 - error handling 47
 - examples 16
 - exception list 47
 - execution group 48
 - for IVP 53
 - input 15
 - interaction 43
 - message order 42
 - message processing node 13, 45
 - output 15
 - parallel processing 42
 - processing messages 46
 - publish/subscribe 15, 49
 - routing 44
 - solving problems using the
 - Debugger 54
 - subflow 41
 - supplied 53
 - supplied defaults 53
 - supplied verification 53
 - throughput 42
 - tracing 136
 - transformation 43
 - tuning for performance 138
 - unit of work 42
 - using the DLQ 48
- message header 58
 - MQRFH 58
 - MQRFH2 58
- message header parser
 - MQCFH 60
 - MQCIH 60
 - MQDLH 60

- message header parser (*continued*)
 - MQIIH 60
 - MQMD 60
 - MQMDE 60
 - MQRFH 60
 - MQRFH2 60
 - MQRMH 60
 - MQSAPH 60
 - MQWIH 60
 - SMQ_BMH 60
- message headers
 - MQRFH 67
 - MQRFH2 67, 69
- message parser 19, 59, 140
 - adding 62
- message parsers
 - default 59
- message processing node
 - adding 54
 - common node characteristics 45
 - creating new 139
 - enhancing 54
 - IBMPrimitives 50
 - input 46
 - input node 15
 - MQeInput 15
 - MQInput 15
 - MQOutput 15
 - output 46
 - output node 15
 - primitives 14
 - publication node 15
 - SCADAInput 15
 - SCADAOutput 15
 - Message Queue Interface 23, 66
- message set
 - assigning to brokers 48
 - assignment 18
 - deployment 18
 - export 59
 - for IVP 53
 - import 59
- message sets
 - inter-enterprise 59
- message template 14, 56
- messages
 - internally defined 58
 - legacy formats 58
- Microsoft Exchange 3
- migration 31
 - MQIntegrator 143
 - MQSeries Integrator Version 1 143
 - MQSeries Publish/Subscribe 147
- mqbrasgn 122
- mqbrdevt 122
- mqbrkrs 122
- mqbrops 122
- mqbrtpic 122
- MQeInput node
 - in example message flows 17
- MQeMbMsgObject 58
- MQeMsgObject 58
- MQeOutput node
 - in example message flows 17
- MQRFH 67
 - mapping to MQRFH2 149
 - message header 58

- MQRFH2 67
 - mapping to MQRFH 149
 - message header 58
 - MQSeries
 - AMI 3
 - business integration 3
 - clusters 117
 - MQI 3
 - trusted applications 137
 - MQSeries Everyplace
 - special considerations 25
 - MQSeries Everyplace applications 25
 - MQSeries Everyplace publications 181
 - MQSeries family
 - business integration 4
 - MQSeries Integrator Version 2.0.2 3
 - MQSeries messaging 3
 - MQSeries Workflow 4
 - MQSeries Integrator on the Internet 184
 - MQSeries Integrator publications 181
 - national language 182
 - platform-specific 181
 - MQSeries Integrator Version 1 migration
 - backing up files 143
 - considerations for installation 143
 - enhancing rules and formats 145
 - existing rules and formats 144
 - MQSeries Integrator Version 2.0.2
 - message flow 145
 - preserving data 144
 - run-time tasks 144
 - uninstalling 144
 - user exits 146
 - MQSeries Integrator Version 2
 - enhancements 5
 - upgrading from previous products 5
 - MQSeries Integrator Version 2.0.2
 - applications 22
 - broker 11
 - business integration 5
 - components 9
 - Configuration Manager 9
 - Control Center 20
 - dependencies 29
 - database 30
 - MQSeries 29
 - getting started 6
 - package contents 165
 - Tour 22
 - User Name Server 27
 - MQSeries Publish/Subscribe
 - migration 147
 - checklist 163
 - heterogeneous networks 161
 - independent networks 160
 - migrating a network 162
 - MQSeries support 147
 - product differences 148
 - content-based filtering 159
 - default topic routing 157
 - message formats 148
 - metatopics 157
 - stream authority 154
 - streams 151
 - streams and migration 154
 - streams and neighbor brokers 153
 - subscription points 158
 - MQSeries Publish/Subscribe
 - migration 147 (*continued*)
 - product differences 148 (*continued*)
 - throughput 159
 - topics 156
 - wildcards 156
 - scenarios 147
 - MQSeries resources
 - application use of queues 70
 - MQSeries Workflow 4
 - multilevel wildcard 83
- ## N
- National Language Support 105
 - NEON
 - accessing rules and formats 146
 - existing rules and formats 144
 - message definitions 18
 - rules and formats 144
 - NEONRules and NEONFormatter Support
 - code page support restrictions 106
 - NEONRules and NEONFormatter Support publications 181
 - nodes 46
- ## O
- ODBC
 - drivers 119
 - drivers on AIX 98
 - drivers on HP-UX 99
 - drivers on Sun Solaris 101
 - drivers on Windows NT 104
 - online Help 22
 - online Tour 22
 - Order Mode
 - By Queue Order 70
 - By User ID 70
 - Order Mode property 70
 - output terminal 14
- ## P
- parsers 59
 - partner flow 37
 - Partner Offerings 184
 - PDF (Portable Document Format) 182
 - planning
 - database resources 109
 - MQSeries Integrator resources 107
 - MQSeries resources 108
 - naming conventions 107
 - plug-ins
 - guidelines 139
 - message parser 140
 - message processing node 139
 - Portable Document Format (PDF) 182
 - predefined messages 17
 - primary security domain 123
 - principals 27, 121
 - IBMMQSI2 129
 - mqbrasgn 122
 - mqbrdevt 122
 - mqbrkrs 88, 122
 - principals 27, 121 (*continued*)
 - mqbrops 122
 - mqbrtpic 122
 - problem determination 134
 - commands 136
 - database logs 137
 - database messages 137
 - messages 136
 - MQSeries events 137
 - MQSeries logs 137
 - MQSeries messages 137
 - ODBC traces 137
 - optional traces 135
 - service trace 135
 - UNIX syslog 135
 - user trace 135
 - user traces 135
 - Windows NT event log 135
 - problems
 - solving using the Debugger 54
 - publication access 91
 - publications
 - MQSeries Everyplace 181
 - MQSeries Integrator 181
 - publish/subscribe
 - event information 76
 - global 78
 - local 78
 - retained 76, 81
 - state information 76
 - retained 77
 - publish/subscribe 75
 - ACLs 28
 - collectives 86, 110
 - default ACL 29
 - default subscription point 16
 - filters 80
 - interactions with broker 75, 85
 - messages 112
 - multiple topics 85
 - publication node 15
 - reserved characters 83
 - security 86
 - special characters 83
 - subscription point 16
 - throughput 49
 - topic 23
 - topic-based security 27, 112
 - topic root 28
 - topics 82
 - unnamed publication node 16
 - wildcards 83
- ## Q
- quality of service 72
- ## R
- resource definition
 - import and export 21
 - retained publications 76, 77, 81
 - performance implications 78
 - roll back 14
 - routing
 - message flow 44

rules 3

S

SAP/R3 3

SCADA

- quality of service 72
- security 73
- special considerations 25

SCADA applications 26

SCADAInput node

- in example message flows 17

scenario

- business 33
- retail 33

security

- ACLs 89, 130
- applications 73, 129
- configuration 127
- Control Center roles 128
- database 128
- domain 121
- message flows 129
- operational 127
- primary security domain 123
- principals 121
- public access 87
- run-time 127
- SCADA 73
- topic-based 130
- trusted security domain 123

security domains

- UNIX 124
- Windows NT 123

security subsystem

- principals 27

self-defining messages 17

single-level wildcard 84

SmartGuide for message creation 18

softcopy books 182

solving problems

- using the Debugger 54

special characters

- topic level separator 83

stock flow 37

subscription options

- new publications only 81
- publish on request only 81

subscription point

- default 16

subscription points 79

- default 80
- using 80

subscriptions

- content filter 79
- local 81
- registration 78
- subscriber queue 79
- subscription point 79

Sun Solaris installation

- delivery media 167
- disk space requirements 100
- hardware requirements 100
- software requirements 100
- system requirements 100

superuser, IBMMQSI2 129

Supply Chain Management 4

SupportPac 184

SYSTEM.BROKER.CONTROL.QUEUE 69,
70

system management 13, 138

system requirements 95

AIX installation 96

HP-UX installation 98

summary 95

Sun Solaris installation 100

Windows NT installation 102

wire format 17

X

XA

commit 14

roll back 14

XA technology 14

XML messages 57

T

template 14, 17

throughput

message flow 42

publish/subscribe message flow 49

Tivoli 13

topic access 91

topic-based security 86, 112

topic root 24

topic tree 24

topics

publish/subscribe 82

semantics 84

wildcards 85

Tour 22

trace commands 136

transactionality 46

transactions 71

database interactions 71

transformation

message flow 43

trusted security domain 123

U

User Name Server 27, 112

MQSeries resources 116

multiple 112

principals 86

V

version control

SupportPac IC04 21

W

wildcard

multilevel 83

single-level 84

wildcards

multi-level 85

single-level 85

Windows 2000 xii

Windows NT installation

database 104

delivery media 168

disk space requirements 102

hardware requirements 102

MQSeries 103

operating system software 103

software requirements 102

system requirements 102

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 44-1962-816151
 - From within the U.K., use 01962-816151
- 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 the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

GC34-5599-02



Spine information:



MQSeries[®] Integrator

MQSeries Integrator Introduction and Planning

Version 2.0.2