WebSphere Business Integration Pub/Sub Solutions

JMS scenarios showing high message rate and high availability

Brokers using persistent and real-time transports

Pub/Sub architectures using WBI Event Broker

Saida Davies
Suresh Amujuri
Rudyanto Linggar
Kishore Muthyala
Ramesh Surisetty
Ewan Withers

Redbooks

ibm.com/redbooks

International Technical Support Organization

WebSphere Business Integration Pub/Sub Solutions

May 2004

SG24-6088-00

Note: Before using this information and the product it supports, read the information in "Notices" on page ix.

First Edition (May 2004)

This edition applies to:

- Windows 2000 Service Pack 3 or later
- IBM DB2 Universal Database Enterprise Server Edition 7.2 Fixpack 9 or IBM DB2 Universal Database Enterprise Server Edition 8.1 Fixpack 3
- IBM WebSphere MQ Version 5.3.0.5 and v5.3.0.6 (V5.3 with CSD05 and CSD06)
- Java Runtime Environment 1.3.1 or later
- Microsoft Data Access Components (MDAC) Version 2.7 SP1, Version 2.7 SP1a, or Version 2.8
- IBM Agent Controller Version 5.0.1
- WebScale Distribution Hub Version 2.0.0.1
- WebSphere Business Integration Event Broker and WebSphere Business Integration Message Broker Version 5.0.3 (V5.0 with CSD03)

Contents

Notices

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®	ibm.com®	SupportPac™
DB2 Universal Database™	MQSeries®	SP1®
DB2®	Parallel Sysplex®	WebSphere®
HACMP™	Redbooks™	z/OS®
IBM®	Redbooks (logo) ™	

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Preface

The first part of this IBM® Redbook provides an overview of various technologies needed for the development of Publish/Subscribe applications in IBM WebSphere Business Integration Event Broker (within IBM WebSphere Business Integration Message Broker) using Java™ Message Service (JMS).

The second part of this publication documents a business case scenario which demonstrates high message rate and high availability handled by IBM WebSphere Business Integration Event Broker using both Java Message Service using Internet Protocol (JMS-IP) and Java Message Service using Websphere Message Queue (JMS-MQ). In this part the advantage of multicasting over unicasting is demonstrated and a set up for cloned brokers which handles high availability is also demonstrated. The setup for multi brokers for high message rate is also demonstrated.

Appendix A, "Code used in the business case scenario" on page 223, contains the JMS source code used in the business case scenario.

Appendix B, "Additional material" on page 235, contains the softcopy downloadable code used in the business case scenario.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Hursley Center.

Saida Davies is a Project Leader at the International Technical Support Organization (ITSO). She is a certified senior IT specialist and has fifteen years of experience in IT. Saida has published several Redbooks™ on various business integration scenarios. She has experience in the architecture and design of WebSphere® MQ solutions, has extensive knowledge of IBM's z/OS® operating system and a detailed working knowledge of both IBM and Independent Software Vendors' operating system software. In a customer-facing role with IBM Global Services, her role included the development of services for WebSphere MQ within the z/OS and Windows® platforms. This covered the architecture, scope, design, project management and implementation of the software on stand-alone systems or on systems in a Parallel Sysplex® environment. She has a degree in Computer Science and her background includes z/OS systems programming.

Suresh Amujuri is software engineer working for Miracle Software Systems. Inc. India. He has four years of experience in the IBM WebSphere MQ family products. His major areas of expertise include Java, J2EE, .NetFramework, DB2®, Enterprise Application Integration. He worked for Global Parts Management System for Delphi USA. He is a WebMethods certified specialist in B2B.

Rudyanto Linggar is a senior Web developer for Sampoerna PrintPack in Indonesia. He has developed a corporate Internet banking Java-based Web application product on WebSphere Application Server, DB2 Universal Database™ and WebSphere MQ. This product connects AS400 core banking product via WebSphere MQ. Rudyanto has more than six years of experience building information systems using several programming languages, predominantly Windows. His areas of expertise include database administration, WebSphere Application Server administration and Administration for Windows. Rudyanto is an IBM Certified Specialist IBM for WebSphere Application Server Advanced Edition V3.5 and IBM Certified Solutions Expert DB2 UDB V7.1 Database.

Kishore Muthyala is a Software Consultant working for Miracle Software Systems Inc., India. He has two years of experience in the IBM WebSphere MQ family products. His major areas of expertise include Java, J2EE and Enterprise Application Integration. He has worked with a wide range of EAI tools including Interchange Server, WebMethods and Neon.

Ramesh Surisetty is a Senior Java Consultant for Miracle Software Systems Inc., India. He has over five years of experience in the IT industry with various technologies. His areas of expertise include Enterprise Application Integration using Websphere Business Interchange Server, WebSphere MQ, J2EE, asynchronous messaging and WebSphere development. He has developed an intranet application using ColdFusion and SQLServer 2000 for JPMorgan Chase Bank, New York.

Ewan Withers is a Software Engineer and an IBM Accredited Senior IT Specialist, currently working in the IBM Hursley WebSphere Business Integration Development Pub/Sub Team. He has seven years of experience with the WebSphere Business Integration product family, three as a consultant and the previous four as a developer and tester. He has completed numerous successful highly technical customer engagements drawing on his skill set of WebSphere MQ, JMS, WebSphere MQ Everyplace, WebSphere Business Integration Event Broker, WebSphere Business Integration Message Broker, J2EE and the WebSphere Application Server. Ewan has played a key role in providing updates, further discussions and demonstrations of additional technologies, reviewing and re-structuring this publication.

Figure 1   The authors: (Left) Suresh, Ramesh, Saida, Kishore, Rudyanto; (below) Ewan

The IBM Redbook team would like to thank the following people for their guidance, assistance and contributions to this project, especially with WebSphere Business Integration Message Broker and WebSphere Business Integration Event Broker product installation and configuration:

Gerry Reilly, Carmel Palti, Jens Diedrichsen, John Overstall, Cerys Giddings, Jonathan Woodford, Sam Rogers, Rob Convery, Rhys Francis Alasdair Paton

and Chris Harris
IBM Hursley

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

`ibm.com/redbooks/residencies.html`

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  `ibm.com/redbooks`

- Send your comments in an Internet note to:

  `redbook@us.ibm.com`

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HZ8 Building 662
  P.O. Box 12195
  Research Triangle Park, NC 27709-2195

Part 1

Introduction

This publication is divided into two parts, a theoretical part and a practical part.

Part 1 explains the background to Publish/Subscribe messaging and WebSphere Business Integration Event Broker, which is the backbone of an IBM Pub/Sub solution. It introduces Pub/Sub concepts and the WebSphere Business Integration product family, and discusses WebSphere Business Integration Event Broker technologies that power Pub/Sub solutions.

This part is divided into the following chapters:

- Chapter 1, “Project overview” on page 3.
- Chapter 2, “Transport and messaging technologies” on page 5, starts from the beginning, covering basic messaging paradigms, the Java Message Service (JMS) Application Programming Interface (API) and the underlying transports that they rely upon, including unicast and multicast networking, and point-to-point and Pub/Sub messaging. This chapter does not go into detail for all subjects, as there are many other good publications in this area (these are referenced where appropriate), although multicast messaging is covered in detail.

- Chapter 3, “The WebSphere Business Integration product family” on page 19, contains a brief overview of the WebSphere Business Integration product family, and then concentrates on the Pub/Sub provision and the implementation of the technologies covered in Chapter 2, “Transport and messaging technologies” on page 5.
- Chapter 4, “Choosing the right Pub/Sub solution” on page 39, discusses the choices facing the architect when creating a Pub/Sub messaging infrastructure from a set of business requirements. This chapter discusses the various types of requirements, how they affect each other, and how this maps into an eventual Pub/Sub design. However, this document does *not* provide detailed performance analysis of the various messaging technologies.
- Chapter 5, “Configuration and tuning of WBI Event Broker” on page 47, builds on the information provided in the previous two chapters to provide a *cookbook* for building a WebSphere Business Integration Event Broker infrastructure to match a set of business requirements.

1 Project overview

The purpose of this publication is to coherently document the use of WebSphere Business Integration Event Broker as a Pub/Sub provider.

There are many new technologies in version 5.0, and this document shows how these can be harnessed by JMS applications for greater performance or reliability. This document shows the reader how to design and deploy WebSphere Business Integration Event Broker solutions in a way that satisfies his organization's requirements.

The scope of this publication is to demonstrate the following:

- The configuration possibilities of WebSphere Business Integration Event Broker v5.0, to provide degrees of reliability and performance that match business requirements
- The programming of JMS publish and subscribe applications, to show both the de-coupling of the JMS application from the messaging provider, and the coding of JMS applications to take advantage of high availability provider facilities

Part 1 introduces the concepts of Publish/Subscribe (Pub/Sub) and the WebSphere Business Integration product family, and explains the technologies that power Pub/Sub solutions. It describes unicast and multicast transports, and also covers requirements satisfaction using Pub/Sub technologies and explains how WebSphere Business Integration Event Broker provides high availability (using cloned brokers) and high message rate in Pub/Sub solutions.

Part 2, “Rationale of WBI Event Broker” on page 63, contains installation and configuration details for WebSphere Business Integration Event Broker and introduces Pub/Sub scenarios in which WebSphere Business Integration Event Broker is tailored to meet different sets of requirements. A simple scenario and applications are used for all the different configuration demonstrations, and these are described in detail. The chapters that follow discuss the configuration of WebSphere Business Integration Event Broker in single and multiple broker scenarios to satisfy performance and reliability requirements in the infrastructure. In particular, technologies such as multicast, cloned brokers, broker collectives and message persistence are demonstrated to meet the various business requirements.

2 Transport and messaging technologies

This chapter starts from first principles and explains networking and messaging paradigms. Although there is much here that may not be new to the reader, it is necessary to cover the basics, as there are recent technologies that are occasionally not understood. The later chapters rely on the reader having a good understanding of the separation between the different levels of technology discussed below.

The first and second sections cover low level networking, which is onerous, but this is required background for understanding multicasting. Building on this networking base, the different messaging paradigms are introduced. The final section describes Java Message Service (JMS) application programming.

2.1 Basic networking

The foundation of networking technology is the network protocol. This section provides a very brief overview of basic low level networking using Internet Protocol (IP), which is the adopted industry standard.

2.1.1 Internet Protocol (IP)

IP is the lowest level of networking, and is responsible for providing a rule set for moving blocks of data from node to node in the network. Packets are directed using a four-byte destination address, known as the IP address. Ranges of addresses are assigned to different organizations by the Internet authorities. IP operates on gateway machines at different levels of the IP range hierarchy to route data either internally within an organization, or externally from server to server.

There are two main versions of IP currently available. The most widely used is IP Version 4 (IPv4), which provides basic networking support. However, support for IP version 6 (IPv6) is now becoming standard in some networking hardware and software, and has the following benefits:

- IPv6 was designed as a superset of IPv4, and therefore IPv6 systems communicate with IPv4 systems. This allows staged and independent upgrades of network resources in different organizations.
- The limit on the length of an IP address is 32 bits in IPv4. IPv4 and IPv6 both place an upper limit on the number of IP addresses available. IPv6 changes the IP address maximum length to 128 bits, allowing many more addresses and the corresponding growth of the Internet.
- IPv6 offers direct support for unicast, anycast and multicast addressing (these are explained below).

2.1.2 Transmission Control Protocol (TCP)

TCP is a protocol built on top of IP to ensure correct delivery of data (hence, TCP/IP). The data is actually delivered over the IP layer, but this is prone to error, so TCP was developed to encapsulate the IP data into *packets*. These TCP packets provide a basis for acknowledgement, and support error detection in order to trigger repeated transmissions until the data is correctly received.

2.1.3 User Datagram Protocol (UDP)

UDP is also a protocol implemented on top of IP (hence, UDP/IP), but unlike TCP it does not provide error detection and correction facilities. Instead, it allows applications direct access to the data service of the IP layer. This flexibility allows UDP to offer a greater range of transmission options, such as multicast (see below), which are not available in TCP. However, UDP applications are more complex because they must deal with the error detection that is transparently available to TCP applications.

2.2 Advanced networking

This section briefly covers the different transmission methods that are available using the IP transport. These methods are now directly supported by the latest networking hardware and software.

2.2.1 Unicast

Unicast network communication involves data transfer between a single sender and a single receiver. This is the standard method used with Transmission Control Protocol/Internet Protocol (TCP/IP), where a sender and a receiver are each identified by a unique IP address and a TCP port, and the sending of data outside of this relationship is an error. If data is to be sent to multiple receivers, the sender must transmit the data to each receiver separately. Figure 2-1 on page 8 illustrates unicast.

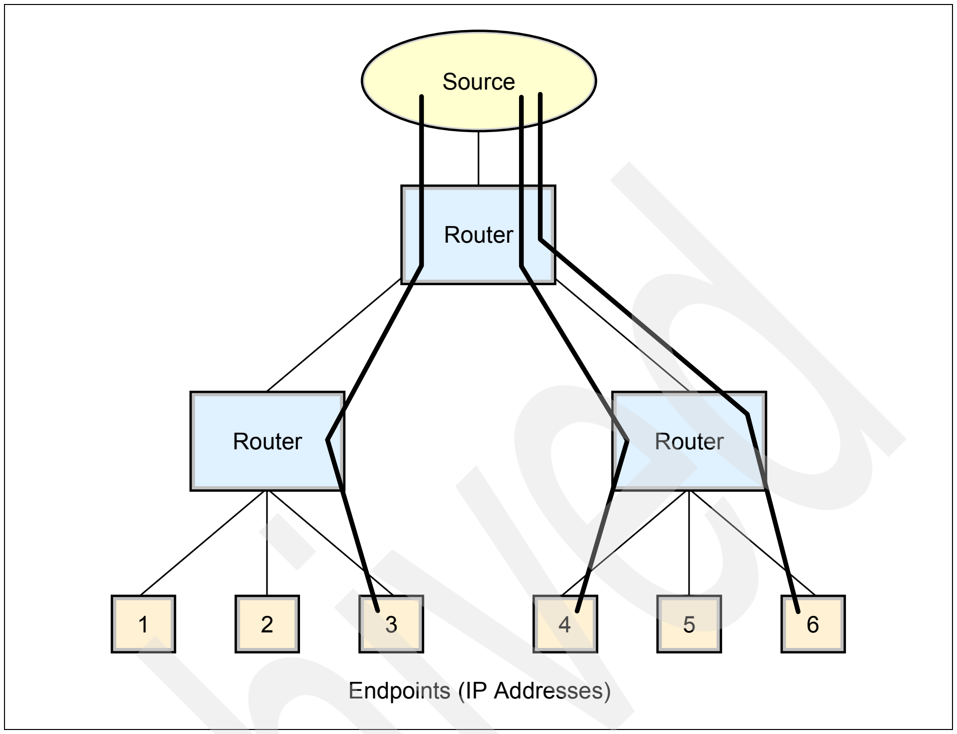


Figure 2-1 Unicast

As seen in the figure, sending data to receivers 3, 4 and 6 requires three separate data transmissions from the source.

2.2.2 Anycast

Anycast network communication is still data transfer between a single sender and a single receiver, but the receiver is determined to be any one of several receivers within a group. It is mainly designed to allow the efficient updating of router tables within a server network and is not relevant to this discussion, although it has been included for completeness.

2.2.3 Broadcast

Broadcast network communication is data transfer between a single sender and multiple receivers. In this case, data sent is received by *all* other addresses on the network, and although it is simple to use, it can over-utilize both network bandwidth and the runtime of applications that have received superfluous data. Figure 2-2 on page 9 illustrates broadcast.

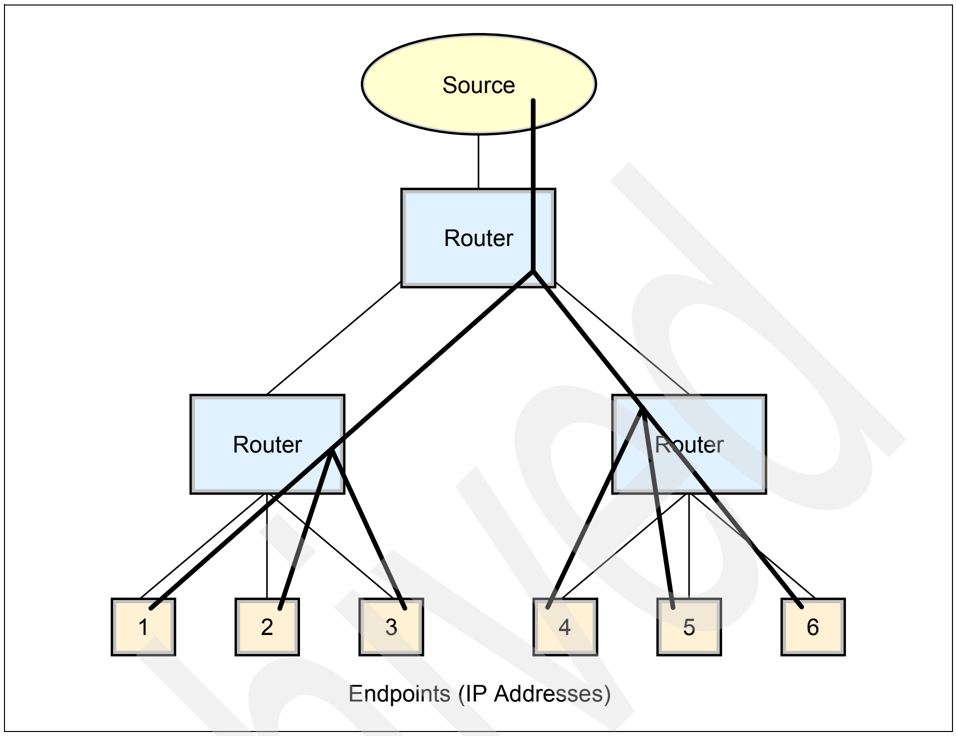


Figure 2-2 Broadcast

As seen in the figure, sending data to receivers 3, 4 and 6 only requires one data transmission from the source, but all receivers will receive the data, whether required or not.

2.2.4 Multicast

Multicast network communication is also data transfer between a single sender and multiple receivers, but is more discriminate than broadcast. Partitioning of receivers is achieved using a *multicast group*, which is an IP address selected from a dedicated IP range of 224.0.0.0 to 239.255.255.255. The sender opens an outbound socket to the designated multicast group, and the receivers open inbound sockets to the same multicast group. After the group has been joined, the network itself distributes every packet sent to the multicast group by the sender to all joined group members. Therefore, only receivers that have registered an interest in the correct multicast group will receive the transmitted data, so it provides the benefits of broadcast while minimizing network traffic. Figure 2-3 on page 10 illustrates multicast.

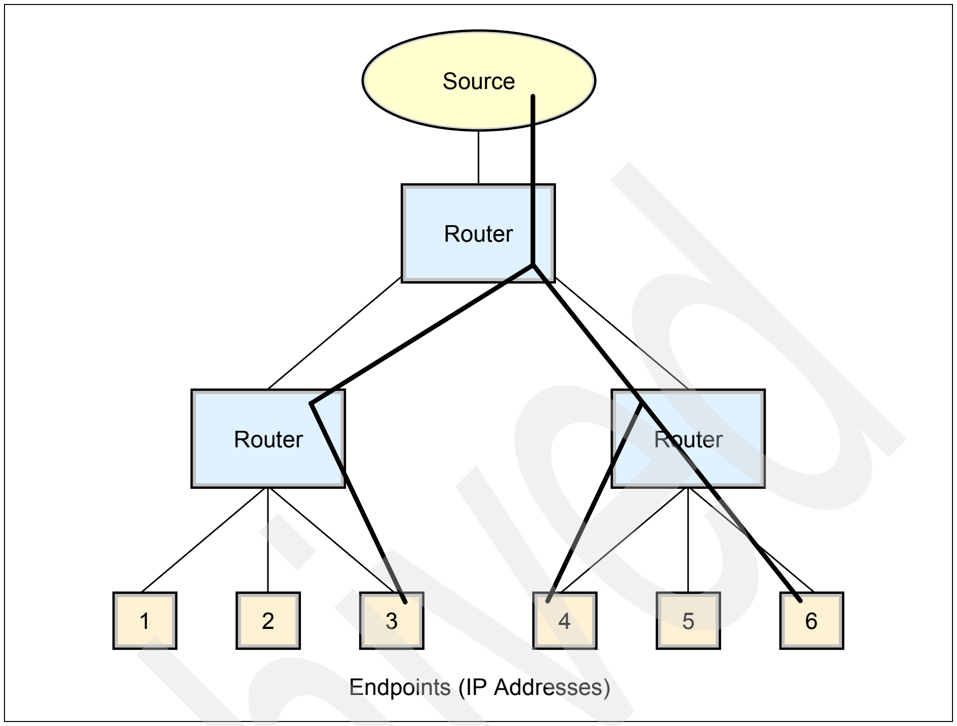


Figure 2-3 Multicast

As seen in the figure, sending data to receivers 3, 4 and 6 only requires one data transmission from the source, and only receivers 3, 4 and 6 will receive the data.

2.3 Unicast versus multicast

The question of whether unicast or multicast is superior is a difficult one to answer. As is usually the case in questions like this, the answer depends on what the technology is to be used for. Both unicast and multicast have strengths and weaknesses because of their fundamental paradigms, and although multicast is an excellent solution for business requirements like large fan-out, it is not the best solution in all situations. This is why it is very important to understand the underlying business requirements before choosing even the most basic components of the solution.

Multicast offers significant benefits in the following situations:

- Large fan-out: This is where it is required to distribute data to a large number of receivers while making the most efficient use of the network, and also ensuring equitable delivery times to all receivers. Multicast offers a scalable solution to this requirement regardless of the number of receivers, and there is little extra bandwidth used than in a unicast environment where a single sender transmits the data to a single receiver. Also, if unicast is used in this case, the first receiver on the sender's list receives the data ahead of all the others, giving an unfair advantage. This is not the case with multicast; all receivers get the data simultaneously.

- Small fan-out: This is where most or all of the available network bandwidth is used by a small number of consumers, such as a high-performance Web site where a front-end Web server distributes work to a number of back-end servers. If a large load is distributed to the back-end servers over unicast, the network will reach saturation quickly if more back-end servers are required. If multicast is used, additional servers can be added to the back-end without significantly impacting the network traffic, allowing greater scalability.

However, multicast is not always the optimal transport. It works most efficiently when the sender is physically close to the receivers (as few as possible routers between them in networking terms), and best of all when they are in the same or neighbouring Local Area Network (LAN) segments. So, the more widely spaced the senders and the receivers are, the less applicable multicast becomes as a networking solution. Also, multicast is not the best choice if there is only ever one receiver interested in the sender's data.

Network routers, as alluded in the previous paragraph, are instrumental in the way that multicast is dealt with on the wire. Multicast packet headers contain a user-configurable field called *time to live* (TTL), which controls the number of routers that the packet can pass through before being discarded. The setting of this number obviously has extreme ramifications. If it is set too low then the data is unlikely to reach all its destinations, and if it is set too high then the message may be sent to routers that have no subscribed receivers. The implication of this point is that multicast is not an easy technology to deploy, especially if the network is used for conventional unicast too. It is very important for architects and administrators to understand the topology of the network in order to correctly design and manage the messaging system.

Note: WebSphere Business Integration Event Broker has enhanced reliability for multicast. This is discussed further in "Reliable Multicast Messaging" on page 30 and "Configuring Reliable Multicast Messaging (RMM)" on page 31.

2.4 Messaging

Messaging technology and software was invented to solve the problems inherent in application communication, namely volatile networks, platform heterogeneity, and communication synchronicity. IBM's successful messaging software, WebSphere MQ, began development (as MQSeries®) in the early 1990's.

WebSphere MQ was developed to include the following features:

- Assured delivery: Message persistence and transactional control allowed business critical applications to depend on messaging where before they could not due to its inherent unreliability.
- A single API: On all 35+ platforms supported by the WebSphere MQ brand, it has a common interface that hides all platform-specific complexities from the programmer.
- Asynchronous communication: Applications sending and receiving messages do not have to be online at the same time to communicate using messaging.

This solved the business need at the time, and although still relevant today, understanding of messaging and what it can provide as a technology has expanded enormously. In particular, the relationship between message sender and receiver has changed, as the following two sections describe.

2.4.1 Point-to-point

Point-to-point (comparable to unicast networking) involves the sending of data by one application to a defined and unique receiving endpoint application. In other words, there is a mathematical bijection describing the relationship between the sender and the receiver. WebSphere MQ has traditionally excelled at providing this type of messaging, and until fairly recently most messaging technology was based in this area.

2.4.2 Publish/Subscribe

Publish/Subscribe (comparable to multicast networking) involves the sending of data by one application to any number of receivers. This introduces a much more complex relationship between the sender and the receiver, where there is an abstraction between them so that they are unaware of each other. This abstraction layer is usually provided by a broker service, which manages incoming data (*publications)* from sending applications (*publishers)* and matches them to interested receivers *(subscribers)*. The broker then forwards the data to the registered subscribers.

The correlation between publishers and subscribers is defined by using *topics*, which are contained in a hierarchical arbitrary namespace (that is, application defined). Subscribers register interest in a topic (or topics) with the broker, and publishers publish messages using those topics. The names of topics usually relate to the information carried within them. The broker routes publications on particular topics to all those subscribers that have registered an interest in those topics. This is best illustrated in the figure below.

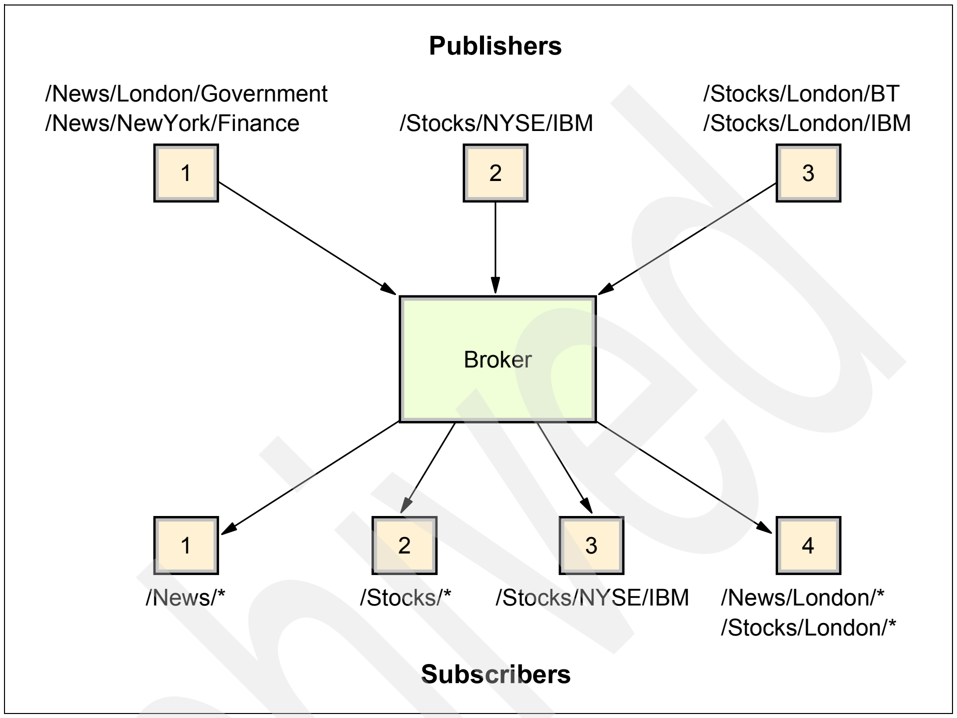


Figure 2-4 Example Pub/Sub architecture

The figure shows the following:

1. Publisher 1 publishes a message on to /News/London/Govt. The broker routes the publication message to subscribers 1 and 4.
2. Publisher 1 publishes a message on to /News/NewYork/Finance. The broker routes the publication message to subscriber 1 only.
3. Publisher 2 publishes a message on to /Stocks/NYSE/IBM. The broker routes the publication message to subscribers 2 and 3.
4. Publisher 3 publishes a message on to /Stocks/London/BT. The broker routes the publication message to subscribers 2 and 4.
5. Publisher 3 publishes a message on to /Stocks/London/IBM. The broker routes the publication message to subscribers 2 and 4.

The figure also clarifies several important points about topics and how they relate publishers and subscribers together:

- The only restriction within the topic-space is that the structure is hierarchical. Beyond this, topic names within a level in the hierarchy are arbitrary. However, it is useful to give meaning to the hierarchy by naming the topics in a way that is indicative of the data that will be published on them.
- Subscribers can subscribe to a single topic (subscriber 3), multiple topics, a single sub-tree of the topic hierarchy (subscribers 1 and 2), or multiple sub-trees of the hierarchy (subscriber 4). Topic-space sub-trees are specified by inserting a wildcard into the topic identifier. A subscriber that subscribes using a wildcard will receive all publications that fall under that partition of the topicspace.
- A publication is published to a single topic. That is, wildcards are not permitted when publishing data, although a publisher can choose to publish messages to multiple different topics (publishers 1 and 3).
- A publisher can publish messages on a topic to which there are no subscribers. In this case, the broker will usually discard the messages, although there are some cases that this may not happen. The point emphasized here is that the publishers and subscribers are completely de-coupled from each other.

2.4.3 A conceptual paradigm view

Another way of looking at the differences between point-to-point and Pub/Sub is to identify who has the responsibility of defining how the message flow is organized and driven.

In the point-to-point paradigm, messages are *pushed* from the sending side, and the sending application decides which applications will receive the messages.

However, in the Pub/Sub paradigm, the senders and the receivers are de-coupled, and therefore the senders have no knowledge (and no control) of who receives their messages. In this situation the receivers decide what information they receive, and therefore *pull* the messages from the source, subject to their authorization for receiving them.

Examples of message push and pull can be found in many places and on several levels. For instance, message pull is used extensively in wireless networking so that all the messages for one application can be received in a batch rather than as a trickle feed that would require a longer connection time. It is therefore important to remember that all the descriptions above are at the conceptual level and may not bear resemblance to the underlying technology.

2.5 JMS application programming

This section covers the basics of the JMS API, but does not go into detail. There are many sources of information on JMS programming. For example:

- The JMS Specification

  http://java.sun.com/products/jms/docs.html
- *WebSphere MQ Using Java,* SC34-6066

2.5.1 Introduction to JMS

Java Message Service (JMS) is a Java API for messaging. It is part of the Java 2 Platform, Enterprise Edition (J2EE) specification, and from J2EE 1.3, an implementation from a J2EE-compliant application server is required. The specification is owned by Sun and is managed as part of the Java Community Process by an Expert Group (IBM being one of the founding members). When viewed on its own, JMS provides a vendor-independent API for writing messaging applications in Java. The API itself is abstract, and numerous implementations are made available by JMS providers, one of which is implemented by IBM using WebSphere MQ.

Note: Although JMS guarantees provider independence, it does *not* guarantee provider interoperability. This is because although the JMS specifies an API, it does not specify a standard wire format. So JMS applications can be ported from one JMS provider to another, but they must *all* be ported at the same time. This is because they must all use the same underlying provider-specific communications layer if they are to talk to each other.

The benefits of JMS are as follows:

- Both point-to-point and Pub/Sub messaging models are available, and are both mapped to the same simple underlying structure.
- Applications written correctly in JMS can be moved between messaging providers, and therefore the burden of infrastructure and platform knowledge is removed from the developer.
- High-level message classes remove the complexity of byte-level data management.
- Message selection allows receivers to filter incoming messages based on logical constructs.
- Asynchronous message delivery (callback when a message is available) is provided as an option.

- There is a choice of transactional semantics: Client-controlled acknowledgement, locally transacted, or external transaction coordination. This is explained further in “Transactional control” on page 52.

2.5.2 JMS class structure

The JMS class hierarchy is illustrated in the following figure.

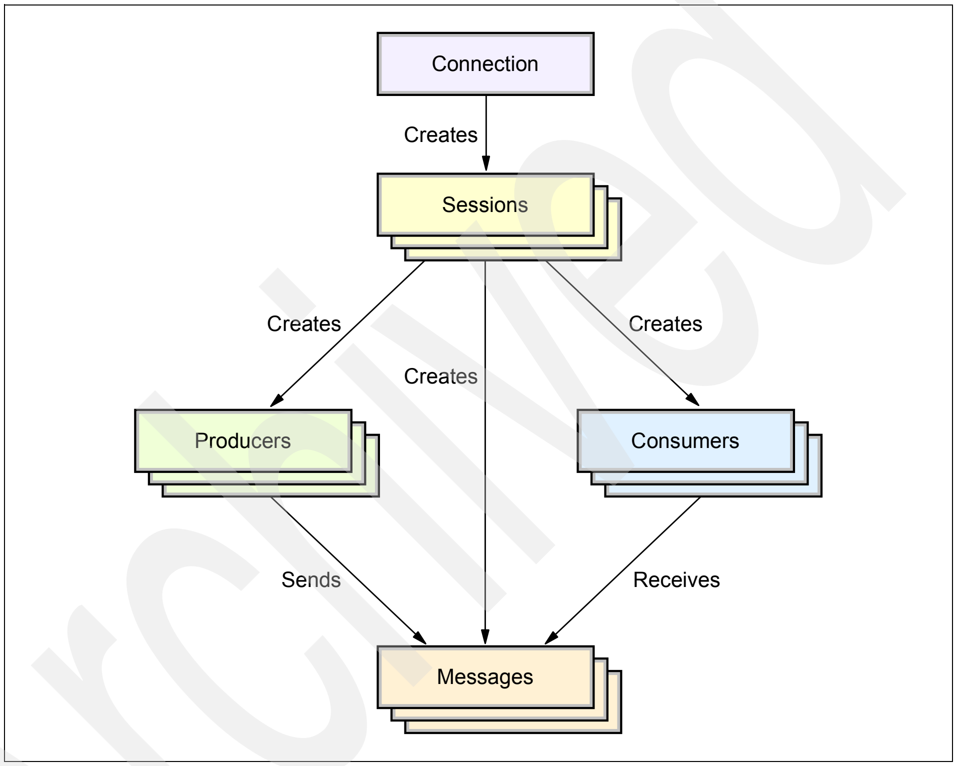


Figure 2-5 JMS class hierarchy

This figure shows the runtime classes:

- The Connection (subclassed by QueueConnection and TopicConnection) is a multi-threaded object, and encapsulates a connection to the JMS provider. It is used to create Sessions.
- The Session (subclassed by QueueSession and TopicSession) is the most important class in the JMS hierarchy. It is a single-threaded object, and provides the context for creating messages as well as defining transaction scope and message ordering. It is also used to create *Producers* and *Consumers*.

- The Producer (subclassed by QueueSender and QueueReceiver) is a single-threaded object and is used for sending messages.
- The Consumer (subclassed by TopicPublisher and TopicSubscriber) is a single-threaded object and is used for receiving messages. JMS Consumers can also be created with a selector, which is a logical construct used as a template to match messages against. The consumer is only then delivered messages that match its selector.
- The Message object is a Serializable object that is a superclass for the five JMS message types. These are TextMessage, BytesMessage, StreamMessage, MapMessage and ObjectMessage.

Examples of JMS code (sample application code used in Part 2, “Rationale of WBI Event Broker” on page 63) are available in Appendix A, “Code used in the business case scenario” on page 223.

2.5.3 JMS Administered Objects

The previous section described the JMS runtime objects, but some configuration is needed in order to both store and mask provider-specific configuration and state information. This is purposely kept separate from the runtime (hence the application programs) to enforce the provider-independence of the JMS API. In order for JMS providers to implement JMS functions, two sets of abstract classes are provided that are designed for storage in a Java Naming and Directory Interface (JNDI) namespace. These are:

- The Destination (subclassed by Queue and Topic) encapsulates the concept of a message *target,* where messages are sent to and retrieved from. When creating a Producer or a Consumer it is usual to provide a Destination object that has been looked up from its serialized form in a JNDI namespace.
- The ConnectionFactory (subclassed by QueueConnectionFactory and TopicConnectionFactory) encapsulates the concept of the engine or manager that controls the messaging framework. This is also stored in a JNDI namespace, and once retrieved is used as a basis for creating Connections.

The way of coding JMS Pub/Sub applications is to store Topic and TopicConnectionFactory definitions in a JNDI namespace, and to retrieve them at runtime. In this way, both the JMS provider and the target messaging systems themselves can be changed without impacting the application code. The only information that the application needs is the JNDI connection information and the lookup keys for the JMS objects required.

It is possible to directly create ConnectionFactory and Destination objects in the application, but this is not recommended, because not only is the code then tied to the names of the underlying messaging components, but also to the JMS provider. This is because the administered object classes above are all abstract; they are implemented by the JMS providers independently, and it is these implementation classes that must be instantiated in a program. Using JNDI as a storage medium hides this and allows the abstract classes to be referenced in the code.

2.5.4 JMS coding hints and tips

This section covers some general best JMS coding practices. These are for any JMS provider and are not specific to WebSphere MQ.

- The JMS specification contains some areas of loose definition. Provider implementation can (and is allowed to) vary in these areas. There are also some counter-intuitive areas in the API and these must be treated with care. Use the JMS specification as a guide when writing JMS code and do not rely solely on a provider's documentation.
- JMS Sessions are single-threaded objects. They must not be accessed from multiple threads simultaneously and this rule extends to child objects, such as Producers and Consumers. Access to these objects must always be serialized.
- Always close JMS objects when shutting down an application. JMS objects usually contain or own resources not under the control of the JVM and therefore cannot be garbage-collected automatically. Typically, an object must be closed from within the class that created or instantiated it. This rule extends to the JNDI context, which is often overlooked.
- The JMSException, thrown on error, usually contains useful information from the JMS provider. This is held in the embedded LinkedException, and taking note of this data can make tracking down problems much easier.
- JMS Connections have the facility for registering an ExceptionListener. This is an application-provided piece of code that implements the *onException* method. Exceptions that cannot be thrown to application code will be sent here, and corrective action can be triggered. If an ExceptionListener is not registered, then the Exception information is lost. This is especially important when using asynchronous delivery.

3

The WebSphere Business Integration product family

This chapter introduces some of the WebSphere Business Integration product family and the way it supports business integration in an organization. It then concentrates directly on the Pub/Sub provision and implementation within the products.

3.1 The products

This section briefly covers the products in the WebSphere Business Integration family relevant to Pub/Sub and the content of this publication. It is not an exhaustive list, and further details of the full extent of the WebSphere Business Integration family can be found here:

`http://www.ibm.com/software/info1/websphere/index.jsp?tab=products/businessint`

3.1.1 WebSphere MQ

WebSphere MQ (previously known as MQSeries) can provide assured, once-only delivery of messages between applications. It offers common Application Programming Interfaces (APIs) and infrastructure across heterogeneous platforms and protocols, and masks differences and complexities from the programmer and the administrator. Its reliability and proven track record have made it the *de facto* industry messaging standard and its platform independence allows the integration of existing business applications.

WebSphere MQ was one of the first messaging providers to implement the JMS API, and the performance and function of the WebSphere MQ JMS implementation have continually improved since then. WebSphere MQ JMS is a mature and proven technology.

It is discussed here because, as the basis of the WebSphere Business Integration family messaging technology, it is a fundamental part of this coverage of Pub/Sub technology using WebSphere Business Integration products. Only WebSphere MQ v5.3 CSD 6 is considered in this document.

WebSphere MQ Product Extension MA0C: "WebSphere MQ Publish/Subscribe": is not covered in this publication, as its performance and scalability are not of the same standard as WebSphere Business Integration Event Broker (although it is a perfectly functional queue manager based Pub/Sub engine). It is covered in the publication *MQSeries Publish/Subscribe Applications* at the link below.

`http://www.redbooks.ibm.com/redbooks/pdfs/sg246282.pdf`

3.1.2 WebSphere MQ Everyplace

WebSphere MQ Everyplace extends business integration to mobile devices, by connecting mobile and wireless applications to the enterprise backbone with the reliable messaging that is the hallmark of the WebSphere MQ brand. It supports a wide range of devices with a small footprint, has highly customizable interfaces in both C and Java and includes support for encryption, authentication and compression for message data.

Its mobile transport integrates seamlessly with other members of the WebSphere Business Integration family and masks the problem of intermittent and fragile networks from the rest of the enterprise infrastructure.

It is discussed here because it is an important part of the WebSphere MQ product set, supports the Pub/Sub paradigm and integrates with the WebSphere Business Integration Brokers. However, it does not have any further part in this coverage of Pub/Sub technology.

3.1.3 WebSphere Business Integration Event Broker

WebSphere Business Integration Event Broker is IBM's Pub/Sub platform of choice. It facilitates the design and creation of flexible, extensible and secure infrastructures across multiple platforms that make best use of existing applications. It distributes information from multiple sources through its central broker to provide integration across the entire enterprise.

WebSphere Business Integration Event Broker can send and receive data using a variety of protocols, including WebSphere MQ, WebSphere MQ Real-time, telemetry (SCADA), mobile (WebSphere MQ Everyplace) and multicast endpoints. Incoming data is separated from its transport within the broker, so the incoming protocol does not bear any influence on the outgoing protocol. This is known as *stream-crossing* and builds huge flexibility into the Pub/Sub paradigm.

It is managed and configured using the Message Brokers Toolkit for WebSphere Studio, which is based on the Eclipse open-source framework. This allows both integrated development environments with other WebSphere products, and simple migration to the powerful transformation features of WebSphere Business Integration Message Broker (see below).

It is discussed here because it provides the most important part of this coverage of Pub/Sub using the WebSphere Business Integration products. Only WebSphere Business Integration Event Broker v5.0 CSD 3 is considered in this document.

3.1.4 WebSphere Business Integration Message Broker

WebSphere Business Integration Message Broker extends the integration possibilities of WebSphere Business Integration Event Broker by adding message transformation and enrichment, database integration, Web Service interfacing, message formatting and validation, and other functions to the basic Pub/Sub information integration provision. This data mediation allows much greater application integration, and dynamic content-based routing also makes the product extremely powerful.

It is discussed here because all that is demonstrated using WebSphere Business Integration Event Broker is also possible for customers using WebSphere Business Integration Message Broker.

3.1.5 WebSphere MQ Workflow

WebSphere MQ Workflow supports long running business process workflows and transactions as they interact with both systems and people. It is similar in the overall function to WebSphere Business Integration Message Broker but differs in that the broker transactions tend to be completely automatic and fairly short-lived, whereas workflow transactions are much longer-lived and may have direct human involvement.

The similarity with WebSphere Business Integration Message Broker implies the possibility of deep integration with other members of the WebSphere Business Integration family as well as business applications directly. It allows systems and people to be brought into a managed process integration environment for business solutions. It also integrates with the WebSphere Business Integration Modeler product for design, analysis, simulation and monitoring of business processes.

It is discussed here because it is an important part of the WebSphere MQ product set, but is not a part of this coverage of Pub/Sub technology.

3.2 WebSphere Business Integration Event Broker configuration concepts

The configuration mechanism of WebSphere Business Integration Event Broker can seem overly complex to the inexperienced. This section aims to explain the various components, their functions, and how they interact. Figure 3-1 on page 23 illustrates the descriptions of the components that follow.

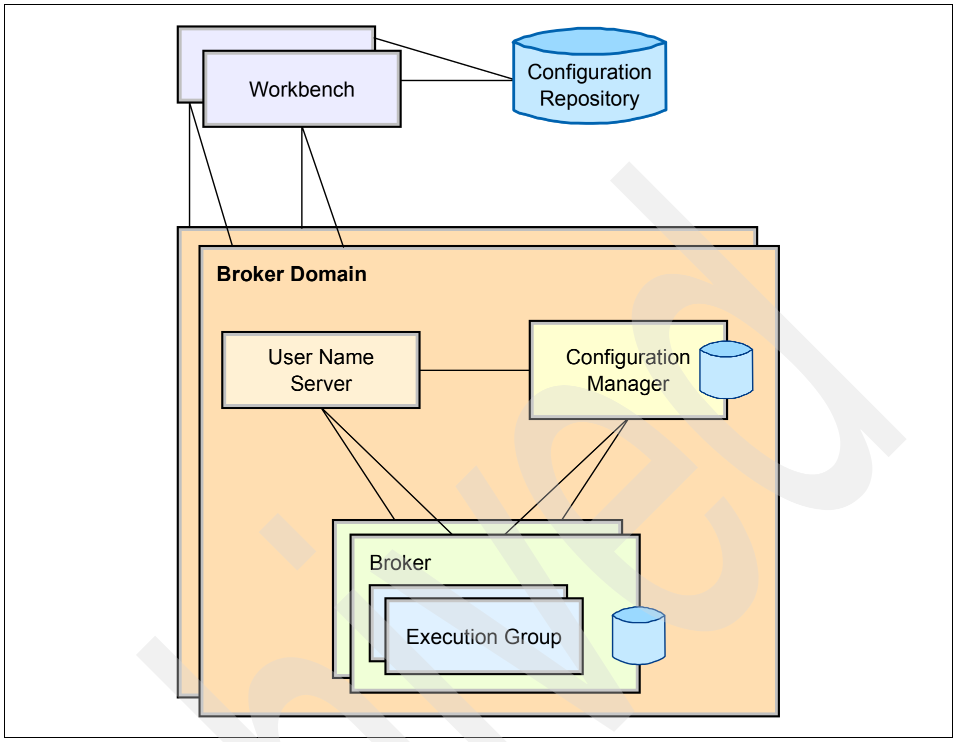


Figure 3-1 WebSphere Business Integration Event Broker architecture

Note: Configuration Repository in Figure 3-1 on page 23 denotes a code-management system for version control of flows and other user data. This is not to be confused with the Configuration Manager's own database (also sometimes referred to as the configuration repository).

3.2.1 WebSphere Business Integration Message Brokers Toolkit

The WebSphere Business Integration Message Brokers Toolkit for WebSphere Studio (workbench) is an Eclipse-based configuration utility for the WebSphere Business Integration brokers. It can interface with configuration repositories for management and version control of configuration data, and a single workbench can be used to connect to and manage multiple broker domains.

3.2.2 The Configuration Manager

The Configuration Manager interfaces between the workbench and an executing set of brokers. It provides brokers with their initial configuration, and also updates them with any subsequent changes. (These configuration delivery conversations between a Configuration Manager and a Broker are called *deployments*). It also maintains the overall broker domain configuration. The Configuration Manager is not required to be running at broker execution time, only at configuration time. There is only one Configuration Manager per broker domain, and it is only supported on the Windows platform. The Configuration Manager requires the use of a WebSphere MQ queue manager, and this can be shared with one broker in the broker domain.

The main functions of the Configuration Manager are:

- To maintain the configuration repository (a set of database tables that hold a central record of broker domain components).
- To deploy the broker topology and message processing operations in response to actions initiated through the workbench. Broker archive files are deployed through the Configuration Manager to execution groups in the brokers.
- To report current status back to the workbench, both relating to the results of deployment operations and current broker status.

3.2.3 The Broker

The Broker is the component that actually processes message data. It runs on the Windows, UNIX® and z/OS platforms and, once deployed correctly from a Configuration Manager, it runs autonomously. The Broker uses standard WebSphere MQ channels and distributed queuing to communicate with its Configuration Manager, and requires the use of a WebSphere MQ queue manager for this purpose.

At deployment time, the Configuration Manager supplies information to the Broker. This is stored locally in the Broker's database repository, and the broker constructs its message processing function to match this information. Message flows (see below) are deployed to components in the Broker called *execution groups*. These execution group processes are logically separated from each other, in terms of runtime, memory and transactional control. A Broker can have several execution groups, and each execution group can contain multiple threads of execution (where one message flow has the use of one or more threads).

3.2.4 Message flows

A message flow is a directed graph of message flow nodes that represent the actions that are performed on a message when it is received and processed by a broker. Each node in a message flow represents a processing step, and the connections between the nodes represent the direction that the flow takes, based on the outcome of each step. A message flow must include an input node that provides the source of the messages to be processed.

A message flow is created in the workbench using the supplied built-in nodes (see below). A flow is deployed to an execution group in a broker, where it runs in one or more threads from the execution group's thread pool.

The message flow nodes supplied with WebSphere Business Integration Event Broker have multiple input and output terminals that are used to define connections between the nodes and thus to define the shape of the message flow. The following list describes the built-in nodes:

- The MQInput node reads messages from a WebSphere MQ queue.
- The MQOutput node writes messages to a WebSphere MQ queue.
- The MQReply node is similar to the MQOutput node, except that the message is written to the queue specified in the ReplyTo fields of the WebSphere MQ message header.
- The Publication node publishes an incoming message to the specified topics in the Pub/Sub framework.
- The MQeInput node receives messages using the WebSphere MQ Everyplace protocol.
- The MQeOutput node sends messages using the WebSphere MQ Everyplace protocol.
- The SCADAInput node receives messages using the WebSphere MQ Telemetry protocol.
- The SCADAOutput node sends messages using the WebSphere MQ Telemetry protocol.
- The Real-timeInput node receives messages using either the WebSphere MQ Real-time or WebSphere MQ multicast protocols.
- The Real-timeOptimizedFlow node is a node that is a complete message flow in itself; it is used for high-performance Pub/Sub using either the WebSphere MQ Real-time or WebSphere MQ multicast protocols.

3.2.5 The User Name Server

The User Name Server is an optional runtime component that provides authentication of users and groups performing Pub/Sub operations. Topic-based security is one of the features of WebSphere Business Integration Event Broker, and the User Name Server enables control, using Access Control Lists (ACLs), over which users can publish and subscribe to which topics. The User Name

Server requires the use of a WebSphere MQ queue manager, which can be shared with a Configuration Manager and/or a Broker in the broker domain.

3.3 WebSphere Business Integration Event Broker Publish/Subscribe

A Pub/Sub infrastructure does not, in theory, need a central broker. However, in practice a broker is a requirement, for the following reasons:

- Brokers tend to interface with each other in a well-behaved fashion, even if applications do not; a poorly designed application can monopolize resources. Brokers are well-developed applications that integrate correctly with the underlying operating system and network, and as a result can detect and mitigate poor application behavior.
- Pub/sub applications tend to be transient or unstable. A long-lived and stable component is required to control the state and transport media, especially when considering multicast and reliability.

Having justified the requirement for a broker in the Pub/Sub arena, WebSphere Business Integration Event Broker's functions (on top of the standard topic management and message delivery responsibilities) is covered in the following sections.

Note: All of the functions covered below are available to applications written using the WebSphere MQ JMS API.

3.3.1 WebSphere Business Integration Event Broker transports

WebSphere Business Integration Event Broker can accept and send data over any of the following protocols, and can stream-cross from one to another seamlessly. This allows a high degree of granularity at design and configuration time, to allow different requirements on different parts of the system to be satisfied without compromising others:

- The traditional WebSphere MQ messages and queues can be used for both publishers and subscribers. This allows business-critical data to have the full protection of WebSphere MQ's persistence and transactional control. However, WebSphere MQ's quality of service can be down-graded as required, with corresponding increases in performance.
- The WebSphere MQ mobile transport, as used by WebSphere MQ Everyplace, can also be used for both publishers and subscribers. The benefits of this are, for instance, a large, distributed workforce can be kept up-to-date with information from one central broker.

- The WebSphere MQ telemetry transport, as used by SCADA (supervisory, control and data acquisition) devices can be used to both publish and subscribe with, allowing control of large mechanical systems such as oil pipelines.
- The WebSphere MQ Real-time transport provides very high performance synchronous IP-based communication for Pub/Sub.
- The WebSphere MQ multicast transport (Reliable Multicast Messaging (RMM), as seen below) is also supported by WebSphere Business Integration Event Broker.

Note: A WebSphere MQ JMS application, correctly written to use JNDI lookup for the administered objects, does not need to be changed in order to use the different transports. It is a case of changing the characteristics of the TopicConnectionFactory and Topics in JNDI instead. The examples in Part 2, “Rationale of WBI Event Broker” on page 63, of this publication aim to show this.

3.3.2 WebSphere Business Integration Event Broker cloned brokers

WebSphere Business Integration Event Broker offers the facility to clone the subscription registrations of a broker across one or more other brokers. The WebSphere MQ queue managers underlying the brokers must all be interconnected, and the brokers themselves must each be informed of all other brokers they are cloned with. When this is completed, any subscription received by a broker is replicated on all the other brokers, so if a broker terminates then its subscribers are still served by the other cloned brokers. WebSphere MQ queues are the only transport that can be used for cloned broker subscriptions (although publishers to cloned brokers can use any of the supported transports). Figure 3-2 on page 28 illustrates a cloned broker system.

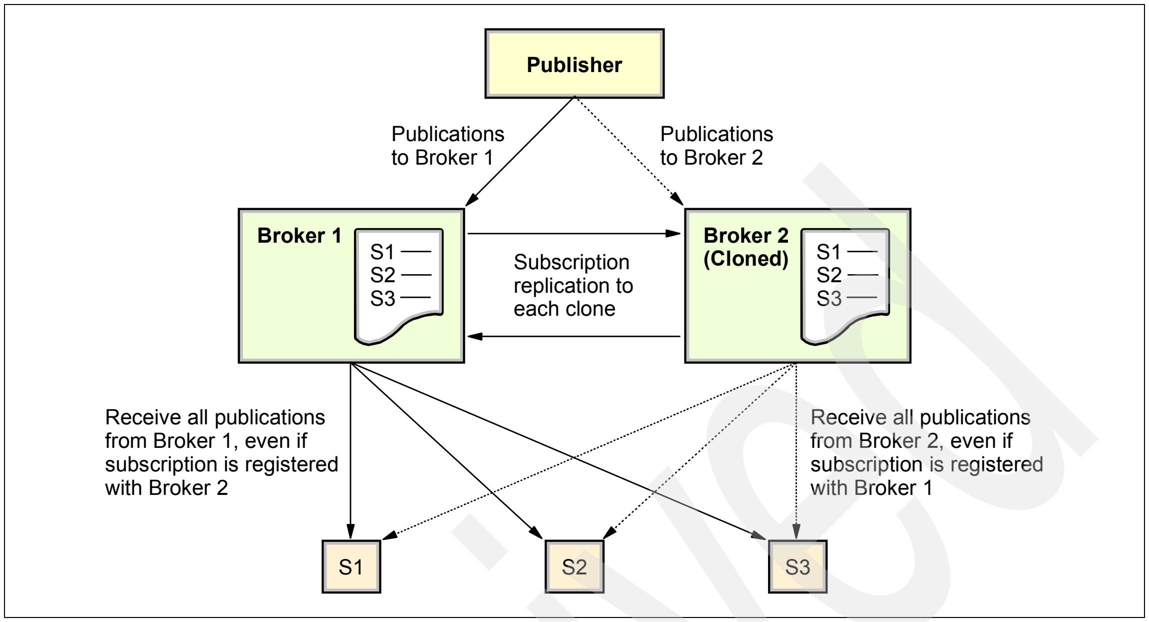

Publisher
Publications to Broker 1
Publications to Broker 2
Broker 1
S1
S2
S3
Subscription replication to each clone
Broker 2 (Cloned)
S1
S2
S3
Receive all publications from Broker 1, even if subscription is registered with Broker 2
Receive all publications from Broker 2, even if subscription is registered with Broker 1
S1
S2
S3

Figure 3-2 A cloned broker system

As seen in the figure, having cloned brokers improves the availability of a Pub/Sub system. By defining cloned brokers on different systems, it can be ensured that publications are delivered to subscribers even when one of the systems is unavailable.

WebSphere MQ queues are the only transport that can be used for cloned broker subscriptions. Also, if two brokers within a collective are clones, duplicate messages might be sent to subscribers registered with brokers inside that collective.

3.3.3 WebSphere Business Integration Event Broker collectives

A WebSphere Business Integration Event Broker collective is a group of brokers that have been configured to share publications and subscriptions. It differs from a cloned system in that brokers do not replicate subscription information to each other. Instead, brokers in a collective subscribe to each other's publications. A client normally connects to its nearest broker for performance reasons, and the broker receives all messages that match the subscription registration of the client from all brokers within the collective. Figure 3-3 on page 29 illustrates a broker collective.

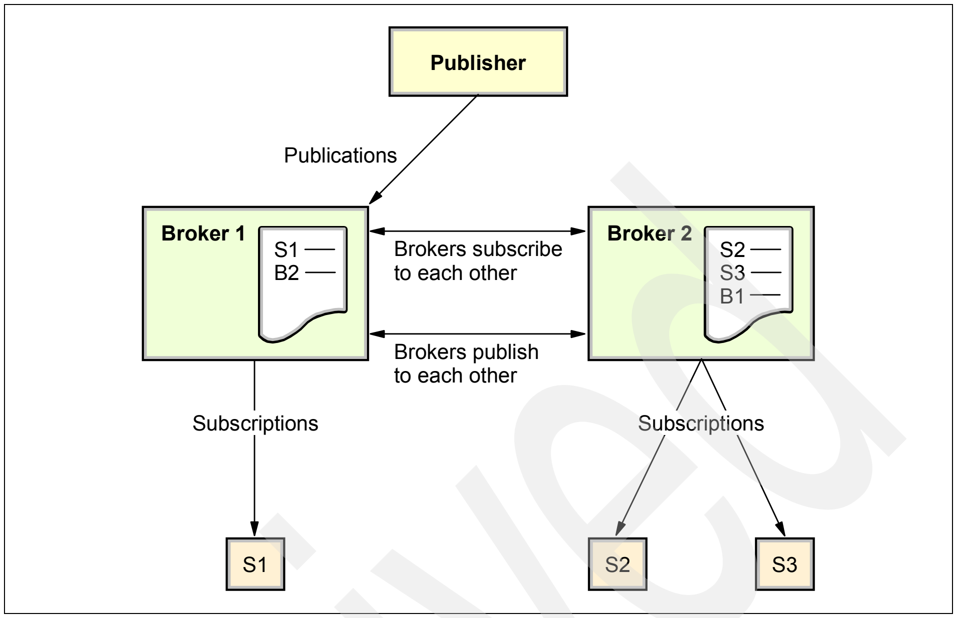


Figure 3-3 A broker collective

As seen in the figure, there are two subscribers registered with Broker 2. For each topic that these subscribers register an interest, Broker 2 subscribes to Broker 1 (at the most once, no matter how many subscribers Broker 2 has for a particular topic). When Broker 1 receives a publication on that topic, it publishes it to Broker 2, which in turn publishes it to all its registered subscribers.

Collectives exhibit the following behavior:

- Messages destined for specific brokers in the same collective are transported directly to those brokers and do not need to pass through any intermediate brokers. This improves broker performance and optimizes inter-broker Pub/Sub traffic, when viewed in contrast with a hierarchical tree configuration.
- Collectives can be joined together (by connecting one broker in each collective together). This is a good solution for where clients are widely dispersed across several locations, because it optimizes the flow of publications and subscriptions registrations through the network.

3.3.4 Multicast Pub/Sub

When considering the multicast transport and the Pub/Sub messaging paradigm, it is obvious that there is a parallel between the two models, which further implies that the latter can exploit the former to great effect. To this end, support for multicast has been included in the WebSphere Business Integration Event Broker Pub/Sub framework, bringing the following benefits:

- The centralized management structure that allows all messaging transports to be configured and monitored from a single console.
- A corollary to the previous point is the fact that this single management of transports allows a separation of the data management, and hence a unified security approach. This allows user access control at the level of the topic, as opposed to the level of the underlying connection.
- As stated in an earlier section, publisher and subscriber applications that are correctly written in a provider-independent fashion in JMS (and *only* JMS) can utilize multicast without the need for any re-coding.
- A brokered multicast solution ensures good behavior on the network, by ensuring data is correctly paced for the available LAN bandwidth. This further ensures application equity. In other words, all applications have an equal share of the network resources as they are required. Contrast this to a standalone multicast application, which may have no concept of its own network usage in comparison to other applications, and may flood the network to such an extent that IP stability is compromised.

By enabling a broker and one or more of its topics as being capable of multicast, it is possible to ensure that a publication on one of these topics can be received at the same time by all multicast subscribers on the same subnet as the broker. (Multicast subscribers on any one subnet receive publications simultaneously, but subscribers on the same subnet as the broker may receive publications ahead of other subnets.) Also, the number of multicast subscribers can be increased without significant network resource overhead, as only one message is sent regardless of the number of subscribers.

In addition, it has been stated earlier that it is not desirable or optimal to multicast across a wide range of routers. WebSphere Business Integration Event Broker provides a solution to this by connecting two brokers over the separation, therefore allowing the two multicast architectures at either end to communicate without using multicast over the long network hops.

3.4 Reliable Multicast Messaging

A well-known limitation of the standard IP multicast is the lack of reliability, due to the fact that the transport uses a best effort delivery mode over UDP with none of the error detection available in TCP. Therefore, if there are network glitches or heavy load, packets can be lost with no chance of recovery. The multicast implementation within WebSphere Business Integration Event Broker addresses this problem with a multi-layered architecture, and is known as Reliable Multicast Messaging (RMM).

RMM delivers a robust scalable solution for multicast, including the following features:

- Enhanced delivery performance, provided by RMM's unique method of message-to-packet mapping of hundreds of thousands of messages per second.
- Highly reliable multicast over existing networks supports acknowledged and persistent delivery.
- Client and network monitoring enables traffic control and bandwidth management of the transport.
- Good network behavior and data delivery fairness with other transports (including TCP).
- Low-level network concepts are masked and abstracted away from the JMS programmer.
- A patented client-existence detection mechanism ensures messages are only transmitted if there are active receivers.
- Flexible transmission modes include unreliable streaming for real-time data and other information that does not require delivery to be guaranteed.

RMM offers distinct advantages over standard multicast, ensuring the best use of network resources by balancing resource usage, performance and reliability against each other in the best possible way.

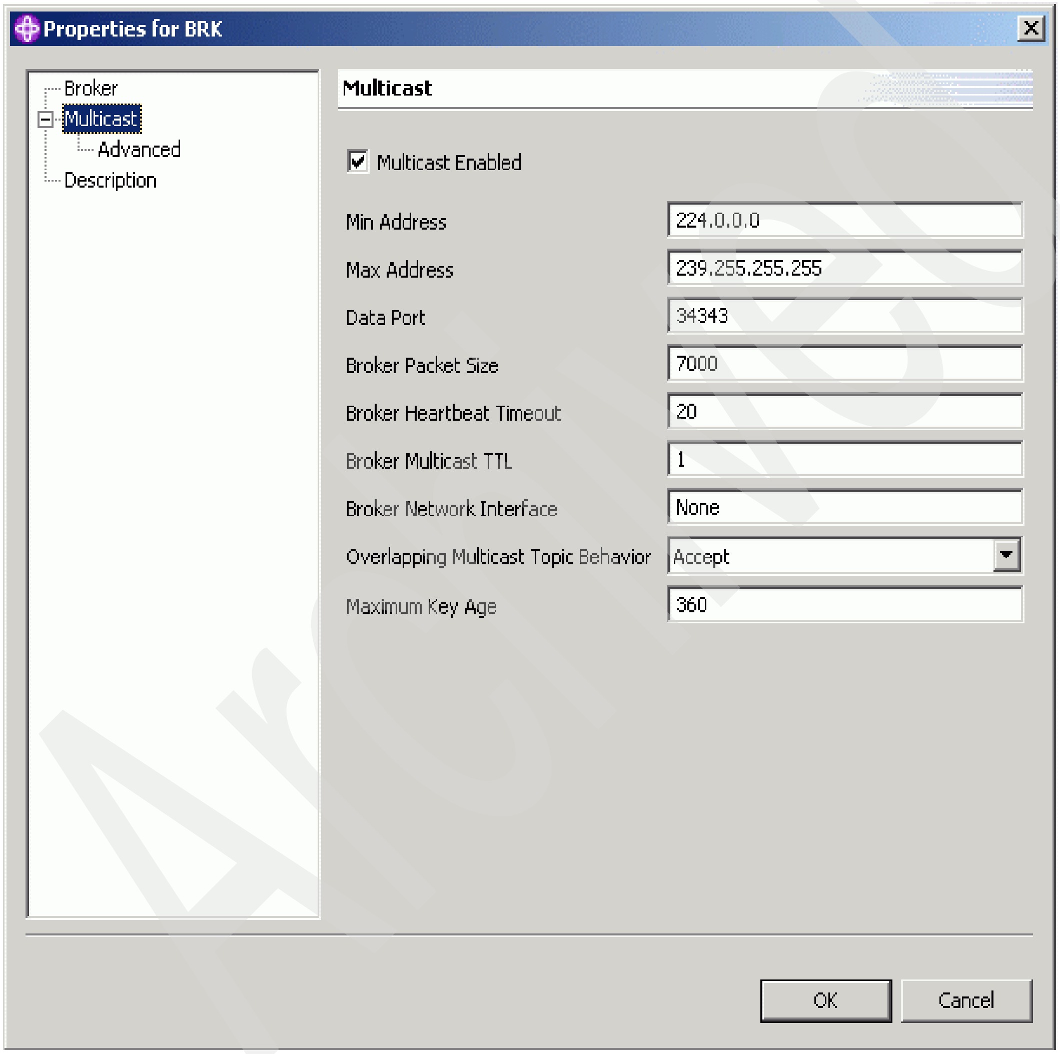
3.5 Configuring Reliable Multicast Messaging (RMM)

This publication does not go into full detail about the configuration of WebSphere Business Integration Event Broker, except for the sample business case scenarios in Part 2, “Rationale of WBI Event Broker” on page 63, where step-by-step illustrations are provided. An exception has also been made for RMM, as it is not a well understood technology at present. Only broker configuration using the workbench is considered, but the reader must be aware that the command **`mqsichangeproperties`** can be used to configure the broker directly.

Note: Any changes made using the **`mqsichangeproperties`** command are lost if the configuration is redeployed from the workbench.

3.5.1 Brokers and RMM

In order to make a broker capable of handling multicast requests, bring up the broker's properties (by right-clicking the broker in the Broker Topology and selecting **Properties**). Select the **Multicast** tab, and the following panel is shown.



[✓] Multicast Enabled

Min Address	224.0.0.0
Max Address	239.255.255.255
Data Port	34343
Broker Packet Size	7000
Broker Heartbeat Timeout	20
Broker Multicast TTL	1
Broker Network Interface	None
Overlapping Multicast Topic Behavior	Accept
Maximum Key Age	360

Figure 3-4 Broker multicast properties

First, select **Multicast Enabled**. This enables all the remaining multicast properties. These are:

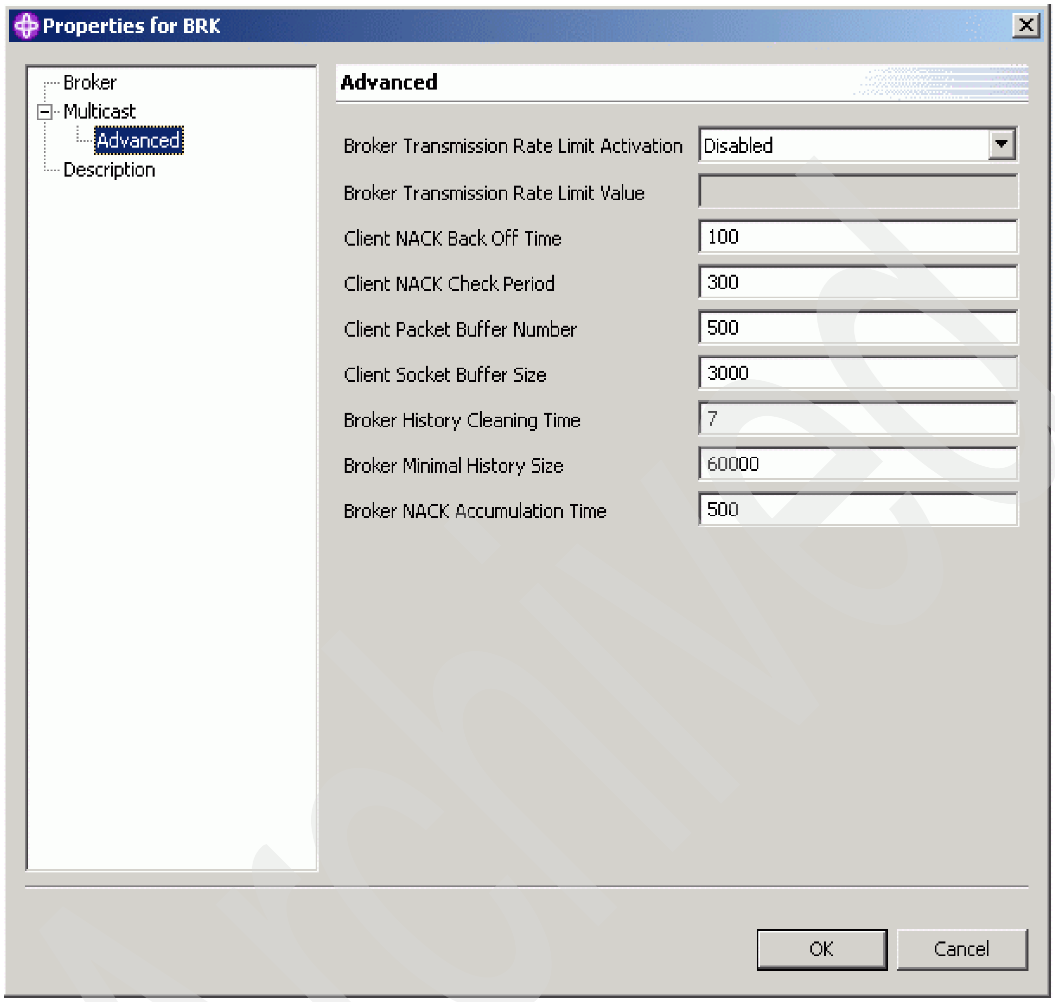
- *Min Address* is the lowest IP address that the broker can use for multicast. This must be in the multicast range of 224.0.0.0 to 239.255.255.255, and the default value is 224.0.0.0.
- *Max Address* is the highest IP address that the broker can use for multicast. This must be in the multicast range of 224.0.0.0 to 239.255.255.255, and the default value is 239.255.255.255.
- *Data Port* is the UDP data port through which multicast packets are sent and received. The default value is 34343.
- *Broker Packet Size* is the size (in bytes) of multicast packets. This can be in the range 500 to 32000, and the default value is 7000.
- *Broker Heartbeat Timeout* is used by clients to detect a transmitter or network failure. The broker sends a control packet periodically to each client, which is used to send various control information (including this value). If a client detects that a control packet does not arrive within a certain number of seconds (twice this value) from arrival of the previous control packet, then the client can assume a transmitter or network failure. The default value is 20.
- *Broker Multicast TTL* is the maximum number of router hops that a multicast packet can make between the broker and a client. This can be in the range 1 to 255, and the default value is 1 (ensuring multicast packets do not pass through any routers).
- *Broker Network Interface* is the name of the network interface over which multicast packets are transmitted (only relevant when the broker is running on a host with more than one network interface). This can be a host name or an IP address. The default is None (leaving selection to the operating system).
- *Overlapping Multicast Topic Behavior* is used to control the behavior of the broker when a client requests a multicast subscription for a topic that is part of a topic hierarchy containing topics that are explicitly disabled for multicast. There are three choices: *Accept* (the default), Reject or Revert. If Accept is set, a matching multicast subscription is accepted and all publications matching the topic are multicast, except those specifically excluded. If Reject is set, a multicast subscription to a topic with children that are disabled for multicast is rejected. If Revert is set, subscriptions to a topic that is disabled for multicast (or has children that are disabled for multicast) results in unicast transmission.

This is best illustrated by example: Consider a topic T1 that has two sub-topics, T2 and T3. T3 is not enabled for multicast, whereas the other two are.

  – If Accept is set, a multicast subscription to T1/# (where # is a wildcard) receives messages published on T1/T2 over multicast, but does not receive any messages published on T1/T3.
  – If Reject is set, multicast subscriptions to T1/# are rejected.
  – If Revert is set, a multicast subscription to T1/# receives messages published on both T1/T2 and T1/T3, but the messages are sent over unicast.

- *Maximum Key Age* is the maximum age (in minutes) of a topic encryption key (as set by encrypted topics) before redefinition is required. The default value is 360.

There are also some advanced properties. Selecting **Multicast -> Advanced** shows the following panel (Figure 3-5 on page 35).



Advanced

Broker Transmission Rate Limit Activation	Disabled
Broker Transmission Rate Limit Value	
Client NACK Back Off Time	100
Client NACK Check Period	300
Client Packet Buffer Number	500
Client Socket Buffer Size	3000
Broker History Cleaning Time	7
Broker Minimal History Size	60000
Broker NACK Accumulation Time	500

Figure 3-5 Advanced broker multicast properties

The properties in this panel are as follows:

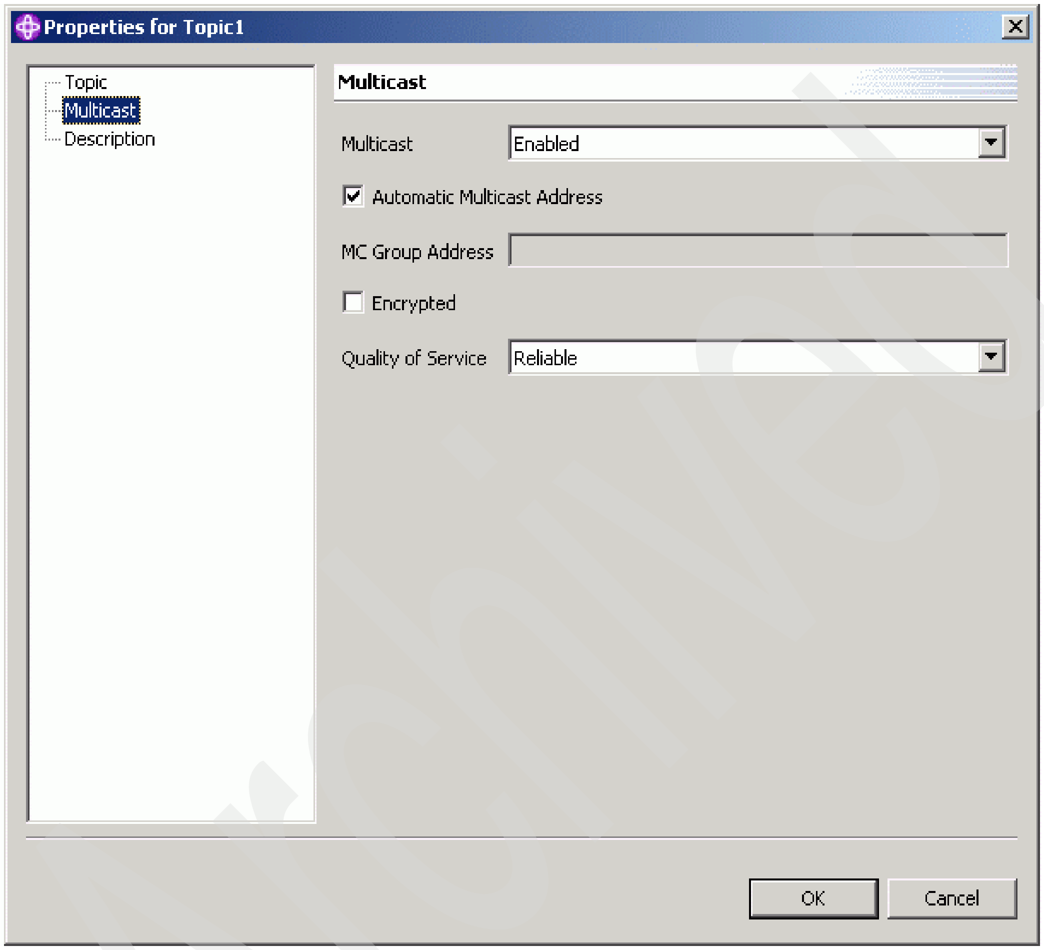
- *Broker Transmission Rate Limit Activation* can be used in conjunction with the following parameter to control network congestion. Possible values are Disabled (the default), Static or Dynamic. If set to Disabled, multicast data is transmitted as fast as possible and the hardware limits define the maximum rate. If set to Static, multicast data is transmitted as fast as possible and the value of the next parameter defines the maximum rate. If set to Dynamic, the rate is controlled by reports from clients, but never exceeds the value in the following parameter.

- *Broker Transmission Rate Limit Value* limits the overall transmission rate (in kilobits per second) of multicast packets. This parameter is used only if the above property is set to Static. This parameter must not exceed the capabilities of the machine or network, or this transmission rate can never be reached. This value can be in the range 10 to 1000000.
- *Client NACK Back Off time* is the maximum time (in milliseconds) that a client listens for another's Negative ACKnowledgement (NACKs) before sending its own NACK. This value can be in the range 0 to 1000, and the default value is 100.
- *Client NACK Check Period* is the time (in milliseconds) between periodic checks of reception status and sequence gap detection for NACK building. This value can be in the range 10 to 1000, and the default value is 300.
- *Client Packet Buffer Number* is the number of memory buffers that are created for packet reception. Having a high number improves performance and minimizes packet loss at high delivery rates, but uses more memory. Each buffer is 33 kb; therefore, 500 buffers (the default) uses approximately 15 Mb of memory. If memory use is at a premium, try using different values for this parameter to balance memory usage versus performance at high transmission rates. This value can be in the range 1 to 5000.
- *Client Socket Buffer Size* is the size (in kilobytes) of the client's socket receiver buffer, and increasing it leads to lower loss rates as fewer packets are dropped by the client receiver (although this uses more memory). This value can be in the range 65 to 10000, and the default value is 3000.
- *Broker History Cleaning Time* is the time (in seconds) defined for cleaning the transmitted-packet archive buffer. This value can be in the range 1 to 20, and the default value is 7.
- *Broker Minimal History Size* is the minimum size (in kilobytes) of the transmitted-packet archive buffer, which is shared by all reliable topics and can be used to recover lost packets. This value can be in the range 1000 to 1000000, and the default value is 60000.
- *Broker NACK Accumulation Time* is the time (in milliseconds) that NACKs are aggregated in the broker before recovered packets are sent. This value can be in the range 50 to 1000, and the default value is 500.

Note: The broker must be restarted for any deployed changes to take effect.

3.5.2 Topics and RMM

Once a topic has been created, to use it for multicast subscriptions requires some additional configuration. Bring up the topic's properties (by right-clicking the topic in the Topic Panel and selecting **Properties**). Select the **Multicast** tab, and the following panel is shown.


Properties for Topic1

Topic
Multicast
Description

Multicast

Multicast: Enabled

[✓] Automatic Multicast Address

MC Group Address:

[ ] Encrypted

Quality of Service: Reliable

Figure 3-6 Topic multicast properties

This provides access to the following properties:

- *Multicast* is set to either enable or disable multicast for a topic hierarchy. Values are Disabled, Enabled or Inherit. *Disabled* and *Enabled* are self-explanatory. *Inherit* detects and uses all values from the parent topic or topic root, and this value cannot be set on the topic root itself.
- *Automatic Multicast Address* tells the broker whether this topic has a pre-determined multicast group address (set in the field below). Default is set

to *true*, where the broker chooses a multicast group for the topic automatically.

- *MC Group Address* is only used if the previous parameter is set to *false*, and determines the multicast group address for this topic hierarchy.
- *Encrypted* can be set to true or false, and determines whether multicast traffic is encrypted.
- *Quality of Service* can be set to Reliable (the default) or Unreliable.

4

Choosing the right Pub/Sub solution

The previous two chapters discussed the technologies supporting Pub/Sub generally and the WebSphere Business Integration implementations specifically. This chapter steps away from technical specifics in order to take an objective look at the kind of business requirements that are likely to be seen for a Pub/Sub messaging system, and how to satisfy these with the Pub/Sub functions and techniques that are available.

Detailed performance analysis of the WebSphere Business Integration products is beyond the scope of this document. See the following URL for more details of WebSphere Business Integration performance reports:

http://www.ibm.com/software/integration/support/supportpacs/perfreppacs.html

This document is also not designed to be a complete guide to requirements definition, risk analysis and software development project management. Only the requirements that have an impact on the eventual Pub/Sub configuration are considered. The basis of the following discussion is that the initial requirement calls for a messaging architecture based on Publish/Subscribe.

4.1 Determining requirements

The first step in completing a successful design is to ensure that the requirements on the system are fully understood and documented. The requirements that can affect the Pub/Sub infrastructure planning have been separated into five broad sections:

- Performance requirements: These define aspects like response time, data throughput, and numbers of supported applications.
- Reliability requirements: These relate to the ability to withstand data loss or corruption.
- Availability requirements: These are similar to reliability requirements, and there is some intersection between them, but they relate more to when the system is accessible. This area can include scheduled and unscheduled down-time.
- Resource and scalability requirements: These concern the load on the underlying operating system and network components that the Pub/Sub system imparts. This area is also used to document and plan changes in the system to accommodate changes in the load upon it (numbers of connections, for instance).
- Security requirements: These document the permissible access to the system and its data, both internally and externally. This area must include a detailed break-down of the authorized users (and the consequences of a security breach) for each level of the Pub/Sub architecture.

Although there is value in separating these requirements out for consideration, it is also mandatory to consider them in relation to each other. This is because changing one attribute or property of the system, while it may satisfy one requirement, may break another. It is important to consider the limits of the software technology in this light, and also to consider how realistic the requirements are.

There is no substitute for prototyping and experimentation. Performance analyses and comparisons give some guidance, but every project, every system, and almost every message is different. Only with a working proof of concept and realistic simulated data load can it be anywhere near certain that the proposed solution can be successful.

4.1.1 Performance requirements

Performance requirements are often the most important and demanding of all the requirements. The following points illustrate the main considerations when determining performance requirements:

- Throughput, either in messages or bytes per unit time, is the most obvious and emotive indicator of system performance. What is the maximum rate that can be achieved? More importantly, what are the *required* throughput rates (at average and peak times) and is there enough headroom available in the system to accommodate backlogs if they accumulate? Product performance reports can offer significant help when planning in this area, and the measurements made tend to focus on raw data throughput.
- A Pub/Sub infrastructure is a pointless enterprise without producers and consumers of information. Therefore, the Pub/Sub infrastructure must accommodate connections for incoming and outgoing data, and the next requirement (possibly correlating with throughput) can be average and peak numbers of connections into the system. Again, product performance reports offer help in this area.
- Another influence on system performance is the introduction of bottlenecks. As the name suggests, these slow down system throughput, and therefore a good design must avoid them. However, it is often difficult to spot bottlenecks in an academic system design, so a functional prototype is important here. Even if the data throughput is a fraction of what is expected on the main system (for example, due to a smaller specification machine being used for the test), the performance of each component can still be analyzed. This helps to determine if there are points in the information flow where the data rate is constrained. Monitoring tools (such as the WebSphere Business Integration Event Broker performance statistics) must be used on the prototype to determine if there are bottlenecks.
- The final point in this list is user response time and, depending on point of view, it can be either the least or the most important. The middleware architect tends to concentrate on throughput and connectivity as requirements, and assume it is the responsibility of the application designer to deal with the user experience. The converse is also likely to be true, and therefore this requirement (heavily dependent on the collaboration of both of the people above) may drop into the cracks during the design process.

The final point is the most interesting, because it may be that being slightly slower is more desirable. For instance, consider the following two systems offering the same function:

- System 1 with a user response time of between 2 seconds and 60 seconds.
- System 2 with a user response time of between 10 and 15 seconds.

If the reader takes the place of a system user, System 2 appears more desirable because of its predictability. The erratic nature of System 1's responses, although some may be faster than those of System 2, is not conducive to productivity on the part of the user. There are many reasons why some responses may take up to 60 seconds, and it may be difficult to distinguish good reasons (for example, heavy system load) from failure. A consistent user response time, even if slower, has been shown to be preferable to erratic user response times.

This is why the application and the middleware designers must collaborate on the system design as it relates to performance. The response time of a consistently performing system can be easier to change than that of an erratically performing system, because the consistent system only needs to be considered as a whole. Consistent performance from the user's point of view can only be achieved if the application designers and the infrastructure designers work together to satisfy requirements.

4.1.2 Reliability requirements

Reliability is the property of a system that determines how it copes with errors, either from the user, from within itself, or from its underlying resources. These errors may manifest themselves in many ways, but they are often realized as missing, corrupted or inconsistent data. Requirements on the reliability of a system can be fairly obvious from the business use of the data flowing through it. For instance, a system that distributes stock ticks on a sub-second basis can have fairly low reliability requirements (as the data has a limited useful life), but a system distributing and updating business-critical information is more likely to have requirements that enforce data integrity and retention. Also in this area is the question of recovery in case of unexpected termination. Can the system recover to a consistent state when brought back online or can any data currently in transit be discarded?

4.1.3 Availability requirements

Intersecting slightly with reliability is the question of availability. Availability is the measure of how available the system is for use over time, and availability requirements specify limits on working time, idle time, and scheduled and non-scheduled downtime. These requirements vary from systems that are only required for a few hours a day to those that can only be permitted to be off-line for less than ten minutes per year. The choice of hardware and operating system depends on the availability requirements, and if the hardware and operating system cannot be changed, then increasing the availability of the system can be difficult.

Availability is a continuum: A system's availability can vary from erratic to continuous, and the availability requirements of the system must quantify this (determining up-time and down-time per week, month, and so on). Operating systems vary in their inclusion of support for high availability and, when supported, this is usually accomplished with redundancy or fail-over capability. The Pub/Sub infrastructure must be based upon the operating system with the correct availability support to satisfy the availability requirements.

Single points of failure (sometimes referred to as SPOFs) can adversely affect system availability in the same way that bottlenecks can adversely affect system performance. A single point of failure is a component in a system that can, if it fails, stop the entire system from working. The impact of single points of failure can be mitigated by adding in redundant or additional instances of the component, so that the failure of one instance does not stop the whole system.

The cross-over between availability and reliability requirements is in the area of restart and recovery. The recovery aspect can affect both data (reliability) and system up-time (availability). If a system terminates unexpectedly, it is desirable for it to be restarted as quickly as possible, but unnecessary effort must not be over expended here if the requirements do not warrant it. Restart in itself does not satisfy the reliability or availability requirements. It is the data integrity and the alacrity of the restart which addresses them. For example, if the requirements are for reliability and not availability, then concentrate on the data integrity rather than worrying about how quickly the system can be brought back online.

4.1.4 Resource and scalability requirements

Requirements relating to resource usage are those that constrain or limit the amount of resources available for the system and its applications, and can involve any aspect of the underlying infrastructure, from operating system resources (such as allocated memory and CPU time), hardware resources (such as disk space) and networking resources (such as sockets or ports).

Related to resource requirements are scalability requirements. These describe how the system responds to change over time as demands alter; this may be related to a phased deployment. Systems scale in two ways: Horizontally and vertically. Horizontal scaling is the addition of new systems into the environment as a whole, in order to increase capacity by using parallelism. Vertical scaling is the addition of extra CPUs to one machine in order to increase capacity by adding extra power to one component. The correct way of scaling the system in order to meet the needs of increased demand is determined by the current limiting factor: The performance bottleneck.

4.1.5 Security requirements

Security requirements state the necessary access permissions to system resources and data, with reasons why based on how critical they are to the business. These translate into requirements for user authentication (validating user entry to the system), session security (securing existing access that was validated at entry), and data security (securing stored system data). A detailed discussion of Pub/Sub security is beyond the scope of this publication.

4.2 Requirement conflicts

This section looks at the various requirements introduced above, and considers how they can complement and conflict with each other. Before proceeding with a Pub/Sub design there must not only be agreement on the balance between the requirements, but also on their priorities, because one must take precedence in case of conflict.

4.2.1 Reliability versus performance

The question of whether reliability or performance is more important is one that confronts every designer of every software system. The purpose of the system may well dictate an immediate and obvious answer. For instance, a Pub/Sub system for a financial institution that involves the transmission of financial transaction data requires absolute reliability, whereas a system where the data has a short useful life leans much more towards performance. However, usually the answer is not so clear cut, and there may well be different types of data in the system that require corresponding differences in reliability (known as *quality of service*).

In the same way that availability is a continuum, so is reliability versus performance. At the one end there is utter reliability where performance is barely considered, and at the other there is the need for maximum performance with some permitted data loss. There is little benefit in defining this continuum in any more detail, as there is no value in pinpointing a particular system's position on it. The value here is in understanding that there is always a trade-off between reliability and performance, and in being realistic about one when there is a non-negotiable requirement on the other. It is also important to realize that without firm requirements in both these areas, or a documented *lack* of a requirement, there is little point in proceeding with further design of the system.

The balancing of reliability and performance requires the implementation of a working prototype and some realistic simulated system load. This provides the platform for experimentation to find the best possible configuration to satisfy both sets of requirements.

4.2.2 Availability versus performance

The consideration of availability requirements may well have a positive impact on performance. The single point of failure, given its solitary nature, may well also be a performance bottleneck. If availability requirements dictate that SPOFs must be eliminated, then this could also increase throughput, due to the fact that making a component highly available usually implies adding capacity. However, this may not be the case if the component is made highly available by the utilization of monitoring and fail-over instead of redundancy.

4.2.3 Resource usage versus performance

Resource requirements, especially if driven by budget constraints or existing hardware, are very likely to come into conflict with performance requirements. Consider the following resources likely to be used by a system, and the impact upon performance if their use is constrained:

- Memory: If the memory allocated to an application or component is limited to less than it needs, then the designer needs to come up with some other method of allocating volatile storage. This can be accomplished by persisting state out to another resource such as disk, relying on the operating system to manage the situation by swapping out, or re-writing the application to use less memory. The last is probably the most effective but is not likely to be the most efficient, and both the other two solutions definitely degrade the application's performance. The same is true if the memory resource for the Pub/Sub broker is concerned. In either case, the limitation of memory seriously impacts the performance of the component.
- CPU: If CPU time on a machine is limited, or budget constraints force the purchase of a smaller machine than is really required, then the upper limit on processor operations has an obvious and detrimental impact on any processes running on that machine.
- Network: Constraints on network resources are one of the most damaging to the performance of a Pub/Sub infrastructure. A slow network, or one which does not support advanced networking technologies like multicast, limits the amount of data which can flow between applications and components no matter how well designed and fast they are. As an aside, limits on numbers of connections may well impact against availability requirements too.

All of the above situations fall under the general area of performance bottlenecks. As for the balancing of reliability and performance, the analysis of performance bottlenecks requires prototyping of the system. This either forces the changing of the resource constraints (for instance, by feeding into a budget approval process) or provides hard figures about the final performance of the production system based on the existing requirements.

4.2.4 Security versus performance

Security requirements can have similar impact on performance to those caused by reliability requirements. Secure systems have checks and processes that must be completed in order to validate incoming connections, or to secure existing ones. These take time and, therefore, impact performance. User authentication at connection time, encryption of communication, fire-walling, and other security measures are the least likely to be negotiable, and therefore the overheads introduced must be accounted for in the performance planning.

4.2.5 Availability versus reliability

Consider availability and reliability as conflicting sets of requirements. The definition of a reliable system is one that maintains state in order to recover to a consistent point in the event of a failure. This definition implies that there is some time spent when the system is starting, in both checking for and recovering in-flight transactions that were interrupted by termination. The availability requirements concerning down-time may conflict with this, as they are likely to demand a maximum restart time for the system that does not account for recovery.

Unless the availability requirements are very lax, there is a likelihood of conflict in this area, which can have a knock-on effect on performance (because reliability and performance are so heavily dependent on each other). In the same way as the performance requirements may need to be changed to satisfy the reliability requirements, the availability requirements may need to accede to the reliability requirements as well. The converse is also true, as reliability could be sacrificed for speed of restart, if that is the highest priority requirement.

5

Configuration and tuning of WBI Event Broker

In the previous chapter, the importance of eventual design requirements were discussed and also the relationships that the sets of requirements can have with each other. It is very easy to get involved with details during the requirements definition and negotiation process, and keeping one eye on the wider picture of the system architecture is definitely recommended. In order to help with this, the next few chapters match the requirements up against functionality provided by WebSphere Business Integration Event Broker.

5.1 Satisfying performance requirements

There are numerous functions and technologies that can be harnessed to make a WebSphere Business Integration Event Broker high performance Pub/Sub architecture. These vary from utilizing efficient network protocols right up to the high-level configuration of the application and are listed below together with the benefits they bring.

5.1.1 WebSphere MQ real-time transport

The WebSphere MQ real-time transport is a very lightweight messaging protocol that is designed to be used for non-persistent messages. It is highly scalable, offers very high message throughput, and is only available to applications written in WebSphere MQ JMS. The reason for its high performance (and also the reason why it does not support persistent messages or durable subscriptions) is the fact that the applications and the broker do not communicate over queues. Instead, the communication is accomplished using direct connections to TCP/IP sockets. As a result, this transport is sometimes referred to as *Direct IP*.

This transport is best suited for applications where the requirements are for very high performance but *not* for high reliability, such as a stock ticker application that has many client connections but where the message data has a limited useful life and therefore does not need persisting.

5.1.2 WebSphere MQ multicast transport

The WebSphere MQ multicast transport is similar to the WebSphere MQ real-time transport above, in the sense that it uses direct IP connections between applications and brokers. This brings the same performance improvements detailed above; however, the difference between the two is that this transport uses a multicasting model and therefore brings even greater performance improvements, especially in a Pub/Sub environment. The Reliable Multicast Messaging (RMM) framework that provides multicast functionality in WebSphere Business Integration Event Broker also implements an acknowledgement system that supports reliable message delivery.

5.1.3 Non-persistent Messages on WebSphere MQ queues

If WebSphere MQ queuing is required to be the transport for Pub/Sub applications, then higher performance can be achieved by using non-persistent messages. These out-perform persistent messages (see the WebSphere MQ performance reports for detailed comparisons of throughput), but there is no advantage in using non-persistent messages on WebSphere MQ queues if the real-time or multicast transports can be used instead. See “Tuning for performance” on page 60 for more guidance on performance tuning.

Note: WebSphere MQ 5.3 CSD 6 offers a new setting onto local queues called *NPMCLASS*. This can be set to HIGH or NORMAL (where the latter is the default, and matches previous WebSphere MQ behavior). Setting this new parameter to HIGH will force the queue manager to store non-persistent messages to disk so that they will survive a queue manager restart.

5.1.4 Broker collectives

The parallelism inherent in WebSphere Business Integration Event Broker collectives can improve the general performance of a Pub/Sub system. This is not due in any direct way to the collective itself, but more due to the fact that the load-balancing aspect helps to eliminate any performance bottlenecks. The efficient way in which the brokers in the collective share information (both subscriptions and publications) together with the availability of several brokers for applications to connect to, leads to a more balanced use of the system's resources, especially if spread over several machines.

Collectives can also be managed in an intelligent fashion to further help in the area of performance. For instance, two geographically separate sites can be connected for Pub/Sub by having a broker instance in each site, and joining them in a collective. There is then efficient use of the network between the sites, as only two applications are communicating (instead of all the Pub/Sub applications from one site connecting to a broker at the other).

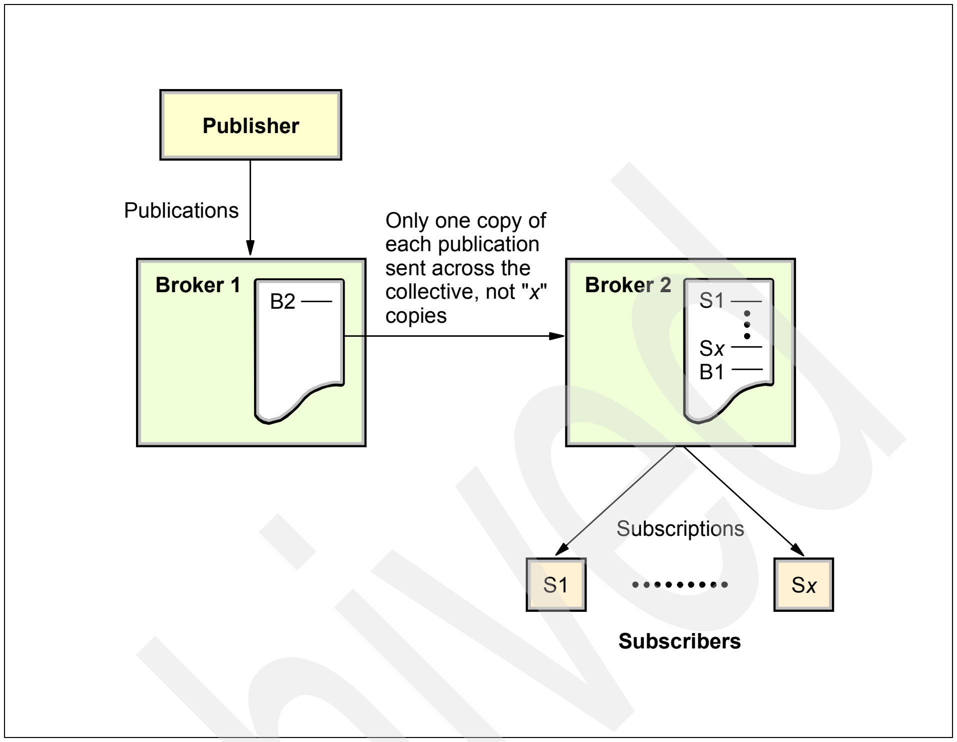

Publisher
Publications
Only one copy of each publication sent across the collective, not "x" copies
Broker 1
B2
Broker 2
S1
Sx
B1
Subscriptions
S1
Sx
Subscribers

Figure 5-1 Efficiency of a broker collective

5.1.5 JMS message selection

The JMS specifies the facility of message selection, which limits the messages that a subscriber receives to those whose JMS headers match a logical expression, supplied by the subscriber when it registers its subscription. The selector is a logical construct based on the SQL92 syntax, and provides a template for matching incoming messages against.

The selectors themselves can have a performance impact (a complex selector string matched against every incoming message has significant overhead), but WebSphere MQ JMS and WebSphere Business Integration Event Broker provide a means of managing this in two ways:

- The TopicConnectionFactory can be set to select the messages at the broker side. This cuts down network traffic (as unselected messages are not sent) but increases the processing loads on the broker.
- The TopicConnectionFactory can be set to select the messages at the client side. This does not decrease network load, and forces more processing to be

done in the client application, but the broker does not have the extra load from the message selection.

5.1.6 Broker statistics

Statistics gathering can be an important tool in monitoring and managing Pub/Sub networks. WebSphere Business Integration Event Broker provides support for this with its statistics reporting functionality. These are disabled by default, and are enabled from the command line with the **`mqsichangeproperties`** command. Gathering performance numbers for an execution group has a small performance impact, although this can be mitigated by choosing a wider reporting interval when statistics reporting is enabled.

Once a broker has had statistics reporting enabled, it publishes a statistics report at defined regular intervals. The reports are published on topics based on the following naming convention:

```
$SYS/Broker/<broker>/ExecutionGroup/<execution group>/Statistics
```

As for all Pub/Sub, wildcards are permitted when subscribing to the topics and the publications are in the form of JMS messages in XML format. Multicast statistics reports are published in the same way, but on a different set of topics:

```
$SYS/Broker/<broker>/ExecutionGroup/<execution group>/Statistics/Multicast/Topics
$SYS/Broker/<broker>/ExecutionGroup/<execution group>/Statistics/Multicast/Groups
$SYS/Broker/<broker>/ExecutionGroup/<execution group>/Statistics/Multicast/Summary
```

5.1.7 Message flow design

The design of WebSphere Business Integration Event Broker generally provides several ways of tuning message flows for performance. Multiple brokers, multiple execution groups within a broker, and multiple flow instances within an execution group provide several levels of tuning for parallelism of message processing. This parallelism can break message ordering requirements, and configuration can be managed at deployment to mitigate for this, but in effect it turns a multi-threaded system back into a single-threaded one.

Experimenting with numbers of message flow instances, numbers of execution groups, and numbers of brokers can provide important feedback into resource requirements, as well as help to satisfy performance and scalability requirements.

5.2 Satisfying reliability requirements

If the applications using WebSphere Business Integration Event Broker are sending messages containing business-critical data, then the driving requirement on the architecture is likely to be reliability. WebSphere Business Integration Event Broker offers several different ways of making a system reliable, so that data inconsistencies do not occur and recovery from system failure is possible. These reliability functions are listed below, together with the reliability benefits they bring. It is important to note that using these facilities increases reliability but has a corresponding impact on performance.

5.2.1 Persistent messages on WebSphere MQ queues

If reliable message delivery is required for applications using WebSphere Business Integration Event Broker, then using the WebSphere MQ queue-based transport and persistent messages is a good solution. Based on one of the cornerstones of the WebSphere MQ product design, message persistence involves the writing of message data to disk, and when combined with the underlying queue manager's check-pointing process, provides complete recovery of data in the case of an unexpected program termination.

5.2.2 Transactional control

A transactional system is one that uses a dedicated protocol between applications in order to ensure that data exchanged between them is in a consistent state. A transaction (sometimes referred to as a *unit of work* or *UOW*) is the interval covering a sequence of data exchanges between the applications which must be kept atomic; in other words, either all the updates must happen, or none. The action of completing a transaction is either a commit (the transaction was successful) or a rollback (there was an error, none of the changes within the transaction were made, and everything is restored to the state it was in before the transaction started).

WebSphere MQ provides for two types of transactional control. These are one-phase commit and two-phase commit:

- One-phase commit is when there is a controlling WebSphere MQ application involved in communication with one WebSphere MQ queue manager. There is only one round of communication between the application and the queue manager when the application commits or rolls back the transaction.
- Two-phase commit is when there is a controlling application involved in communication with more than one WebSphere MQ queue manager (or other transacted resources, such as databases). The two-phase commit protocol is more complex and requires an external transaction coordinator such as

WebSphere Application Server to control the two rounds of communication between the application and the resources it has updated. For data integrity to be maintained, two-phase commit transactions are required if more than one data resource is to be updated. An accepted industry standard for the two-phase commit protocol is the X/Open Group's *Distributed Transaction Processing: The XA Specification*, found here:

`http://www.opengroup.org/public/pubs/catalog/c193.htm`

Depending on the implementation of the Pub/Sub system and its interaction with other data resources, either one or two-phase commit transactions must be used to ensure reliability (in the sense that the system's data remains consistent).

Note: One- or two-phase commit transactions are only supported for applications using WebSphere MQ queues as their transport. No transactional control is available with the WebSphere MQ real-time or multicast transports, although Reliable Multicast Messaging (RMM) does offer enhanced protection against data loss. Also, WebSphere Business Integration Event Broker cannot act as an external transaction coordinator. If a message flow is to be part of a distributed transaction, a separate external transaction coordinator must be used.

5.2.3 JMS durable subscriptions

Durable subscriptions are specified in the JMS, and provide for reliability of message delivery for clients. They are only supported if WebSphere MQ is the message transport (not real-time or multicast) and means that a client can disconnect, re-connect later, and have all the messages that were published in the meantime delivered to it on re-connection. The subscription has a unique identifier that must be used when re-connecting in order to match up with the previous subscription.

5.2.4 Stream-crossing within the broker

When considering the reliability requirements of different parts of the Pub/Sub architecture, differences may be found. For instance, consider a system where the publications *must* be received and published by the broker, but it is not mandatory for all subscribers to receive every message. In this case, if the highest reliability requirement dictated the requirement for the whole system, then everything would be constrained by WebSphere MQ persistent messaging (as required by the publishing side).

However, WebSphere Business Integration Event Broker offers a solution to this kind of requirements mismatch: Stream-crossing within the broker. This facility allows the connection of different types of messaging transport within the broker, so that messages incoming from the publishers in one protocol can be mapped to one or more different protocols for the subscribers. In the example above, this would mean that the incoming WebSphere MQ-queued persistent messages could be published out to the subscribers using the WebSphere MQ real-time transport, therefore not constraining the information flow to the subscribers with the lower performance of the incoming protocol.

5.3 Satisfying availability requirements

Availability requirements on a system, as previously stated, are those that specify when a system's infrastructure must be available for use by its dependent applications. This can vary from a certain number of hours per day, up to full 24-by-7 availability. WebSphere Business Integration Event Broker supports two technologies that help satisfy availability requirements, and these are listed below together with the benefits they bring.

5.3.1 High availability

WebSphere Business Integration Event Broker and its underlying queue manager can be configured to be made highly available by exploiting operating system support for failure detection and failover to a standby machine. Instructions and support for this kind of configuration can be found here:

`http://www.ibm.com/software/integration/support/supportpacs/`

To make a WebSphere Business Integration Event Broker instance highly available, the broker, its database, and its queue manager all need to be configured to be monitored and failed over by the High Availability (HA) application monitors. This usually involves setting up a shared disk between the two HA nodes, and setting up the file systems required by the broker on this shared disk. The HA monitors the need to be configured to watch the appropriate processes, or in some other manner to detect that the resources are working correctly.

If a failure occurs, then remedial action is taken by the HA monitors, which may involve failing the whole system over to the backup machine. Because the nodes share the disk, the only overheads for failing over are those related to process initiation and resolution of any in-doubt transactions, and this therefore helps ensure only a short downtime should the node fail.

HA nodes can be configured in groups of three or more, with one node being a standby node for any of the active nodes. This kind of architecture reduces the possibility of a total system outage due to software or machine failure by an even greater degree, especially if the nodes are geographically separated in order to counter the effects of a power failure or other disaster scenario.

5.3.2 Cloned brokers

WebSphere Business Integration Event Broker instances can be cloned, which involves one broker copying all information about its registered subscribers to its clone (or clones). This means that an incoming publication to one broker is automatically sent to all subscribers (of *all* the cloned brokers) by the broker that received the publication. In a sense, a subscriber to one broker can be thought of as though it had made subscriptions to all the cloned brokers.

From the messaging point of view, cloned brokers differ from collectives. Collectives share publications at the broker level, whereas cloned brokers each send any publications they receive on to all subscribers rather than to the other brokers. Another important point is that brokers must not be cloned inside a collective. This results in duplicated publications being received by subscribers, as publications received by one broker are sent to both other brokers (by the collective) and directly to all subscribers (by the cloning).

Cloned brokers rely on WebSphere MQ to be the transport between the cloned brokers and their subscribers. This means that subscribers to cloned brokers cannot be used with either the WebSphere MQ real-time or multicast transports. However, the benefit that they bring to a group of brokers in availability terms (if one broker goes down then its subscribers still receive publications from its clones) can make this a worthwhile trade-off.

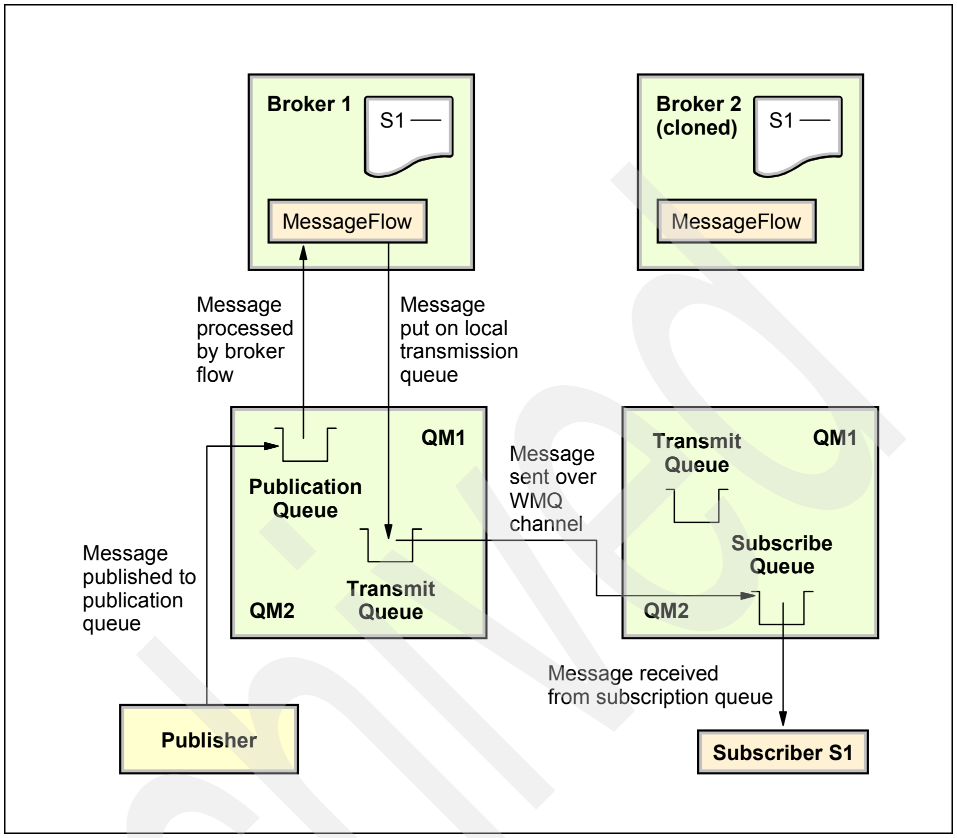

Broker 1
S1
MessageFlow
Broker 2 (cloned)
S1
MessageFlow
Message processed by broker flow
Message put on local transmission queue
QM1
Publication Queue
QM2
Transmit Queue
Message sent over WMQ channel
Transmit Queue
QM1
Subscribe Queue
QM2
Message published to publication queue
Message received from subscription queue
Publisher
Subscriber S1

Figure 5-2 Message flow through cloned brokers and their queue managers

5.4 Satisfying scalability requirements

WebSphere Business Integration Event Broker itself scales well to suit changing loads on the Pub/Sub system, providing that its applications are either generic or have been designed with system scalability in mind. There are two major points to cover regarding scalability, and these are listed below, together with the benefits they bring.

5.4.1 Broker collectives

Collectives provide the most obvious benefit to a scalable system. When contained in a collective, brokers share subscription information and publications in the most efficient manner. Therefore, adding extra brokers into a collective significantly increases the numbers of publishers and subscribers that can be supported without a corresponding rise in inter-broker traffic in the collective. Collectives can be used with either WebSphere MQ queue-based clients, WebSphere MQ real-time transports, or both at the same time.

5.4.2 Topic hierarchies

The topic hierarchy structure in a Pub/Sub environment is a totally arbitrary creation. Topics and sub-topics can be named in any way that does not contravene the naming convention, and the topic tree (as it cascades from the root topic) can be of any depth and complexity. However, examination of the data to be passed as publication messages suggests sensible partitioning of the data into the topic hierarchy. The information requirements of the subscribing applications also have a bearing on this partitioning.

It is important to remember that when planning the topic hierarchy structure, the data running through the system may change over time, and therefore the plan must allow for this expansion.

5.5 Satisfying security requirements

WebSphere Business Integration Event Broker offers significant security features at several levels that can help to secure Pub/Sub architectures. These are listed below.

5.5.1 ACLs and topic-based security

WebSphere Business Integration Event Broker allows security to be configured at the level of a single topic with an Access Control List (ACL) determining which users can publish and subscribe to the topic. This security is managed by the User Name Server component. If no specific ACL is specified for a particular topic, the security for it is inherited from its parent (or ancestor) topic. Because subscription registrations can contain wildcards, the decision to deliver to a subscriber is made only when a publication on a particular topic is being processed by the broker.

5.5.2 Authentication for real-time connections

WebSphere Business Integration Event Broker directly supports authentication for connections using the WebSphere MQ real-time transport. These authentication services require a User Name Server, and verify that a broker and a client application are who they claim they are, and can therefore participate in a Pub/Sub session. WebSphere Business Integration Event Broker supports the following four security protocols for WebSphere MQ JMS applications using the WebSphere MQ real-time transport, in strength order (weakest first):

- Simple telnet-like password authentication: This can be thought of as *password in the clear* because the password is sent unencrypted over the network when the client connects. The broker then verifies the user and password against the information held in the User Name Server.
- Mutual challenge-response password authentication: More sophisticated than the previous protocol, and that involves the generation of a secret session key. Both the broker and the client generate the key based on the password of the client, and prove to each other that they know the generated key by challenge and response. Again, the broker uses information held in the User Name Server in order to validate the client.
- Asymmetric SSL: This is the standard SSL authentication used in most Web browsers; only the brokers have public/private key pairs, and the clients know the public keys of the brokers. A secure connection is established, in which the broker is authenticated to the client using public key cryptography, before the client sends its encrypted password for authentication with the broker.
- Symmetric SSL: In this protocol, both the broker and the client have public/private key pairs, and public key cryptography is used for mutual authentication.

Secure Sockets Layer (SSL) is not maintained for the lifetime of the connection in either case. It is used to establish the connection, but is not used after this point because of the unacceptable overheads involved in encrypting all traffic over the connection.

If the broker and its clients reside on different intranets, there can be connection problems at the firewalls. However, provided that a broker's firewall has been configured to allow incoming connections from clients, HTTP tunneling can be used to allow the clients and broker to communicate. HTTP tunneling is not supported in conjunction with SSL; either one or the other must be used. There are two options for a client to connect to a broker through its own firewall:

- HTTP tunneling: This is suitable for situations where an attempt to connect explicitly to an HTTP proxy server would be rejected. Activation of HTTP tunneling support in WebSphere Business Integration Event Broker is supported on a per-machine basis.
- Connect via proxy: This is suitable for situations where a direct connection to an HTTP proxy is not rejected.

5.5.3 Authentication for connections using WebSphere MQ queues

WebSphere Business Integration Event Broker does not directly offer security support for clients that use WebSphere MQ queues to communicate with the broker. However, WebSphere MQ itself has several security options that can be used:

- WebSphere MQ directly supports SSL over its channels, both for server-server and client-server connections. This is fully documented in the *WebSphere MQ Security, SC34-6079*, publication.
- WebSphere MQ offers exit hooks for the development of custom security and message encryption exits. These can either be written directly by the reader or can be purchased from various third-party suppliers and IBM business partners.

5.5.4 Quality of protection for messages

Message protection and the various levels thereof are provided to complement the real-time connection authentication by preventing tampering with messages flowing over existing connections. Message protection must be enabled at the broker in question, but once this has been done, the granularity of the protection can be set at the level of a single topic or topic sub-tree. The settings on a topic can be one of the following (excluding *none*, which is the default):

- Channel Integrity: This prevents the insertion or deletion of messages from a connection without detection.
- Message Integrity: This prevents the contents of messages being altered without detection.
- Encrypted for Privacy: This prevent the contents of messages being examined, thus securing the connection completely.

The settings are cumulative, with the top level (Encrypted for Privacy) including both the lower two types of security. Message protection is not supported with simple telnet-like password authentication.

5.6 Developing a broker architecture

The previous sections have shown how WebSphere Business Integration Event Broker can satisfy performance, reliability, availability, scalability and security requirements for a Pub/Sub system. The previous chapter also described how these requirements can conflict with each other. This final section brings together this information, describing how to configure the broker based on the highest priority requirement.

5.6.1 Tuning for performance

If performance is the overriding requirement on WebSphere Business Integration Event Broker, then the solution is to use either the WebSphere MQ real-time or multicast transport. The multicast transport in particular also offers a high degree of reliability (if required) for such a protocol, so it is not a complete trade-off between high performance and high reliability.

Only non-persistent WebSphere MQ messages are supported over the real-time and multicast protocols, but they are also supported on WebSphere MQ queues. However, using non-persistent messages on WebSphere MQ queues is not the best use of the technology. If non-persistent messages fit into the reliability requirements of the system, why not use a protocol that maximizes the throughput? The Pub/Sub system may contain legacy applications that use WebSphere MQ queues for communication, but the stream-crossing and multiple protocol configuration possibilities of WebSphere Business Integration Event Broker minimize this impact.

5.6.2 Tuning for reliability

If WebSphere Business Integration Event Broker is required to be a reliable solution, then the configuration is actually very clear. Messages must be persistent (WebSphere MQ queues must be used as the transport), transactions must be enabled, and the JMS subscribers must be durable. There is no point in having only some of these, as the reliability benefits provided by the ones that are used negates the missing ones:

- Transactions and durable subscriptions are pointless without message persistence, as an unexpected termination can lose data because the messages are held in volatile memory and not persisted to disk.
- Not having transactional control over persistent messages and durable subscriptions does not make sense either. Why bother persisting data to disk or holding messages for delivery at the broker, if data inconsistency is tolerated by the applications?
- If durable subscriptions are not required, why would message persistence and transactional control also be required? If applications can tolerate missing messages due to their unexpected termination, why can the broker also not suffer message loss?

Thus, despite the large matrix of possibilities provided by WebSphere Business Integration Event Broker for reliability, it is an all or nothing scenario; a broker must either be highly reliable or highly performing, although it must be noted that even a highly reliable WebSphere Business Integration Event Broker compares well in performance terms to comparably reliable Pub/Sub systems.

5.6.3 Tuning for high availability

A highly available WebSphere Business Integration Event Broker solution probably either uses hardware support (such as HACMP™ on AIX®) or cloned brokers for availability. An HA solution allows either a highly performing or a highly reliable system (as discussed in the previous two sections), but a cloned broker system relies on WebSphere MQ queues as its transport, and therefore is more likely to be highly reliable.

If budget constraints have dictated that cloned brokers are the only possible solution for high availability, then this may be another reason to consider using non-persistent messages over WebSphere MQ queues in order to maximize performance.

5.6.4 Multiple instances and multicast

Multicast messaging is managed on a per-broker basis by the allocation of a multicast group IP address to a particular topic. This means that careful management of an architecture containing multiple brokers is required, if multicast is being used. The consequences of two brokers selecting the same multicast group IP address for different topics could be severe. Therefore, if multicast is to be used with multiple WebSphere Business Integration Event Broker instances, then each broker must be allocated a non-intersecting partition of the multicast address range. This removes the possibility of conflicts caused by a broker using the same multicast group address.

Published messages shared across a collective cannot be received by multicast subscribers. WebSphere Business Integration Event Broker multicast subscribers can only receive publications that have been published to the same broker as they are subscribed to (although the same is not true of the WebSphere MQ queue-based or real-time transports). Brokers in the same collective must still be configured not to interfere with the multicast group allocation of each other, even though publications from across the collective are not delivered to multicast subscribers.

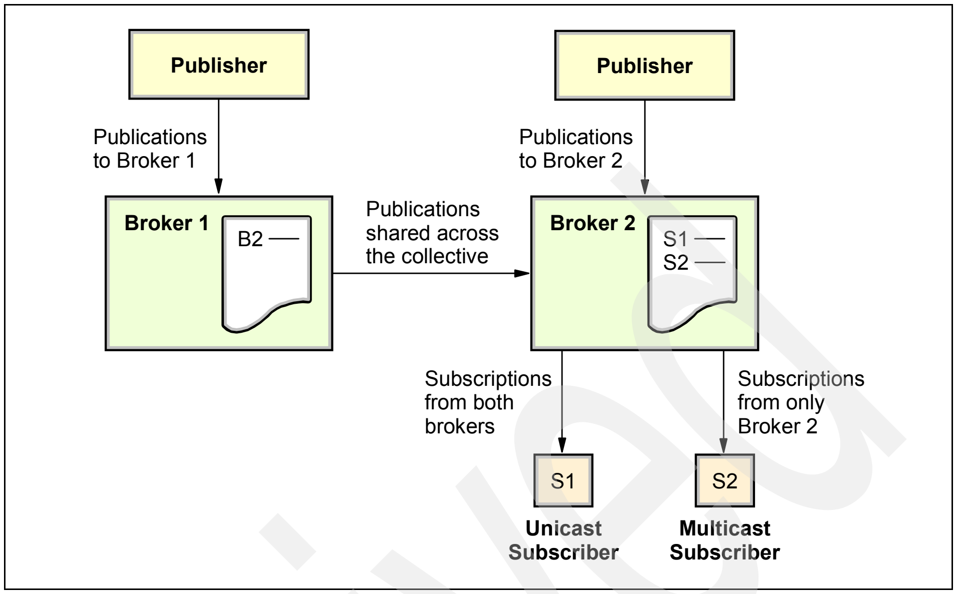


Figure 5-3 Unicast and multicast in a broker collective

Part 2

Rationale of WBI Event Broker

This publication is divided into two parts, a theoretical part and a practical part. This is the second part, and uses the theory in the first part in a series of worked examples showing the various WebSphere Business Integration Event Broker facilities. It is divided into the following chapters:

- Chapter 6, “Installation” on page 67, illustrates step-by-step installation of WebSphere Business Integration Event Broker and the pre-required supporting products.
- Chapter 7, “Configuration and administration of WBI Event Broker” on page 103, covers the setting up of components, and the basic generation and deployment of a message flow.

- Chapter 8, “Overview of basic scenario” on page 159, reviews the basic applications used in the scenarios, and describes how they interact with a broker (or brokers) in the following samples. The applications are used without alteration in order to demonstrate the separation between Pub/Sub *application* implementation and Pub/Sub *architecture* implementation. The Java code used in these business case scenarios is provided in Appendix A, “Code used in the business case scenario” on page 223.
- Chapter 9, “Real-time transport for single broker performance” on page 165, shows a single broker using the WebSphere MQ real-time transport to provide high throughput for the Pub/Sub messages. The Java code used in these business case scenarios is provided in Appendix A, “Code used in the business case scenario” on page 223.
- Chapter 10, “Multicast transport for single broker performance” on page 173, shows a single broker configured to use multicast messaging (WebSphere MQ RMM) for the Pub/Sub messages. The Java code used in these business case scenarios is provided in Appendix A, “Code used in the business case scenario” on page 223.
- Chapter 11, “Real-time transport for broker collective performance” on page 179, shows a broker collective using the WebSphere MQ real-time transport to provide high throughput for the Pub/Sub messages. The Java code used in these business case scenarios is provided in Appendix A, “Code used in the business case scenario” on page 223.
- Chapter 12, “Persistent messages for single broker reliability” on page 187, shows a single broker using WebSphere MQ to provide persistence for the Pub/Sub messages, together with durable subscriptions and transactional control. The Java code used in these business case scenarios is provided in Appendix A, “Code used in the business case scenario” on page 223.
- Chapter 13, “Stream-crossing for single broker flexibility” on page 199, shows a single broker configured to use both persistent and non-persistent messages in the Pub/Sub architecture, thus providing a higher degree of granularity when matching requirements to infrastructure design. The Java code used in these business case scenarios is provided in Appendix A, “Code used in the business case scenario” on page 223.
- Chapter 14, “Cloned brokers for high availability” on page 207, shows a cloned broker system designed to mitigate broker failures and thus provide a greater degree of system reliability. The Java code used in these business case scenarios is provided in Appendix A, “Code used in the business case scenario” on page 223.
- Chapter 15, “Using multicast in a broker collective” on page 215, shows a broker collective configured to use multicast messaging (WebSphere MQ RMM) for the Pub/Sub messages. The Java code used in these business

case scenarios is provided in Appendix A, “Code used in the business case scenario” on page 223.

6

Installation

This chapter provides simple instructions on how to install and configure an initial WebSphere Business Integration Message Broker environment on Windows 2000. This includes the following specific products:

- IBM DB2(R)Universal Database Enterprise Server Edition 7.2 Fixpack 9 or IBM DB2 Universal Database Enterprise Server Edition 8.1 Fixpack 3.
- IBM WebSphere MQ version 5.3.0.5 (V5.3 with CSD05).
- Microsoft® Data Access Components (MDAC) Version 2.7 SP1®, Version 2.7 SP1a, or version 2.8.
- IBM Agent Controller version 5.0.1. This is required if the plan is to use the flow debugger at any time in the future. The Agent Controller can be installed from the WebSphere Business Integration Message Broker V5.0 supplement CDs.
- IBM WebSphere Business Integration Event Broker Version 5 CSD03.
- WebScale Distribution Hub Version 2.0.0.1.

6.1 Installation requirements

Before installing the main products, ensure that their prerequisites are already installed:

- Windows 2000 Service Pack 3
- Java Runtime Environment 1.3.1

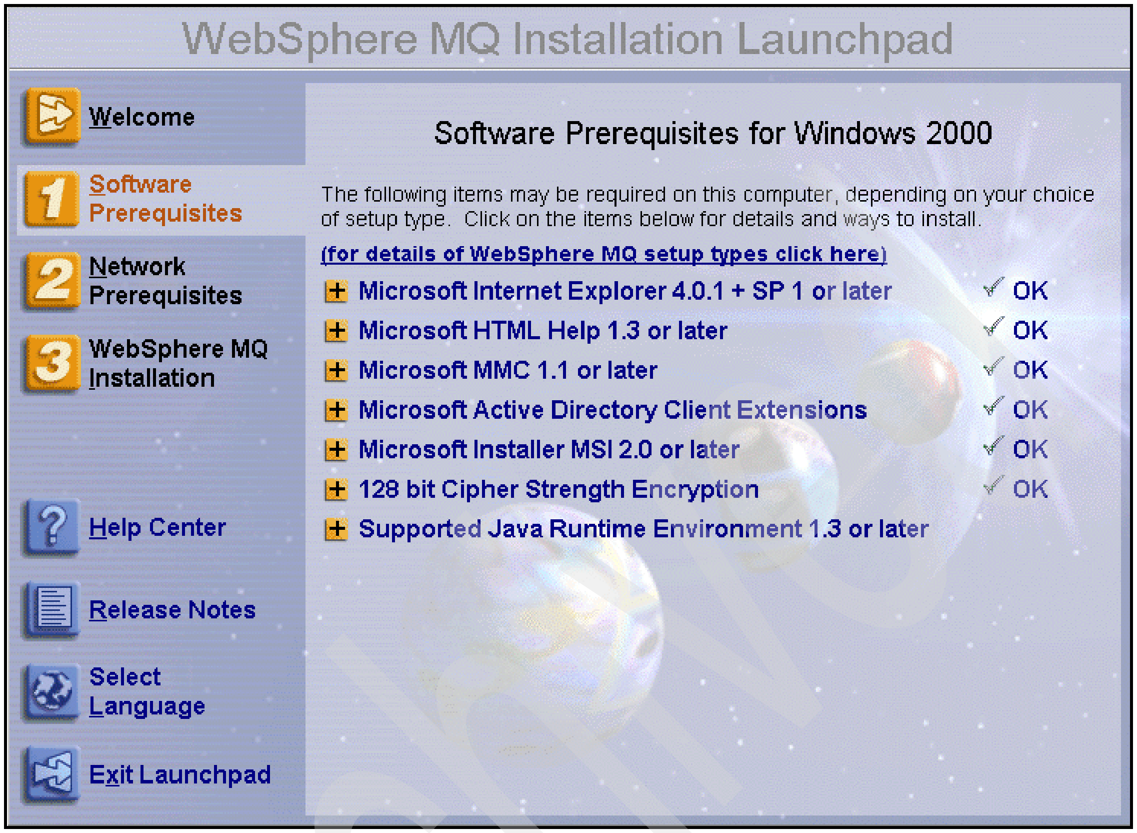
6.2 Installing WebSphere MQ 5.3

This section describes the installation of WebSphere MQ 5.3.

6.2.1 Launching the installation

To launch the installation:

1. The first screen displayed is the Launchpad shown in Figure 6-1 on page 69. This allows a verification that the product prerequisites are installed. Once all of the required software is verified, click **2 Network Prerequisites** to continue.


WebSphere MQ Installation Launchpad

Welcome

Software Prerequisites

Network Prerequisites

WebSphere MQ Installation

Help Center

Release Notes

Select Language

Exit Launchpad

Software Prerequisites for Windows 2000

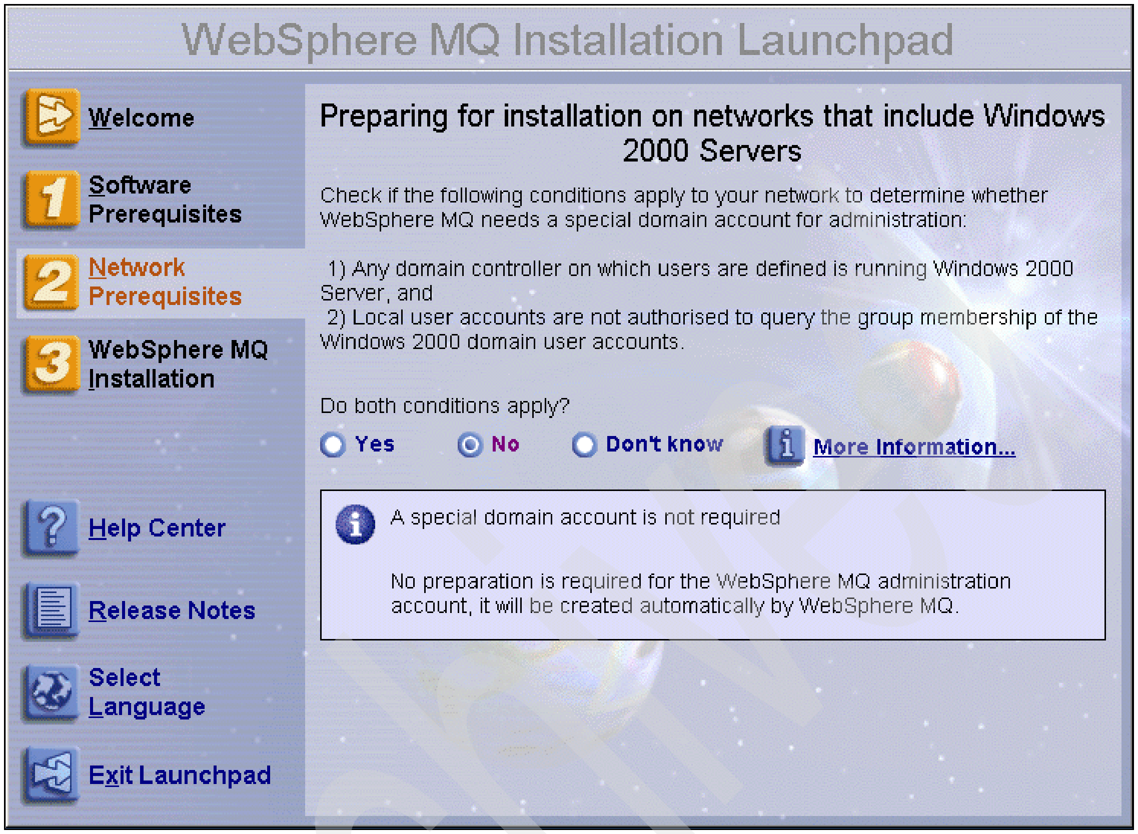
The following items may be required on this computer, depending on your choice of setup type. Click on the items below for details and ways to install.

(for details of WebSphere MQ setup types click here)

+ Microsoft Internet Explorer 4.0.1 + SP 1 or later	✓ OK
+ Microsoft HTML Help 1.3 or later	✓ OK
+ Microsoft MMC 1.1 or later	✓ OK
+ Microsoft Active Directory Client Extensions	✓ OK
+ Microsoft Installer MSI 2.0 or later	✓ OK
+ 128 bit Cipher Strength Encryption	✓ OK
+ Supported Java Runtime Environment 1.3 or later	

Figure 6-1 Checking the WebSphere MQ prerequisite software

2. There are no Windows 2000 domains in this environment and all of the security is local. Select **No** in the Network Prerequisites window (Figure 6-2 on page 70) and then click **3 WebSphere MQ Installation**.



Preparing for installation on networks that include Windows 2000 Servers

Check if the following conditions apply to your network to determine whether WebSphere MQ needs a special domain account for administration:

1) Any domain controller on which users are defined is running Windows 2000 Server, and
2) Local user accounts are not authorised to query the group membership of the Windows 2000 domain user accounts.

Do both conditions apply?

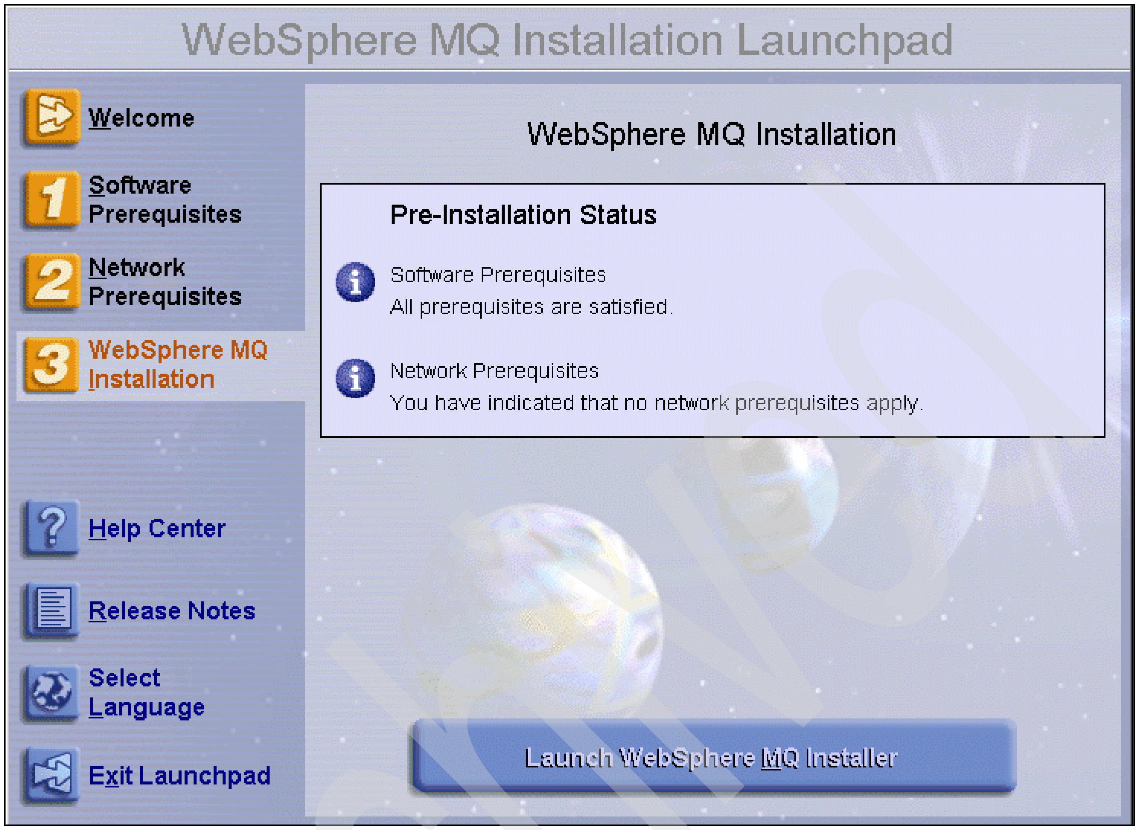
Yes [✓ No] Don't know More Information...

A special domain account is not required

No preparation is required for the WebSphere MQ administration account, it will be created automatically by WebSphere MQ.

Figure 6-2 Network prerequisites window

3. Start the install and click **Launch WebSphere MQ Installer** at the bottom of the next window. This is illustrated in Figure 6-3 on page 71.



WebSphere MQ Installation

Pre-Installation Status

Software Prerequisites
All prerequisites are satisfied.

Network Prerequisites
You have indicated that no network prerequisites apply.

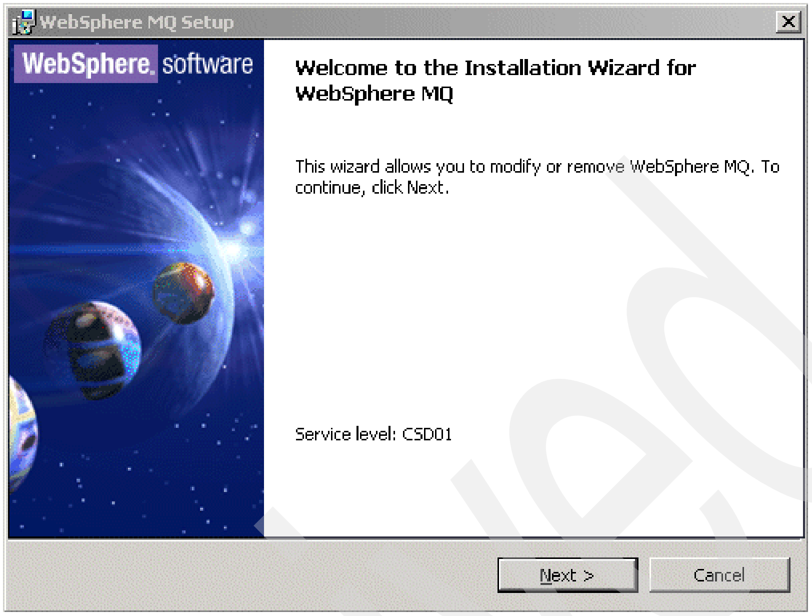
Figure 6-3 WebSphere MQ pre-installation status

6.2.2 Starting the installation

To start the installation:

1. The first choice is the language to use for the installation. The default selection is English. Click **OK**
2. An information screen is displayed that says that the installation is WebSphere MQ with CSD1 (see Figure 6-4 on page 72). Click **Next**.

Note: Installation of CSD05 is shown later in this chapter.



Welcome to the Installation Wizard for WebSphere MQ

This wizard allows you to modify or remove WebSphere MQ. To continue, click Next.

Service level: CSD01

Figure 6-4 Starting the WebSphere MQ installation

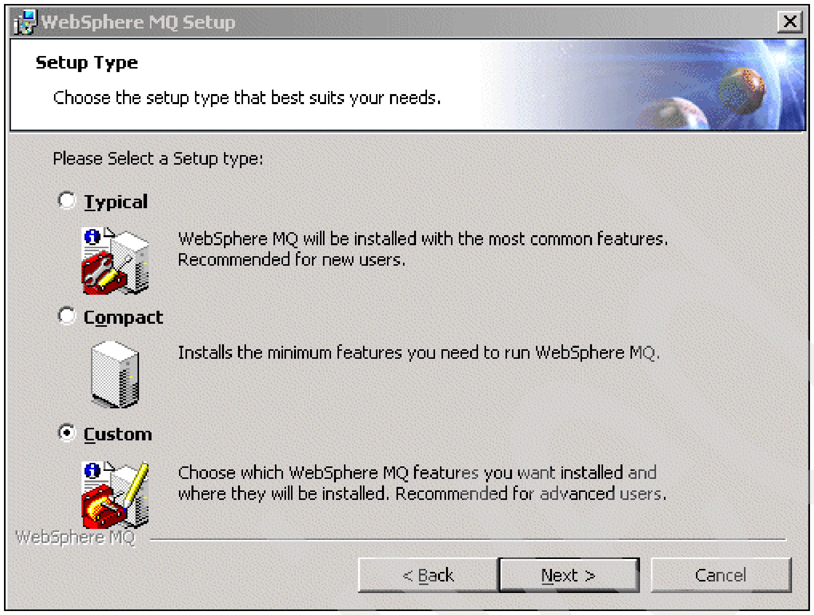
3. The next screen shows the product license agreement. To continue, select **I accept the terms in the license agreement** and then click **Next**.

6.2.3 Choosing the components to install

To choose the components to install:

1. The next screen gives an option to choose the Setup Type for the installation. Choose **Custom**, as shown in Figure 6-5 on page 73, and click **Next**.

Important: Do not choose Typical or Compact, otherwise the integrated Java support is not installed. In earlier releases, Java support was installed separately via the MA88 SupportPac™.


WebSphere MQ Setup

Setup Type

Choose the setup type that best suits your needs.

Please Select a Setup type:

( ) Typical

WebSphere MQ will be installed with the most common features. Recommended for new users.

( ) Compact

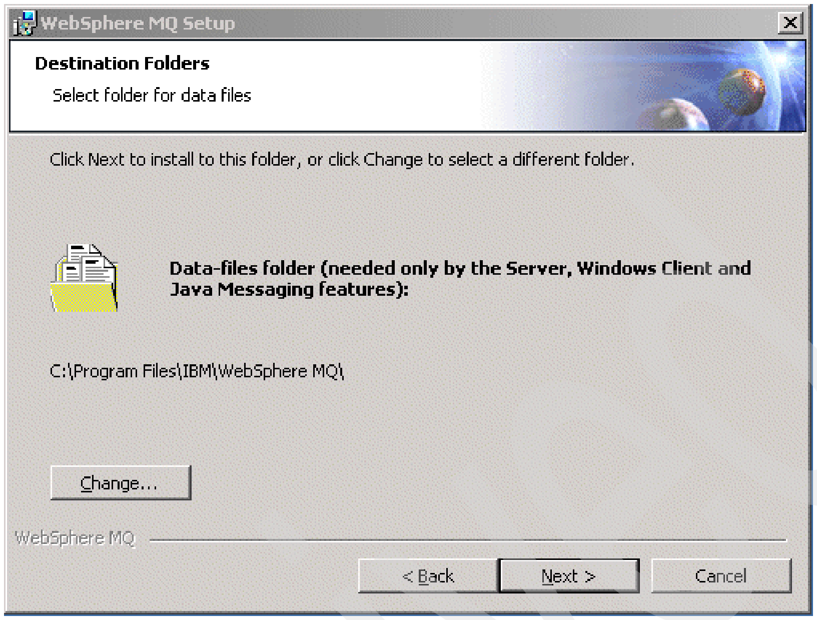
Installs the minimum features you need to run WebSphere MQ.

(●) Custom

Choose which WebSphere MQ features you want installed and where they will be installed. Recommended for advanced users.

Figure 6-5 Choosing the Custom setup type

2. The next three screens confirm the installation paths for the program, data and log files. By default these are installed to C:\Program Files \IBM\WebSphere MQ. If there is enough space on the C: drive then there is no need to change these locations. Figure 6-6 on page 74 illustrates one of these windows. Click **Next** through each of them.


WebSphere MQ Setup

Destination Folders

Select folder for data files

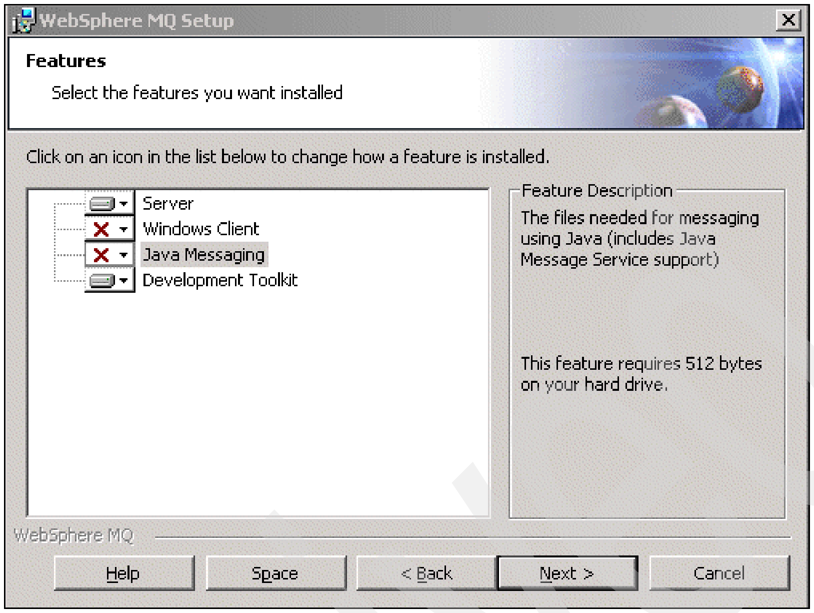
Click Next to install to this folder, or click Change to select a different folder.

Data-files folder (needed only by the Server, Windows Client and Java Messaging features):

C:\Program Files\IBM\WebSphere MQ\

Figure 6-6 Confirm the installation paths for the program, data and log files

3. The next screen gives an option to select the individual features to install. By default, the Windows Client and Java Messaging options are not installed. Figure 6-7 on page 75 shows this screen.


WebSphere MQ Setup

Features

Select the features you want installed

Click on an icon in the list below to change how a feature is installed.

- Server
- ✗ Windows Client
- ✗ Java Messaging
- Development Toolkit

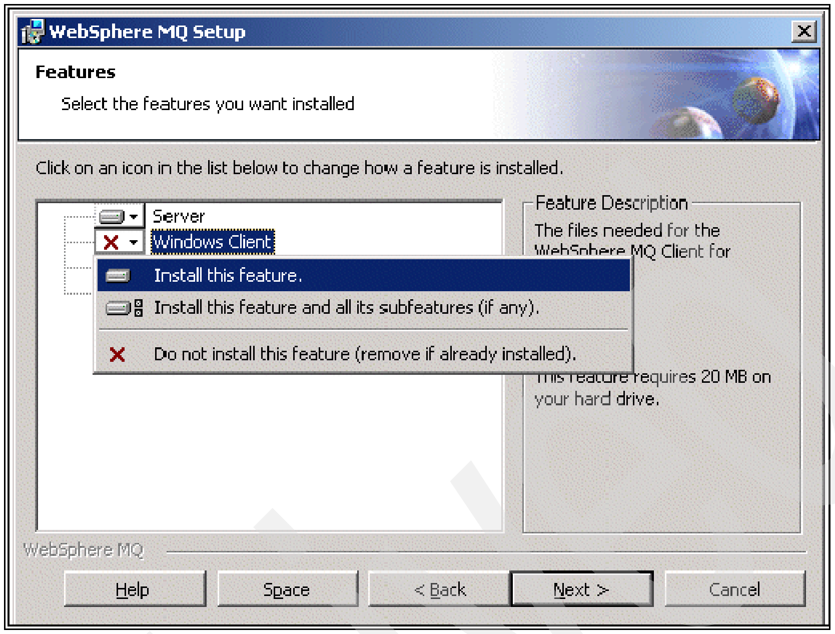
Feature Description

The files needed for messaging using Java (includes Java Message Service support)

This feature requires 512 bytes on your hard drive.

Figure 6-7 Choosing to install Windows Client and Java messaging

4. As these features need to be installed, select the drop-down next to Windows Client and choose **Install this feature** as shown in the example in Figure 6-8 on page 76.


WebSphere MQ Setup

Features

Select the features you want installed

Click on an icon in the list below to change how a feature is installed.

Server
Windows Client
Install this feature.
Install this feature and all its subfeatures (if any).
Do not install this feature (remove if already installed).
Feature Description
The files needed for the WebSphere MQ Client for
This feature requires 20 MB on your hard drive.
WebSphere MQ
Help
Space
< Back
Next >
Cancel

Figure 6-8 Choosing to install a Windows Client additional feature

5. Next select the drop-down next to Java Messaging and choose **Install this feature** as shown in the example in Figure 6-9 on page 77.

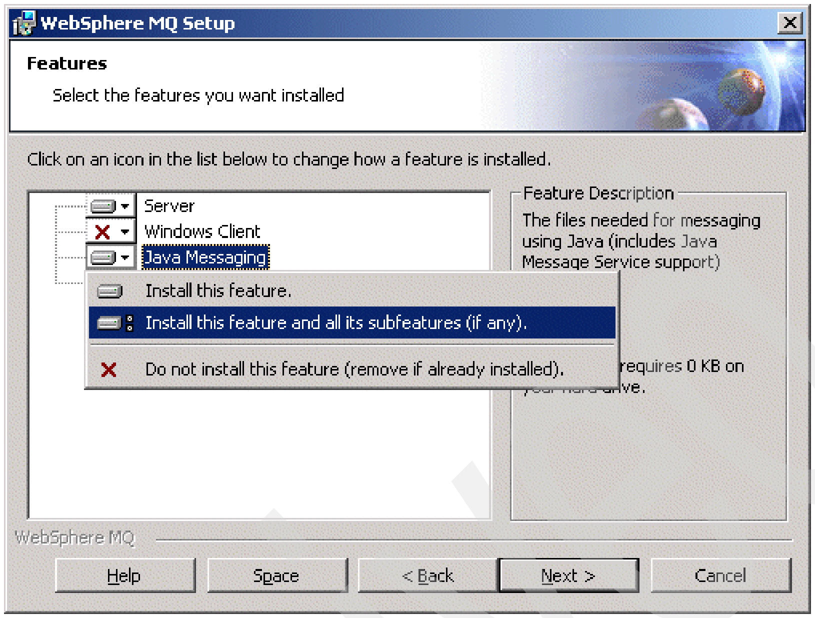

WebSphere MQ Setup
Features
Select the features you want installed
Click on an icon in the list below to change how a feature is installed.
Server
Windows Client
Java Messaging
Install this feature.
Install this feature and all its subfeatures (if any).
Do not install this feature (remove if already installed).
Feature Description
The files needed for messaging using Java (includes Java Message Service support)
requires 0 KB on your hard drive.
WebSphere MQ
Help
Space
< Back
Next >
Cancel

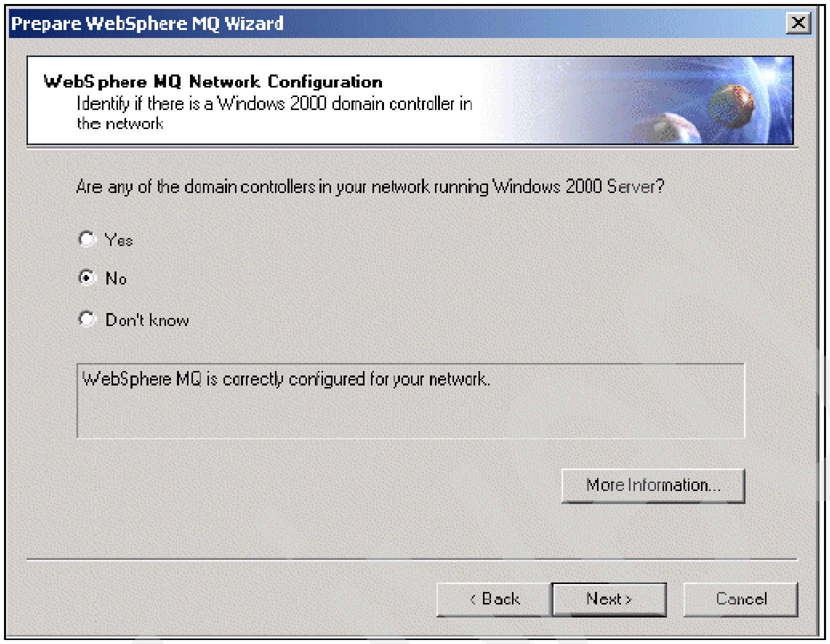
Figure 6-9 Choosing to install a Java messaging additional feature

6. The next screen displays that the installation is ready to start. Click **Install**.
7. A prompt appears asking whether there are enough license units. Click **Yes**.
8. During the installation, a status screen appears with a progress bar. When it is complete, a window is displayed that the installation was successful. Click **Finish**.

6.2.4 Configuring WebSphere MQ

At this point do not configure a default queue manager. Complete WebSphere MQ installation by skipping most of the configuration steps.

1. Once the installation has completed, the Prepare WebSphere MQ Wizard starts. Click **Next**.
2. A window about the network configuration is displayed. Select the option to indicate that *no* Windows 2000 domain controllers are on the network, as shown in Figure 6-10 on page 78. Then click **Next**.


Prepare WebSphere MQ Wizard
WebSphere MQ Network Configuration
Identify if there is a Windows 2000 domain controller in the network

Are any of the domain controllers in your network running Windows 2000 Server?

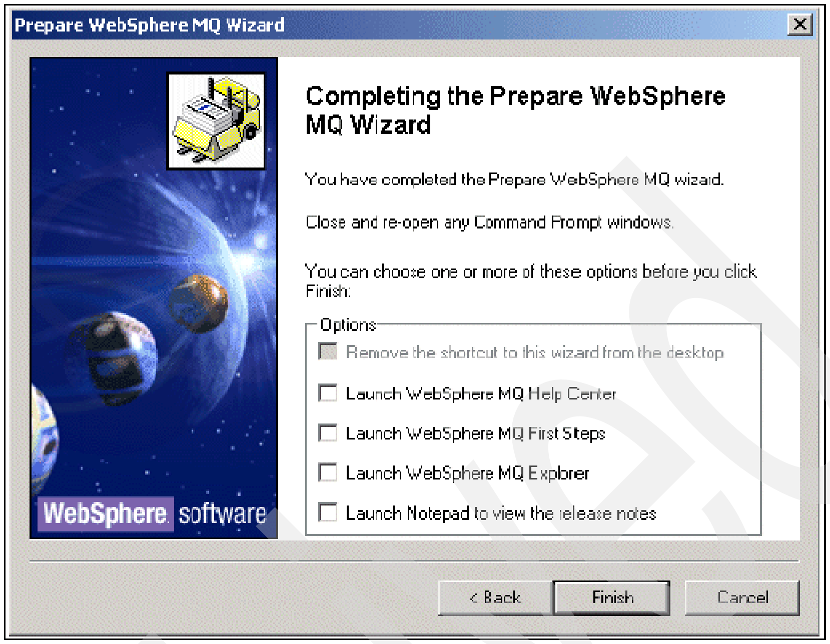
Yes [✓ No] Don't know

WebSphere MQ is correctly configured for your network.

More Information...
< Back
Next >
Cancel

Figure 6-10 Setting the network configuration options for WebSphere MQ

3. The next window recommends that a default configuration for WebSphere MQ is created. It is not necessary to do this at this stage. Click **Next**.
4. The final screen prompts verification of the release notes and launch various tools. Deselect all of the options as shown in Figure 6-11 on page 79 and click **Finish**.


Prepare WebSphere MQ Wizard

Completing the Prepare WebSphere MQ Wizard

You have completed the Prepare WebSphere MQ wizard.

Close and re-open any Command Prompt windows.

You can choose one or more of these options before you click Finish:

Options
- [ ] Remove the shortcut to this wizard from the desktop
- [ ] Launch WebSphere MQ Help Center
- [ ] Launch WebSphere MQ First Steps
- [ ] Launch WebSphere MQ Explorer
- [ ] Launch Notepad to view the release notes

Figure 6-11 Completing the Prepare WebSphere MQ Wizard

5. The basic installation is now complete.

Note: It is advisable not to install the WebSphere MQ documentation. The documentation is available as a separate install image and is not part of the basic installation.

6.2.5 Installing the CSD

After the Prepare WebSphere MQ Wizard completed in the previous section, several MQ processes were started. To install the CSD, stop these processes.

1. Right-click theWebSphere MQ icon in the System Tray in the Windows task bar, and select **Stop WebSphere MQ** from the menu, as shown in Figure 6-12 on page 80. A reminder is displayed at this stage of all WebSphere MQ processes terminating; click **Yes**.

   Note: At the time of writing this publication, the update available was CSD05. We recommend that the latest WebSphere MQ update is applied to ensure that all of the latest updates for the product are available.

The latest CSD for WebSphere MQ v5.3 is available at:

http://www.ibm.com/software/integration/mqfamily/support/summary

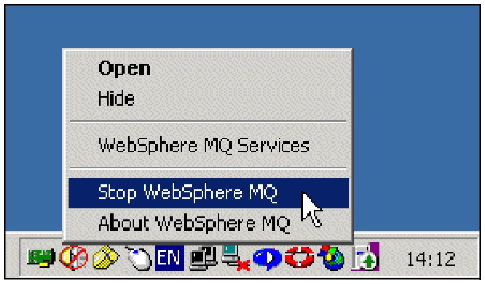
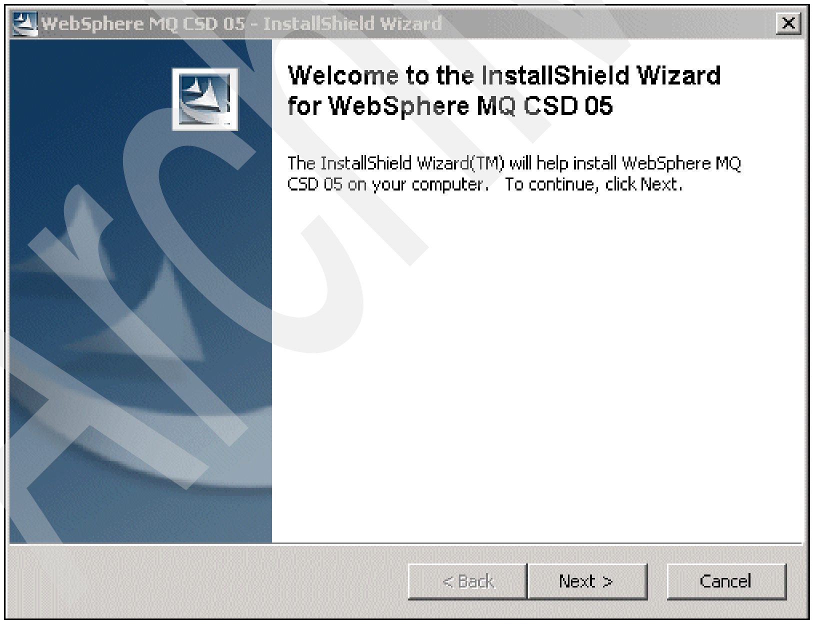


Figure 6-12 Stopping WebSphere MQ from the task bar

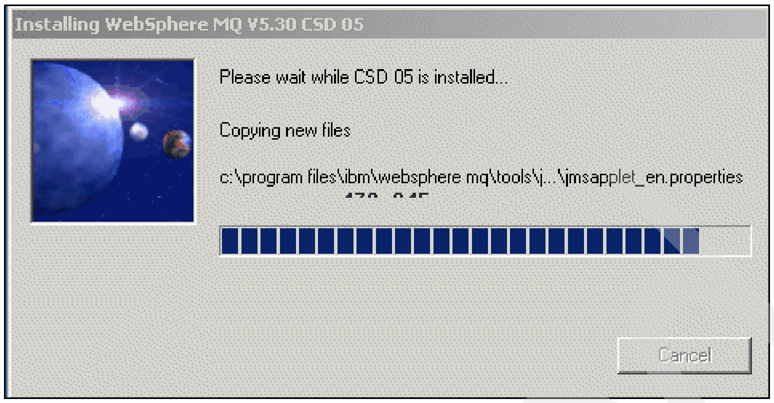
2. To install CSD05 double-click the **WebSphereMQCSD05EnUs.exe** (file name might change) executable file and click **Next** on the welcome page. Click **Next** again to uncompress the files to a temporary folder. When the files have been extracted, the CSD installer starts. Click **Install** to accept the default path for backup files and begin the install. A progress bar appears as shown in Figure 6-13 and Figure 6-14 on page 81.



Welcome to the InstallShield Wizard for WebSphere MQ CSD 05

The InstallShield Wizard(TM) will help install WebSphere MQ CSD 05 on your computer. To continue, click Next.

Figure 6-13 Installation Wizard for WebSphere MQ CSD 05


Installing WebSphere MQ V5.30 CSD 05

Please wait while CSD 05 is installed...

Copying new files

c:\program files\ibm\websphere mq\tools\j...\jmsapplet_en.properties

Figure 6-14 Installing WebSphere MQ CSD05

3. When the install is complete, click **Finish**. WebSphere MQ installation is now complete.

6.3 Installing DB2 Version 8.1

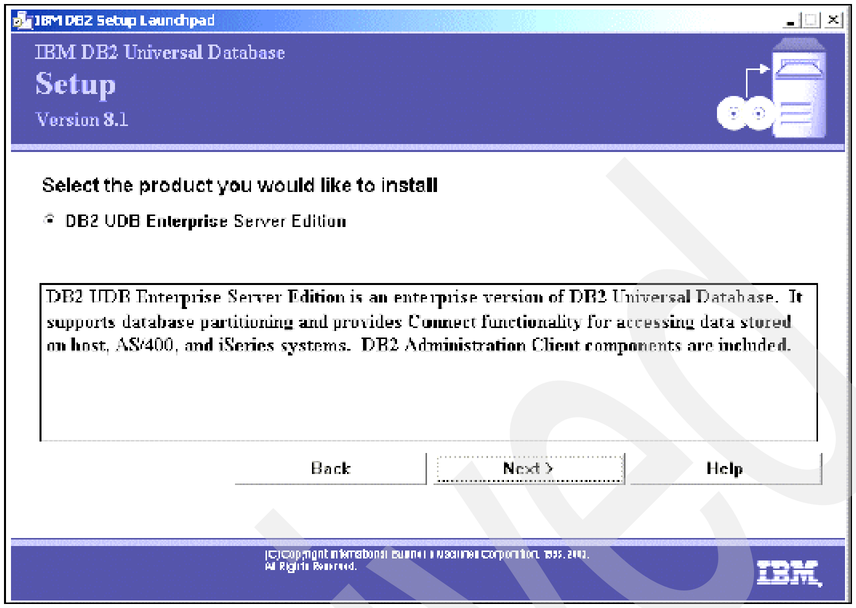
The next stage is to install the database software. Either DB2 Enterprise Server Edition Version 7.2 with Fixpak 9 or Version 8.1 with Fixpak 3 can be used. Choose to install the latest version of DB2.

Note: At the time of writing this publication, Fixpak 3 for DB2 8.1 was installed. We recommend that the latest Fixpak is installed to ensure that all of the latest updates for the product are available.

6.3.1 Starting the installation

To start the installation:

1. The DB2 Launchpad allows the start of the installation by selecting **Install Products** and then clicking **Next** to proceed, as shown in Figure 6-15 on page 82.


IBM DB2 Setup Launchpad
IBM DB2 Universal Database
Setup
Version 8.1

Select the product you would like to install

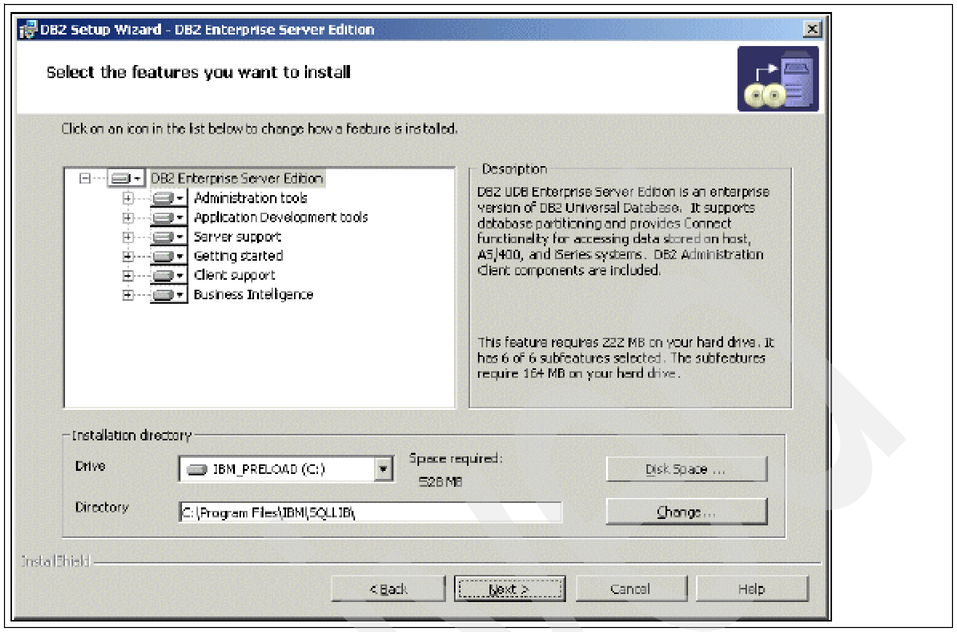
● DB2 UDB Enterprise Server Edition

DB2 UDB Enterprise Server Edition is an enterprise version of DB2 Universal Database. It supports database partitioning and provides Connect functionality for accessing data stored on host, AS/400, and iSeries systems. DB2 Administration Client components are included.

Back
Next >
Help
(C) Copyright International Business Machines Corporation, 1993, 2002.
All Rights Reserved.
IBM

Figure 6-15 Starting the DB2 installation from the Launchpad

2. After some preparation work, the initial DB2 Setup Wizard screen appears. Click **Next** to continue, and accept the license agreement in the next window.
3. Choose the type of installation required. Select a **Custom** installation, and click **Next**.
4. The next screen is to choose an installation action. Accept the default option not to create a response file and click **Next**.
5. A Custom installation allows the features to be selected that are required for the installation. The features are all selected by default. Decide whether to accept the default installation location in the window shown in Figure 6-16 on page 83 and click **Next**.


DB2 Setup Wizard - DB2 Enterprise Server Edition

Select the features you want to install

Click on an icon in the list below to change how a feature is installed.

- DB2 Enterprise Server Edition
  - Administration tools
  - Application Development tools
  - Server support
  - Getting started
  - Client support
  - Business Intelligence

Description

DB2 UDB Enterprise Server Edition is an enterprise version of DB2 Universal Database. It supports database partitioning and provides Connect functionality for accessing data stored on host, AS/400, and iSeries systems. DB2 Administration Client components are included.

This feature requires 222 MB on your hard drive. It has 6 of 6 subfeatures selected. The subfeatures require 164 MB on your hard drive.

Installation directory

Drive: IBM_PRELOAD (C:)

Space required: 528 MB

Directory: C:\Program Files\IBM\SQLLIB\

Figure 6-16 Selecting the DB2 features and installation path

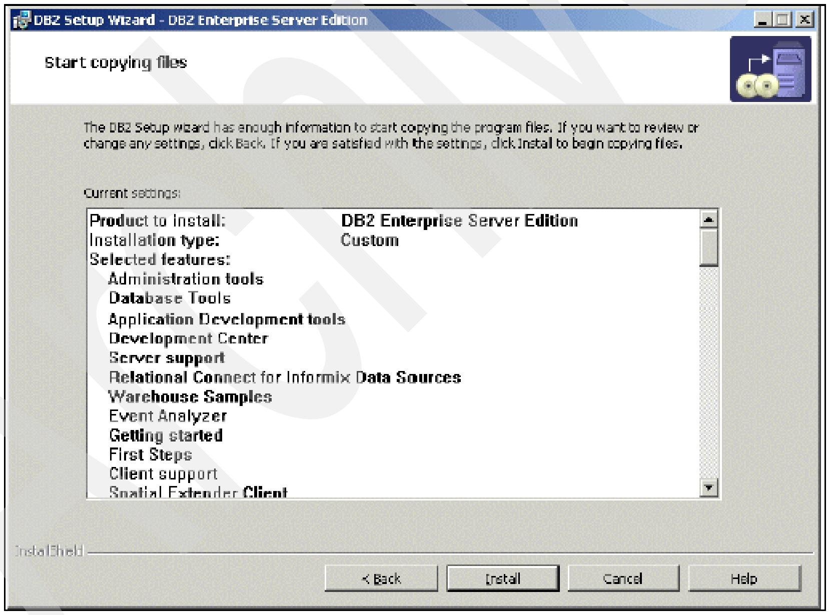
6. A warning dialog stating that there may be issues with the DB2 warehousing functionality and firewalls is displayed. There maybe some alerts from the firewall software later. These can be bypassed. Click **OK**.
7. In the next window the choice of languages for installation is given. As the installation is in English with no additional language, accept the default and click **Next**.

6.3.2 Setting installation options

To set the installation options, do the following.

1. The next step is to set the user ID and password for the DB2 Administration Server. If use of domain authentication is not required, the Domain drop-down is left blank. Accept the default user ID of db2admin and set a secure password. The same user ID and password for the other DB2 services is used so leave the check box enabled.
2. The next window is to set up an administration contact list. Accept all of the defaults (leaving the Local option enabled and the SMTP notification option disabled) and click **Next**. A warning is displayed about the notification SMTP server not being specified. This can safely be ignored. Click **OK**.

3. To create a default DB2 instance, leave the default option to create the default instance enabled and click **Next**.
4. On the following window, accept the default options to create DB2 and DB2CTLSV database instances and click **Next**.
5. The next window is to select the metadata to prepare. Choose the first option, **Prepare the DB2 tools catalog**, and click **Next**. A warning is displayed that the option to create warehouse metadata has not been chosen. Ignore this and click **OK**.
6. Specify a local database for the DB2 tools information. Accept the default options and click **Next**.
7. The next two windows are to specify a contact for the health monitor and request satellite information. As these are not required now, select the option to **Defer this task until after installation is complete** and click **Next**.
8. The installation is now ready to start. A window is displayed confirming the options and features that have been chosen, as shown in Figure 6-17. Click **Install** to start the process.



The DB2 Setup wizard has enough information to start copying the program files. If you want to review or change any settings, click Back. If you are satisfied with the settings, click Install to begin copying files.

Current settings:

```
Product to Install:          DB2 Enterprise Server Edition
Installation type:           Custom
Selected features:
  Administration tools
  Database Tools
  Application Development tools
  Development Center
  Server support
  Relational Connect for Informix Data Sources
  Warehouse Samples
  Event Analyzer
  Getting started
  First Steps
  Client support
  Spatial Extender Client
```

Figure 6-17 Confirming the DB2 installation options

9. The installation takes approximately 5 to 10 minutes to complete. Finally, a window appears to confirm that the setup has completed; click **Finish**. The First Steps window appears again; choose the option to **Exit First Steps**.

6.3.3 Installing the Fixpak 03

To install the Fixpack 03:

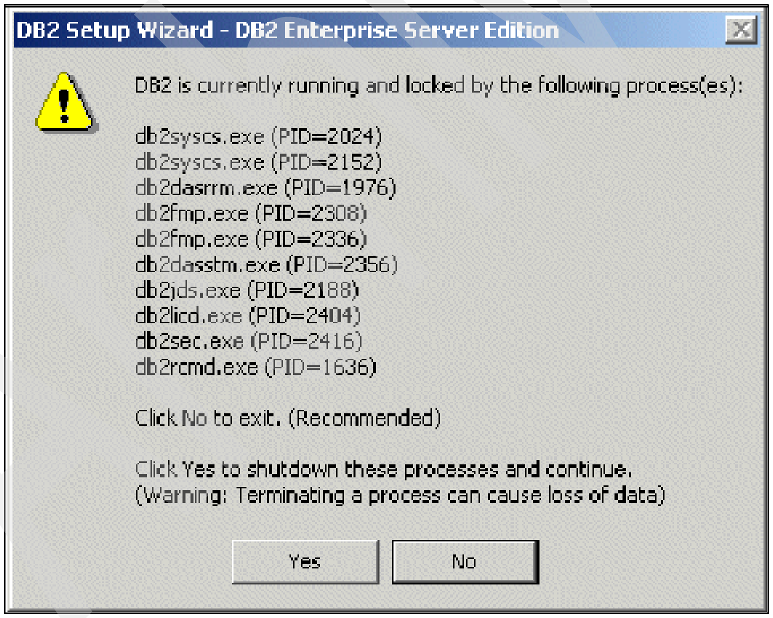
1. After double-clicking the Fixpak installation file and selecting a location to unzip the files, double-click the **FP3_WR21324_ESE.exe** file to launch the Fixpak installer. A progress dialog is displayed while the installation files are extracted and installation process is prepared.

   Note: At the time of writing this publication, the latest fix pack available was Fixpak 03. We recommend that the latest Fixpak is installed to ensure that all of the latest updates for the product are available.

   The latest fix pack is available at:

   `http://www.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/download.d2w/report`

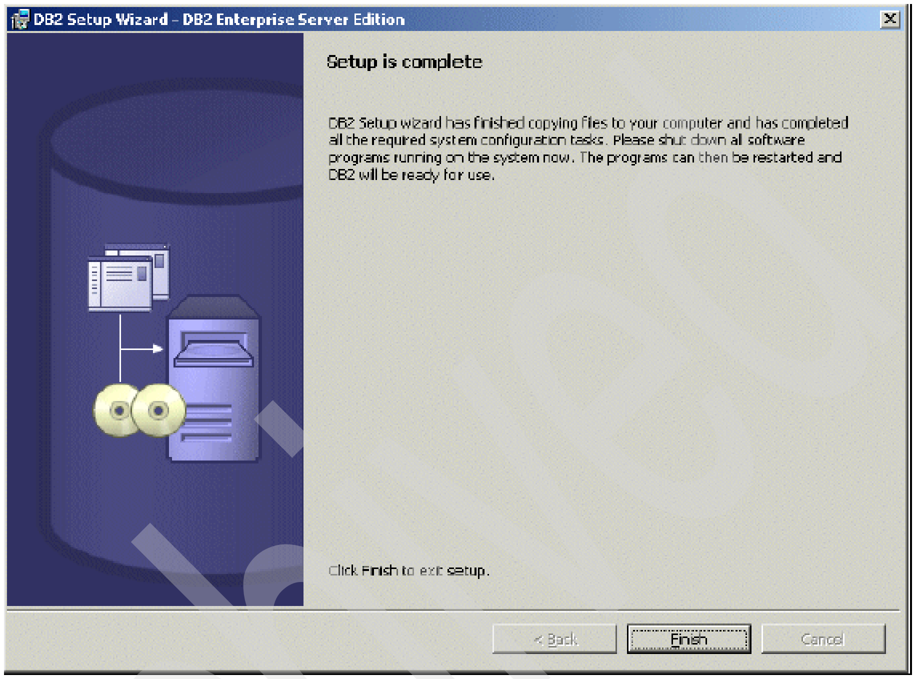
2. After the DB2 Setup Wizard appears, a dialog warns that DB2 is already running and that some processes need to be shut down (in Figure 6-18). The loss of data is not an issue because none of the databases have been created yet. Click **Yes** to terminate the processes.



```
DB2 is currently running and locked by the following process(es):

db2syscs.exe (PID=2024)
db2syscs.exe (PID=2152)
db2dasrrm.exe (PID=1976)
db2fmp.exe (PID=2308)
db2fmp.exe (PID=2336)
db2dasstm.exe (PID=2356)
db2jds.exe (PID=2188)
db2licd.exe (PID=2404)
db2sec.exe (PID=2416)
db2rcmd.exe (PID=1636)

Click No to exit. (Recommended)

Click Yes to shutdown these processes and continue.
(Warning: Terminating a process can cause loss of data)
```

Figure 6-18 Shutting down existing DB2 processes

3. Install the fixpack. A progress window is displayed. The installation takes approximately 5 to 10 minutes to complete on a system. The next window, as shown in Figure 6-19), confirms that installation is complete. Click **Finish**.



Setup is complete

DB2 Setup wizard has finished copying files to your computer and has completed all the required system configuration tasks. Please shut down all software programs running on the system now. The programs can then be restarted and DB2 will be ready for use.

Click Finish to exit setup.

Figure 6-19 Completing the DB2 installation

4. A warning dialog is displayed, reminding that all warehouse servers must be at the same code level. Click **OK**.
5. The First Steps window is displayed again. Click **Exit First Steps**.
6. The DB2 installation is now complete. Reboot the system.

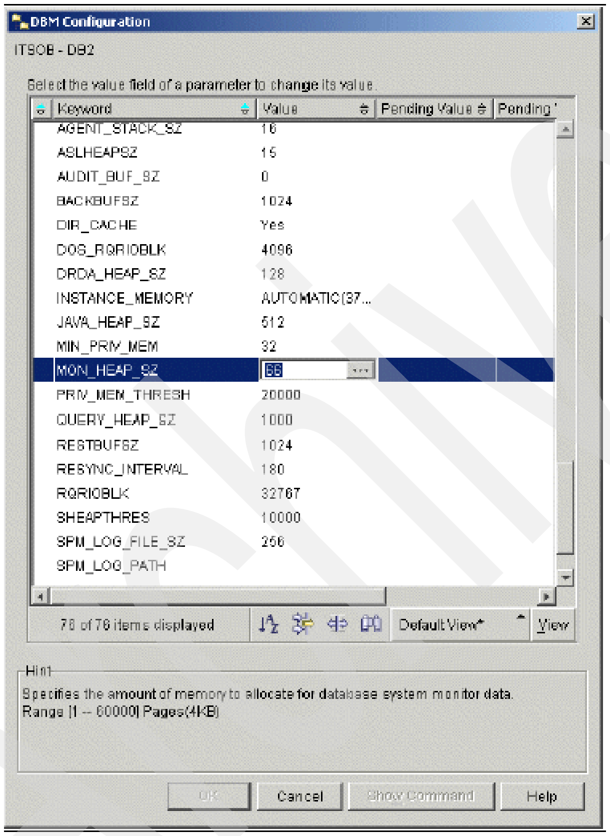
Updating the DB2 configuration parameters

Take the following steps to update the DB2 configuration parameters.

Tip: It is advised to increase the value of the database monitor heap size (MON_HEAP_SIZE) parameter for the database to prevent DB2 warnings in the Windows Event Log when running the message broker.

1. After the machine has restarted, start the DB2 Control Center by selecting **Programs -> IBM DB2 -> General Administration Tools -> Control Center** from the Windows Start menu.

2. Expand the tree on the left side of the window until the default database instance called DB2 can be seen.
3. Right-click the DB2 instance and select **Configure Parameters**... from the menu.
4. The DBM Configuration window appears. Scroll down the list until MON_HEAP_SIZE can be seen, under the Performance section, as shown in Figure 6-20.



Keyword	Value	Pending Value	Pending
AGENT_STACK_SZ	16		
ASLHEAPSZ	15		
AUDIT_BUF_SZ	0		
BACKBUFSZ	1024		
DIR_CACHE	Yes		
DOS_RQRIOBLK	4096		
DRDA_HEAP_SZ	128		
INSTANCE_MEMORY	AUTOMATIC(37...		
JAVA_HEAP_SZ	512		
MIN_PRIV_MEM	32		
MON_HEAP_SZ	66		
PRIV_MEM_THRESH	20000		
QUERY_HEAP_SZ	1000		
RESTBUFSZ	1024		
RESYNC_INTERVAL	180		
RQRIOBLK	32767		
SHEAPTHRES	10000		
SPM_LOG_FILE_SZ	256		
SPM_LOG_PATH			

Specifies the amount of memory to allocate for database system monitor data.
Range [1 -- 60000] Pages(4KB)

Figure 6-20 Viewing the database configuration

5. Click the button next to MON_HEAP_SIZE to edit the value of the parameter.
6. The default value is 66 4-KB pages, so increase it to 1000 and click **OK**.
7. Click **OK** to close the DBM Configuration window. An information window is displayed to remind you that DB2 may need to be restarted for the changes to be effective. Click **Close**.

8. Right-click the DB2 instance in the Control Center window and choose **Stop** from the menu. Check the Disconnect all applications box and click **OK**. A message is displayed, informing you that the database instance was stopped successfully. Click **Close**.
9. Finally, restart the database instance by right-clicking the DB2 instance and choosing Start. A message appears that the database instance was started successfully. Click **Close**.
10. The update to the DB2 configuration is now complete.

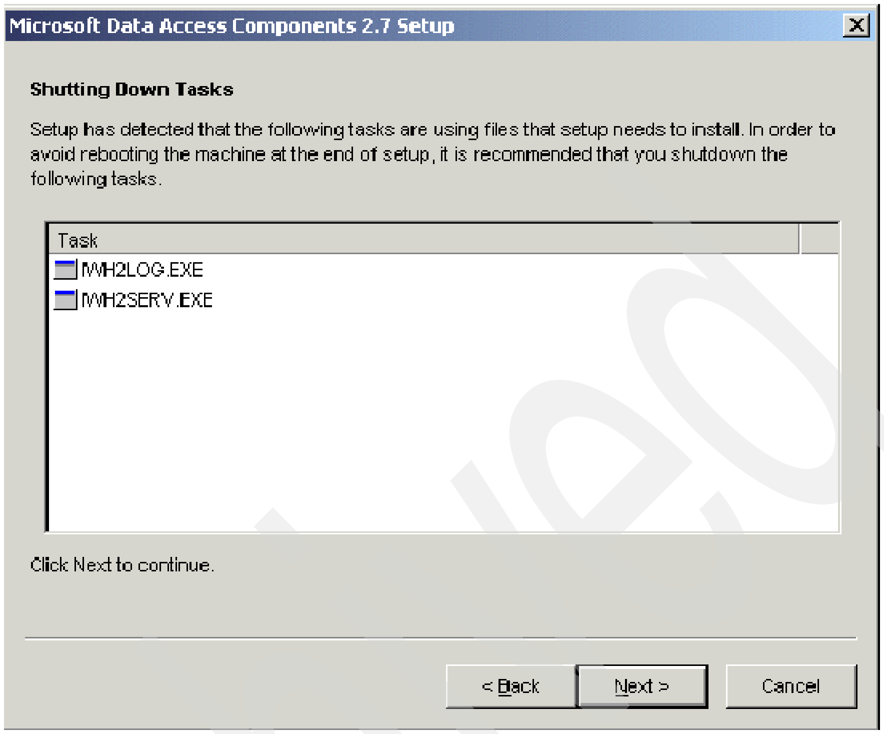
6.4 Installing Microsoft Data Access Components (MDAC) Version 2.7 SP1

WebSphere Business Integration Message Broker uses ODBC drivers that require MDAC Version 2.7 SP1. Install and setup for MDAC Version 2.7 is provided on the product supplemental CD in the \Windows\MDAC folder. It is also available from the Microsoft Web site at:

http://www.microsoft.com

The steps to install MDAC are:

1. Double-click **MDAC_TYP.EXE** in the folder.
2. When the license agreement is displayed, read it carefully and select **I accept all of the terms of the preceding licence agreement** to continue. Click **Next**.
3. A dialog is displayed to advise that some files needed by setup are in use (see Figure 6-21 on page 89). If this message is not displayed, go to Step 5.


Microsoft Data Access Components 2.7 Setup

Shutting Down Tasks

Setup has detected that the following tasks are using files that setup needs to install. In order to avoid rebooting the machine at the end of setup, it is recommended that you shutdown the following tasks.

Task
MWH2LOG.EXE
MWH2SERV.EXE

Click Next to continue.

Figure 6-21 MDAC Installation - Files in use

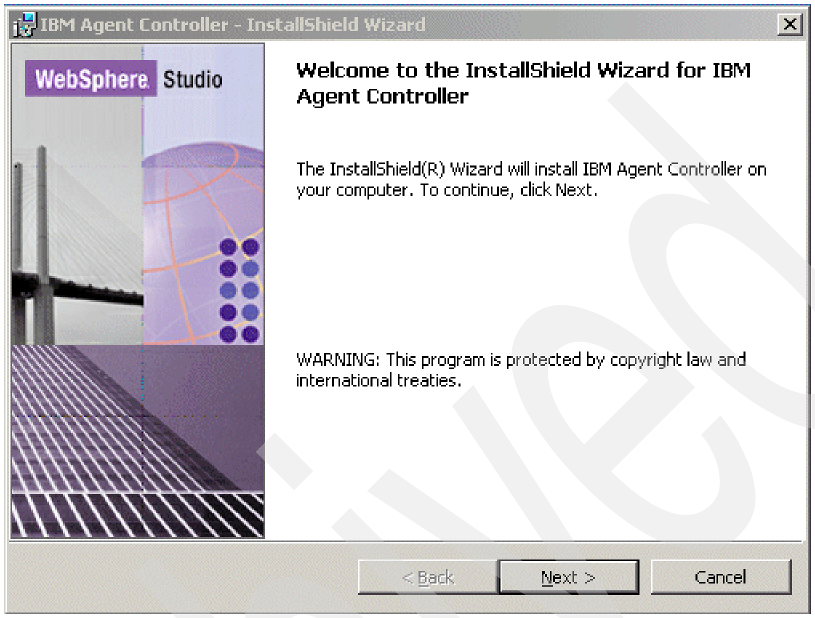
4. An `Access Denied` message is received when attempting to end these processes using the Task Manager. Click **Next** to continue the installation regardless.
5. The next window indicates what software is to be installed. Click **Finish**.
6. When the successful setup window is seen, click **Close** and reboot the system.

6.5 Installing IBM Agent Controller

Install the IBM Agent Controller if the flow debugger is to be used. If the IBM Agent Controller was not installed initially, it can be installed later. There is no need to install the IBM Agent Controller for any other reason. The install setup for this product is provided on the supplemental CD in the \Windows\RAC folder.

1. Double-click the **setup.exe** file in this folder.
2. Choose **Setup Language**. Select the language (English) and click **OK**.

3. When the welcome window is displayed, as shown in Figure 6-22, click **Next** to continue.


IBM Agent Controller - InstallShield Wizard

Welcome to the InstallShield Wizard for IBM Agent Controller

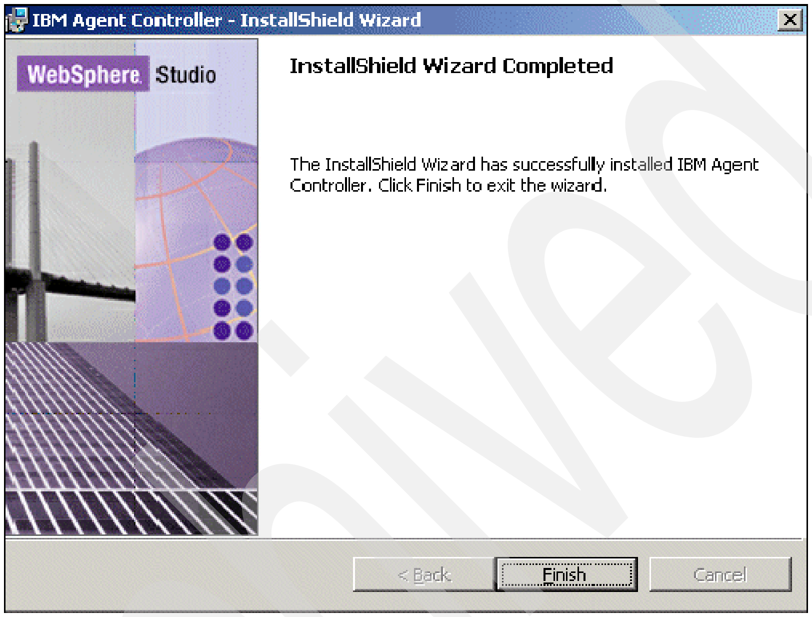
The InstallShield(R) Wizard will install IBM Agent Controller on your computer. To continue, click Next.

WARNING: This program is protected by copyright law and international treaties.

Figure 6-22 Installing IBM Agent Controller

4. Read the license agreement carefully and select **I accept the terms in the license agreement** to proceed. Click **Next** to continue.
5. The Customer Information window is displayed. Enter the user ID and organization in the appropriate fields, and indicate whether Agent Controller is being installed for All Users or the named User. Click **Next** to continue.
6. The next window is to choose a destination folder. The default directory is C:\Program Files\IBM\IBM Agent Controller\, but a different location could be selected by clicking the **Change** button.
7. The Java Runtime Environment window is displayed. Specify where the Java Runtime file (java.exe) is located, with the Browse button, or by entering the location in the text entry box. Click **Next** to continue.
8. The next window is to choose the security level. Choose **Enable Security**. Click **Next** to continue.
9. The Secure User Access is displayed. Specify the users allowed to access the Agent Controller. Enter your user ID this time. Click **Next** to continue.
10. In the Host List window, select a host type that can access the Agent Controller. Select **This Computer Only** for this time. Click **Next** to continue.

11. The Ready to Install Program windows is displayed. At this stage it is possible review or change the settings by clicking **Back**. Click **Install** to apply the chosen settings.
12. The final Install Complete window is displayed as shown in Figure 6-23. Click **Finish**.



InstallShield Wizard Completed

The InstallShield Wizard has successfully installed IBM Agent Controller. Click Finish to exit the wizard.

Figure 6-23 Installed successfully

6.6 Installing WebSphere Business Integration Message Broker

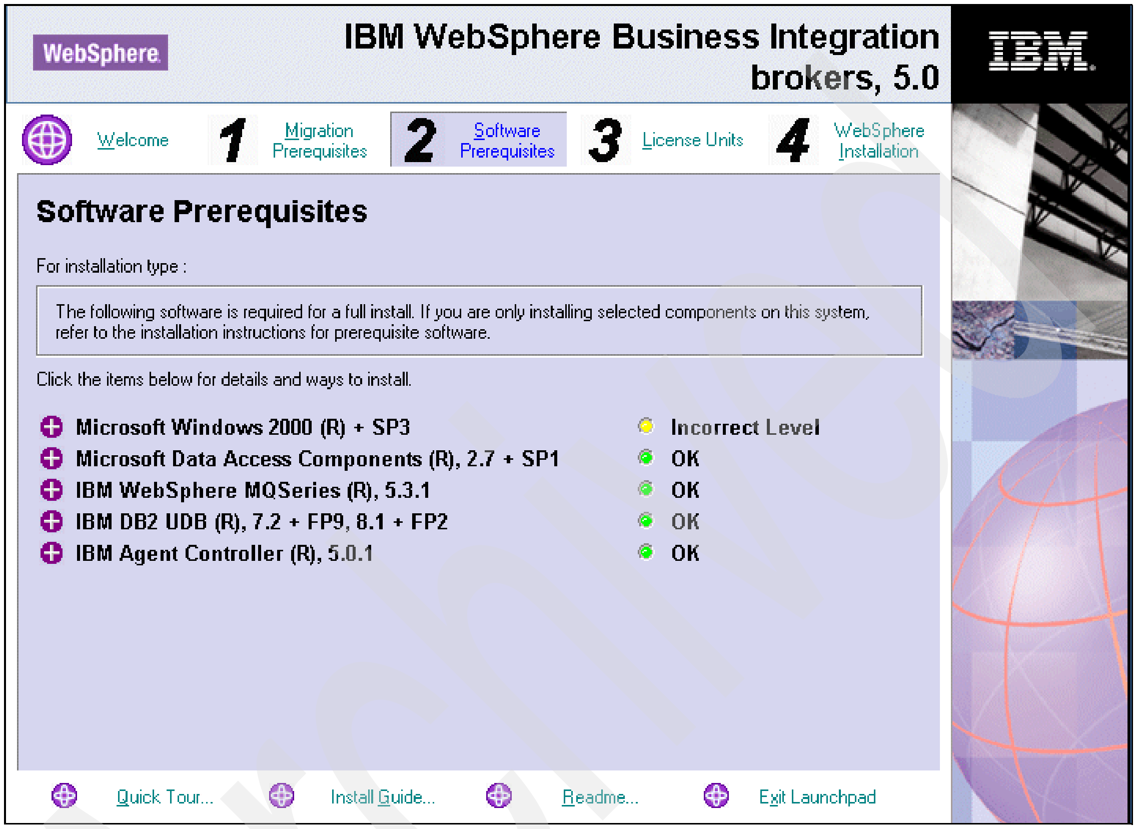
This section covers WebSphere Business Integration Message Broker installation but is applicable to WebSphere Business Integration Event Broker installation as well.

6.6.1 Preparing for installation

To prepare for installation do the following.

1. The WebSphere Business Integration Message Broker launchpad runs automatically when the CD is inserted. If it does not, double-click the **mqsilaunchpad.exe** file. The launchpad guides the installation of WebSphere Business Integration Message Broker.

2. To check the prerequisites, click **2 Software Prerequisites** on the top panel as shown in Figure 6-24. Confirm that all items are already installed to proceed.



Software Prerequisites

For installation type :

The following software is required for a full install. If you are only installing selected components on this system, refer to the installation instructions for prerequisite software.

Click the items below for details and ways to install.

Microsoft Windows 2000 (R) + SP3	Incorrect Level
Microsoft Data Access Components (R), 2.7 + SP1	OK
IBM WebSphere MQSeries (R), 5.3.1	OK
IBM DB2 UDB (R), 7.2 + FP9, 8.1 + FP2	OK
IBM Agent Controller (R), 5.0.1	OK

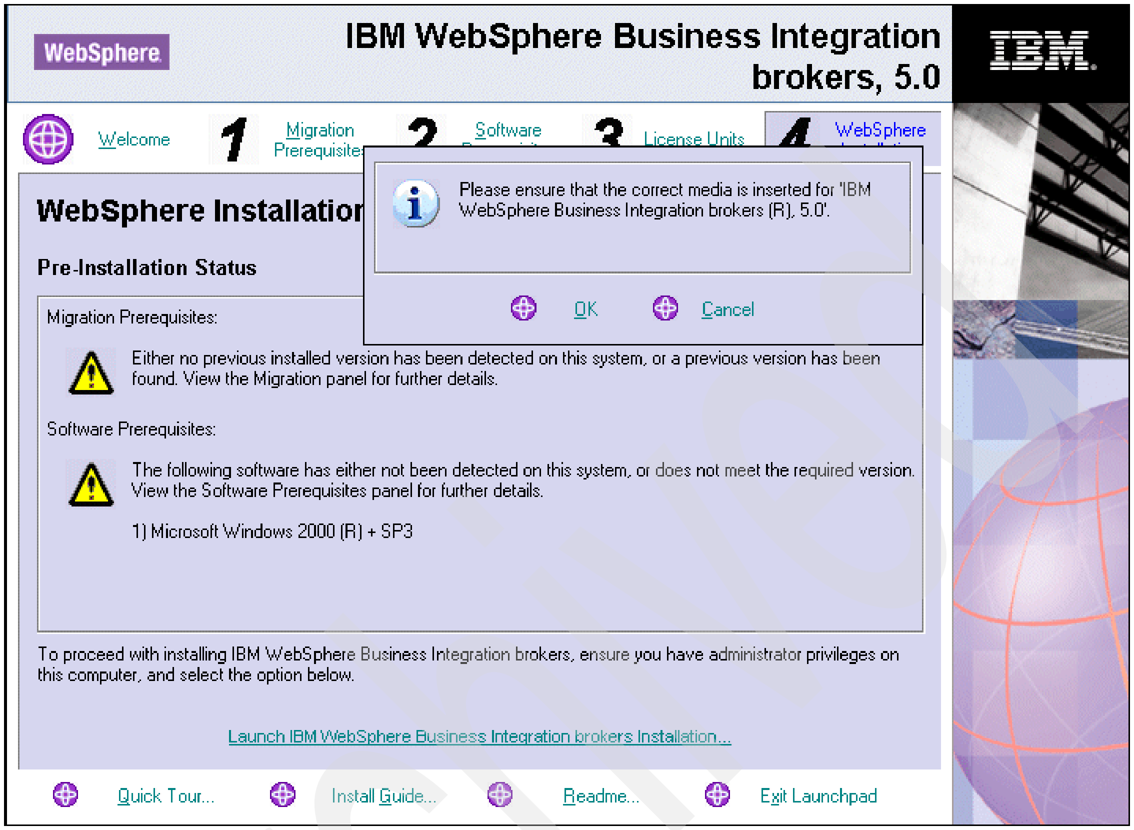
Figure 6-24 Software prerequisites

Tip: The Launchpad states that Microsoft Windows 2000 has an incorrect level of service pack, which is service pack3. However, at the time of this installation the system had been installed with service pack 4, so the message could be ignored and the installation continued.

Installation

For the installation:

1. On the top pane click **4 WebSphere Installation**. After that, **the Launch IBM WebSphere Business Integration brokers Installation** button is shown (see Figure 6-25 on page 93). Select it.



Please ensure that the correct media is inserted for 'IBM WebSphere Business Integration brokers (R), 5.0'.

WebSphere Installation

Pre-Installation Status

Migration Prerequisites:

Either no previous installed version has been detected on this system, or a previous version has been found. View the Migration panel for further details.

Software Prerequisites:

The following software has either not been detected on this system, or does not meet the required version. View the Software Prerequisites panel for further details.

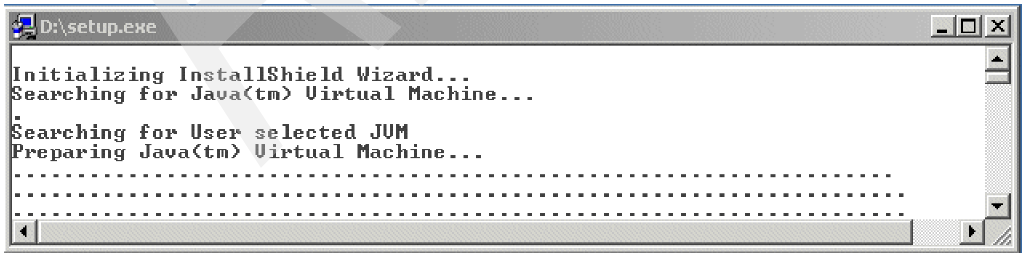
1) Microsoft Windows 2000 (R) + SP3

To proceed with installing IBM WebSphere Business Integration brokers, ensure you have administrator privileges on this computer, and select the option below.

Launch IBM WebSphere Business Integration brokers Installation...

Figure 6-25 Launchpad

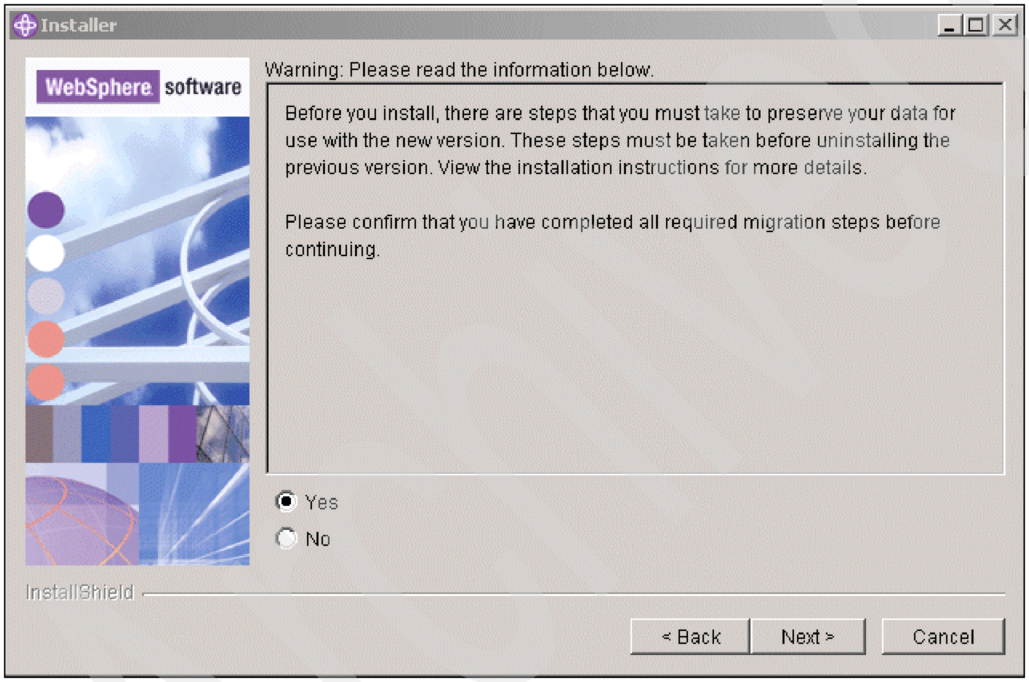
2. The information about ensuring that the right media is inserted is displayed. Click **OK**.
3. The next window shown is Initializing InstallShield Wizard, as shown in Figure 6-26. This step might take a few minutes to complete.



```
Initializing InstallShield Wizard...
Searching for Java(tm) Virtual Machine...
.
Searching for User selected JVM
Preparing Java(tm) Virtual Machine...
....................................................................
.....................................................................
.....................................................................
```

Figure 6-26 Initializing InstallShield Wizard

4. Select the language (English) and click **OK**.
5. The Welcome panel introduces the product. Click **Next** to continue.
6. A warning dialog box is displayed (see Figure 6-27). If migrating from a previous version (that is WebSphere MQ Integrator V2.1 or WebSphere Business Integration Event Broker V2.1), ensure that all the required migration steps have been completed. When ready, or if a previous version is not installed, select the **Yes** radio button and click **Next** to continue. If the installation needs to be cancelled then leave the **No** radio button selected and click **Cancel**.



Warning: Please read the information below.

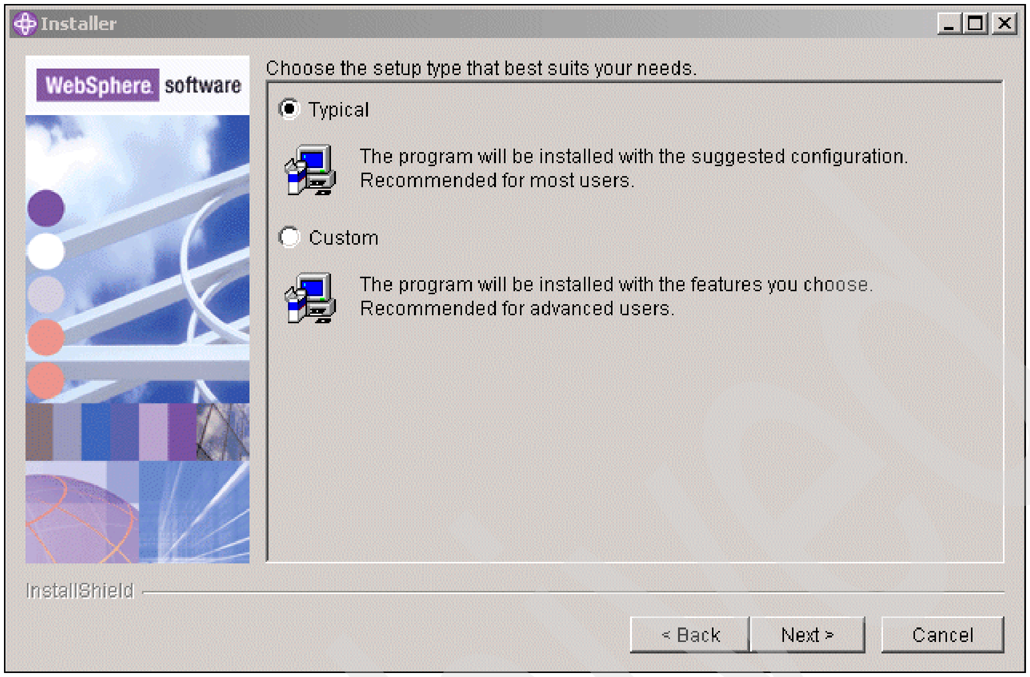
Before you install, there are steps that you must take to preserve your data for use with the new version. These steps must be taken before uninstalling the previous version. View the installation instructions for more details.

Please confirm that you have completed all required migration steps before continuing.

[✓ Yes] No

Figure 6-27 Migration warning

7. Read the licence agreement carefully, then select the **Yes** radio button and click **Next** to continue.
8. The next window is to choose install location. The default directory is C:\Program Files\IBM\WebSphere Business Integration Message Broker, but a different location can be selected by clicking the Browse button. Click **Next** to continue.
9. The Selecting type of installation window is displayed. Either a Typical or Custom is offered. Select **Typical** for this time (see Figure 6-28 on page 95).



Choose the setup type that best suits your needs.

[✓ Typical]

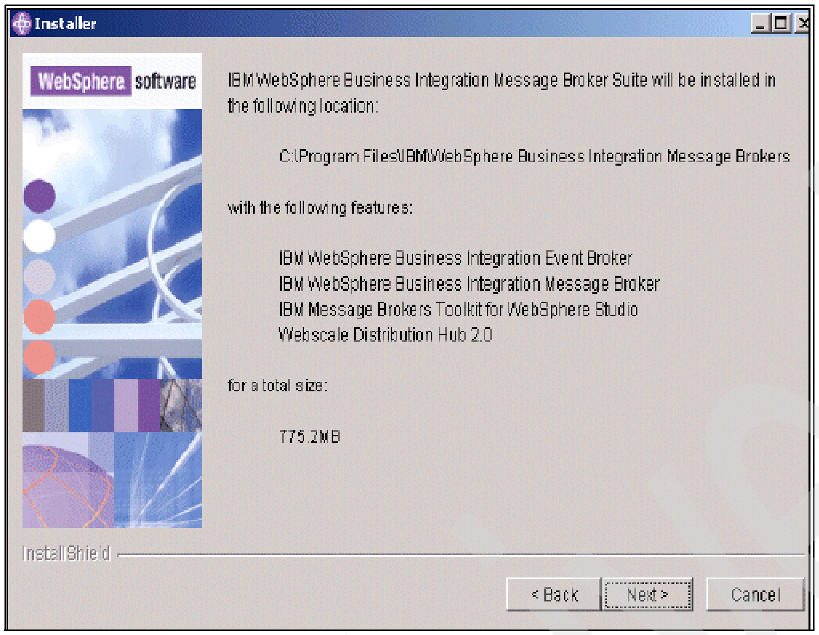
The program will be installed with the suggested configuration.
Recommended for most users.

Custom

The program will be installed with the features you choose.
Recommended for advanced users.

Figure 6-28 Typical installation

10. The Install Summary is displayed. The options selected during the previous panels are displayed with an indication of the total disk space required in WebSphere Business Integration Message Broker and the location chosen to install the product. The option to go back and change the settings is available at this stage, but it is not selected. Click **Next** to install (see Figure 6-29 on page 96).


Installer

IBM WebSphere Business Integration Message Broker Suite will be installed in the following location:

C:\Program Files\IBM\WebSphere Business Integration Message Brokers

with the following features:

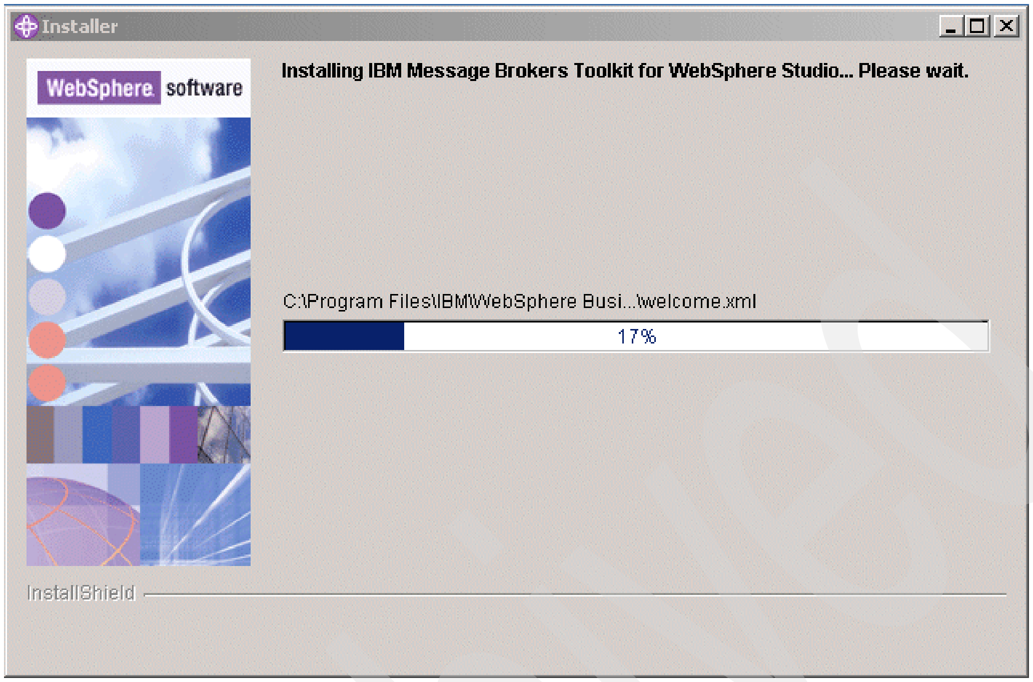
IBM WebSphere Business Integration Event Broker
IBM WebSphere Business Integration Message Broker
IBM Message Brokers Toolkit for WebSphere Studio
Webscale Distribution Hub 2.0

for a total size:

775.2MB

Figure 6-29 Installation summary

11.A sample progress window is shown in Figure 6-30 on page 97.


Installer

Installing IBM Message Brokers Toolkit for WebSphere Studio... Please wait.

C:\Program Files\IBM\WebSphere Busi...\welcome.xml

17%

Figure 6-30 Installing IBM WebSphere Business Integration Brokers 5.0

12. The final Install Complete window is displayed. Click **Finish**.

6.7 Installing product updates

After installing WebSphere Business Integration Message Broker, ensure that the product is up to date by installing any updates that are available.

The CSD install process is almost identical to the full product install process.

Note: At the time of writing this publication, the update available was CSD03. We recommend that the latest WebSphere Business Integration Message Broker update is applied to ensure that all of the latest updates for the product are available.

6.7.1 Installing CSD03

To install CDD03:

1. Insert the CSD03 CD. The WebSphere Business Integration Message Brokerr launchpad window is displayed. If it is not, then run mqsilaunchpad.exe.

   The latest CSD for WebSphere Business Integration Message Broker is available at:

   http://www.ibm.com/software/integration/mqfamily/support/summary/wbib.html

2. The IBMWebSphere Business Integration Brokers, 5.0 launchpad is displayed. Click Option **4 WebSphere Installation** at the top of the screen to confirm that WebSphere Business Integration Message BrokerV5 is already installed, then Click **Launch IBM WebSphere Business Integration brokers Installation...** (near the bottom of the window) to start the install.
3. A dialog box prompts to ensure that the correct media is inserted. Click **OK**.
4. At the Installer dialog box, select a language to install (English). Click **OK**.
5. Click **Next** when the Welcome window is displayed.
6. Select the **Yes** radio button when the Migration Warning is displayed Click **Next**.
7. Read the licence agreement, then select the **Yes** radio button to accept the license and click **Next** to continue.
8. Select the type of installation. Select either a Typical or Custom install. Select **Typical** and click **Next**.
9. Select all components and click **Next**.
10. A summary of the install options is displayed. Click **Next** to begin the install.
11. When installation is complete, click **Finish** to exit setup, and then exit the Launchpad.

6.7.2 Checking for new updates from within the Toolkit

New updates for the Message Broker Toolkit are posted to a publicly available update site. The installation and management of these updates is done from within the Toolkit.

Within the Message Broker Toolkit functionality is provided as a set of plug-ins grouped together as features. A feature is the smallest unit of separately downloadable and installable functionality.

The Toolkit provides an update manager that checks a product's update site for feature updates of existing features and manages their installation. Internet access is required to use this functionality.

1. From within the Message Broker Toolkit, click **Help -> Software Updates -> New Updates** as shown in Figure 6-31.

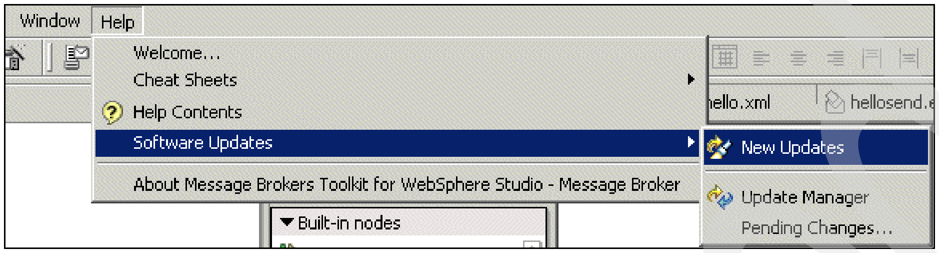
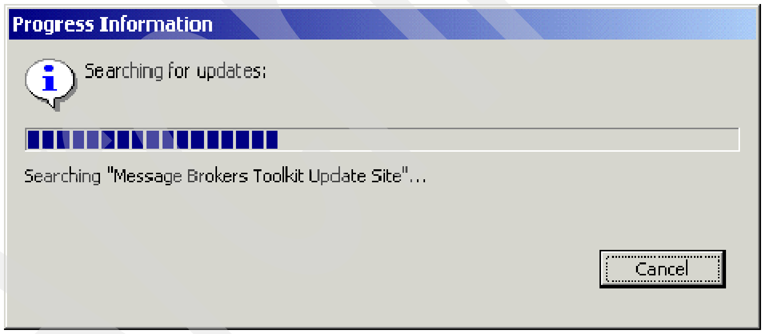


Figure 6-31 Starting an update search

The update manager checks product Web sites to find out if there are any later versions of the currently installed features. This is done for all products that are installed and have a registered update site with the update manager.

The progress of the search is displayed as the sites are checked, as shown in Figure 6-32.

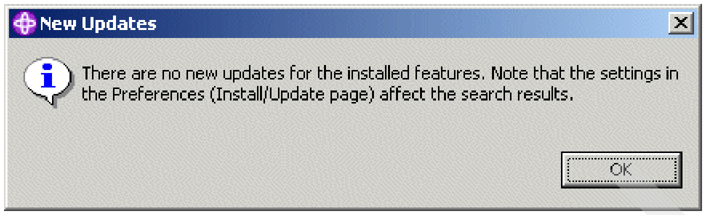

Progress Information

Searching for updates:

Searching "Message Brokers Toolkit Update Site"...

Figure 6-32 Searching for feature updates

2. If there are no updates available, as shown in Figure 6-33 on page 100, click **OK** to complete the update search.



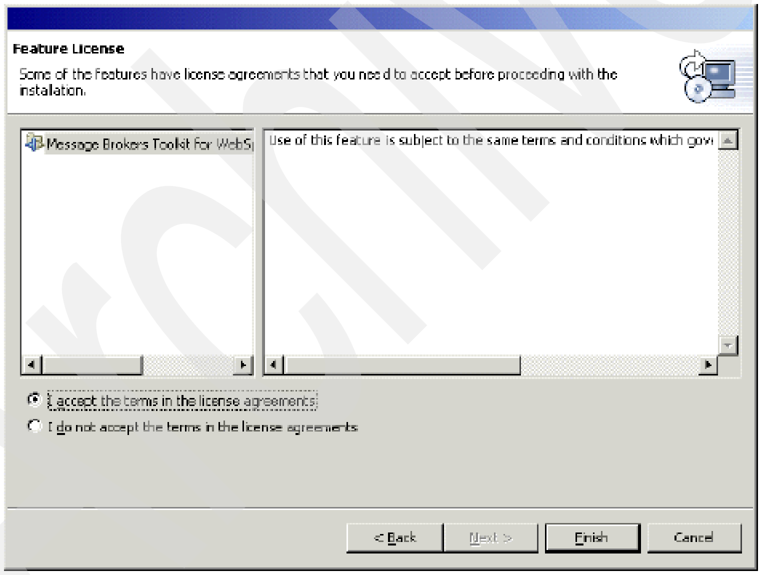
There are no new updates for the installed features. Note that the settings in the Preferences (Install/Update page) affect the search results.

Figure 6-33 No new updates to install

3. If there are feature updates available then a dialog box showing available updates is shown, allowing the requested updates to be selected. Select the features to upgrade and click **Next**.

   License agreements for each of the selected features are displayed with an explicit requirement to accept or reject the license.

4. Select the **I accept the terms in the license agreements** radio button as shown in Figure 6-34 and click **Finish**.



Feature License

Some of the features have license agreements that you need to accept before proceeding with the installation.

Message Brokers Toolkit for WebS

Use of this feature is subject to the same terms and conditions which gov

[✓ I accept the terms in the license agreements]

I do not accept the terms in the license agreements

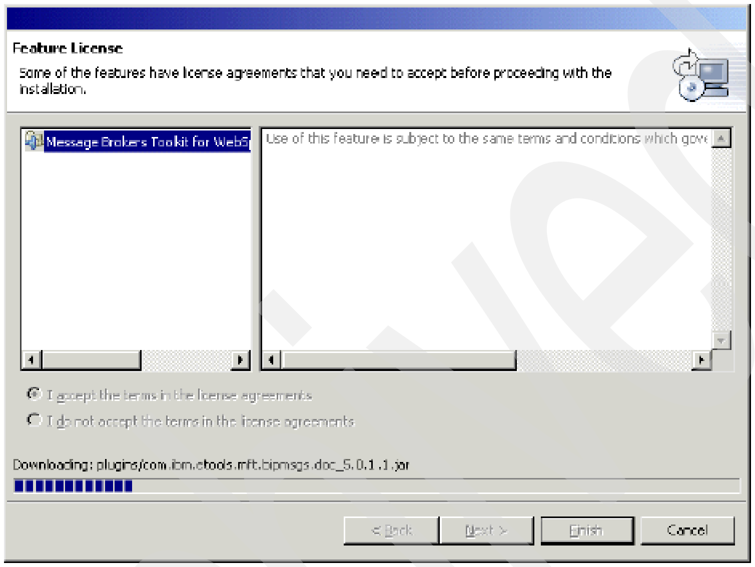
Figure 6-34 Accepting the feature license

A feature verification screen is displayed. This contains full details of the feature to be installed and whether the feature has been digitally signed by the feature provider. Digital signing allows verification that the feature to be downloaded and installed is from a trusted provider.

5. Click **Install** to download and install the feature.

Note: A separate feature verification dialog is displayed for each feature to be installed.

As the download and install occurs, the progress information is displayed. A sample screen is shown in Figure 6-35.


Feature License

Some of the features have license agreements that you need to accept before proceeding with the installation.

Message Brokers Toolkit for WebS
Use of this feature is subject to the same terms and conditions which gove
I accept the terms in the license agreements
I do not accept the terms in the license agreements

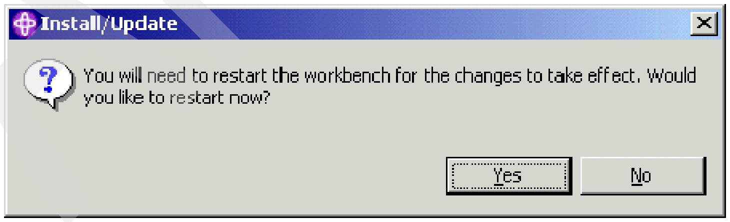
Downloading: plugins/com.ibm.etools.mft.bipmsgs.doc_5.0.1.1.jar

< Back
Next >
Finish
Cancel

Figure 6-35   Downloading feature

After all the features have been downloaded and installed the workbench needs to be restarted to load the new configuration, as shown in Figure 6-36.

6. Click **Yes** to restart the workbench with the new features. The workbench shuts down and then restarts, at which time the new features are available.



You will need to restart the workbench for the changes to take effect. Would you like to restart now?

Figure 6-36   Restart workbench to accept new configuration

The update is now complete.

7

Configuration and administration of WBI Event Broker

This chapter describes the process of configuring WebSphere Business Integration Event Broker on a single Windows machine using the Getting Started Wizard. It also explains the process of developing and deploying a message flow using the Message Brokers Toolkit.

The scenarios shown in the following chapters demonstrate the use of WebSphere Business Integration Event Broker. The objective of this publication is to demonstrate the Pub/Sub features of WebSphere Business Integration Event Broker. WebSphere Business Integration Message Broker contains all these functions too, as well as additional message transformation and enrichment features that considerably add to the power of the product. WebSphere Business Integration Event Broker is fully contained in WebSphere Business Integration Message Broker, and the message processing engine for both products is the same.

Note: Before working on IBM WebSphere Business Integration Message Broker V5.x.x, open the IBM WebSphere Business Integration Message Broker toolkit, click the **Help** tab, and click **About WebSphere Message Brokers toolkit**.

Check the version, which has to be:

```
Version: 5.0.2
Build id: 20031009_1516
```

If a previous version is displayed, for example:

```
Version: 5.0.0
Build id: 20030502_1315
```

Then perform the following steps:

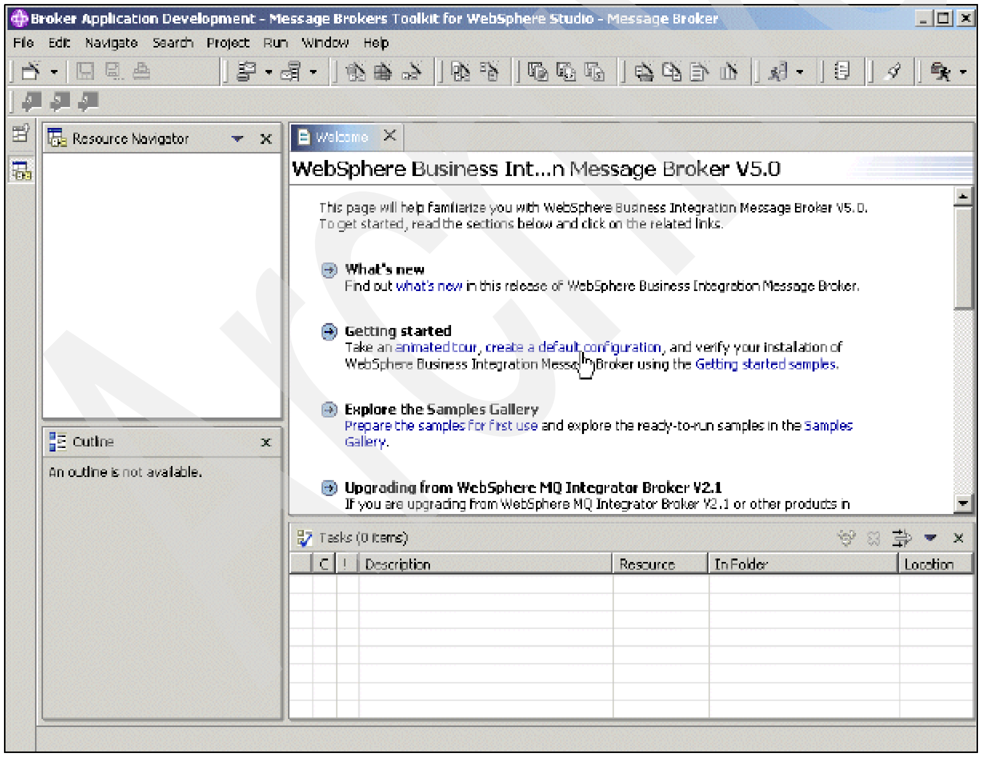
1. Close the Message Broker Toolkit for WebSphere Studio.
2. Open the workspace folder C:\Program Files\IBM\WebSphere Business Integration Message Broker\eclipse\workspace.
3. Delete the .metadata folder completely.
4. Re-open the IBM WebSphere Business Integration Message Broker toolkit and check the version. The latest updated plug-ins are displayed.

7.1 Creating the default configuration

This section illustrates how to create a default configuration on a single Windows machine by using the WebSphere Business Integration Message Broker Getting Started Wizard. A Configuration Manager and a Broker is created and a connection to the workbench is established so that WebSphere Business Integration Message Broker resources can be deployed to the broker. The Getting Started Wizard creates the necessary WebSphere MQ, DB2 and Windows resources required by the Configuration Manager and broker. These products were available and installed as discussed earlier in Chapter 6, “Installation” on page 67.

7.1.1 Using the Getting Started Wizard

Start the WebSphere Business Integration Message Broker Toolkit with **Start -> Programs -> IBM WebSphere Business Integration Message Brokers -> Message Brokers Toolkit**. See Figure 7-1 for an example of the Toolkit.



WebSphere Business Int...n Message Broker V5.0

This page will help familiarize you with WebSphere Business Integration Message Broker V5.0.
To get started, read the sections below and click on the related links.

What's new
Find out what's new in this release of WebSphere Business Integration Message Broker.

Getting started
Take an animated tour, create a default configuration, and verify your installation of WebSphere Business Integration Message Broker using the Getting started samples.

Explore the Samples Gallery
Prepare the samples for first use and explore the ready-to-run samples in the Samples Gallery.

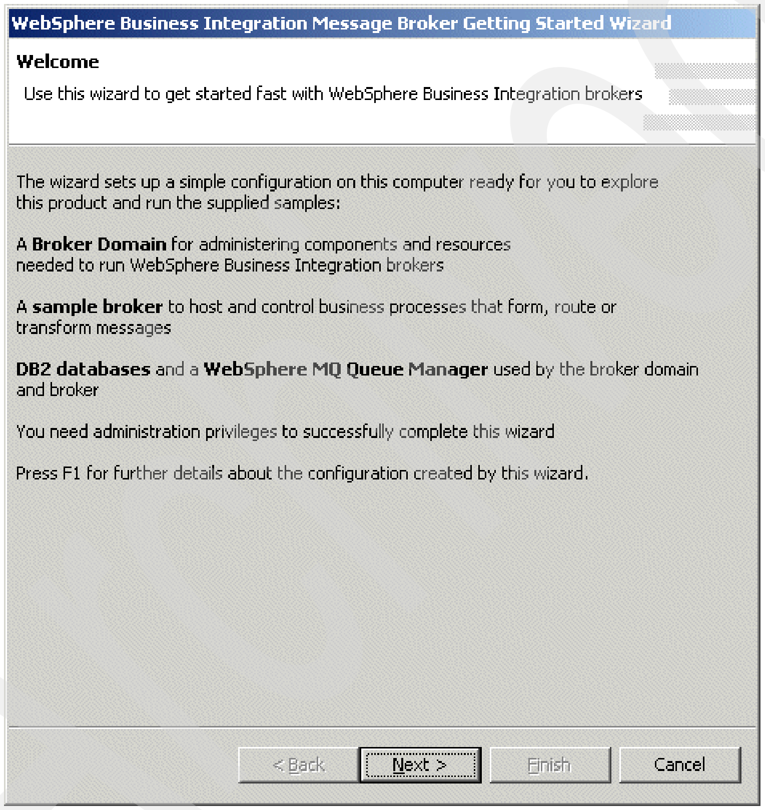
Upgrading from WebSphere MQ Integrator Broker V2.1
If you are upgrading from WebSphere MQ Integrator Broker V2.1 or other products in

An outline is not available.

Figure 7-1 Getting started

The Getting Started Wizard for WebSphere Business Integration Message Broker V5 was used by clicking the **Create a default configuration** hyperlink from the Welcome page in the workbench. This presents the Getting Started Wizard initial dialogue box as shown in Figure 7-2.

Note: If the Welcome page (as shown in Figure 7-1) is not displayed, open it by selecting **Help -> Welcome** from the title bar on the Message Brokers Toolkit workbench.


WebSphere Business Integration Message Broker Getting Started Wizard
< Back
Next >
Finish
Cancel

Welcome

Use this wizard to get started fast with WebSphere Business Integration brokers

The wizard sets up a simple configuration on this computer ready for you to explore this product and run the supplied samples:

A **Broker Domain** for administering components and resources needed to run WebSphere Business Integration brokers

A **sample broker** to host and control business processes that form, route or transform messages

DB2 databases and a **WebSphere MQ Queue Manager** used by the broker domain and broker

You need administration privileges to successfully complete this wizard

Press F1 for further details about the configuration created by this wizard.

Figure 7-2 Starting the wizard

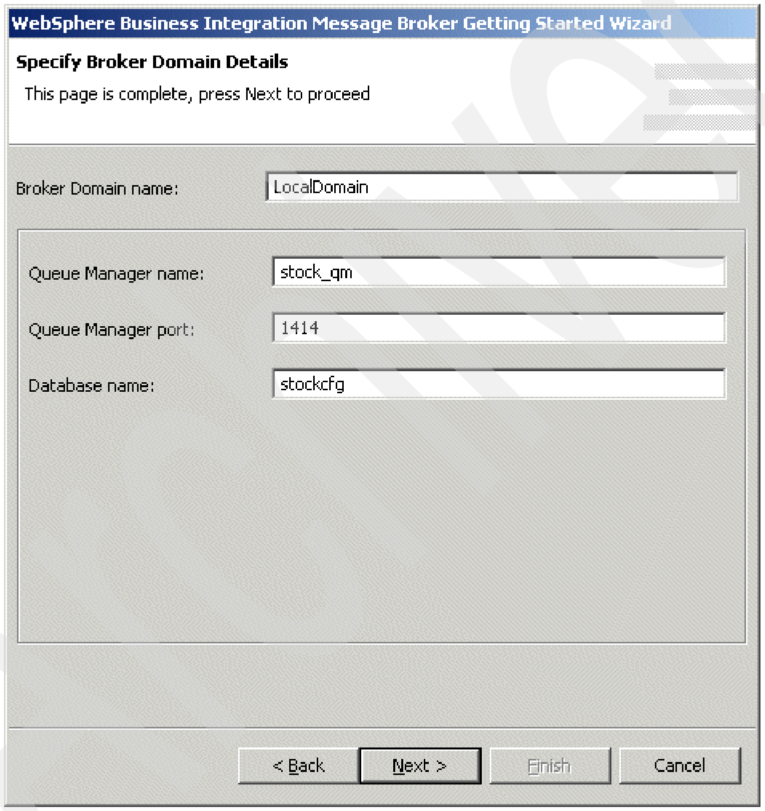
Steps to get started:

1. Click the **Next** button to go to the WebSphere Business Integration Message Broker Services User Account dialogue.
2. Select the **New User Account** option, and click **Next**. If an existing user ID is already a member of all the required groups (mqm, mqbrasgn, mqbrdevt,

mqbrkrs, mqbrops and mqbrtpic) then use that user ID. Use this account for accessing the DB2 databases as well.

3. The next dialogue is where the domain name information is required. Enter the Queue Manager name `stock_qm`, enter the port number `1414` for the WebSphere MQ listener as default, and the database name `stockcfg`, as shown in Figure 7-3.

> **Note:** If the queue manager is already created, delete the listener for this queue manager, as this process creates a new listener.



WebSphere Business Integration Message Broker Getting Started Wizard

Specify Broker Domain Details

This page is complete, press Next to proceed

Broker Domain name: LocalDomain

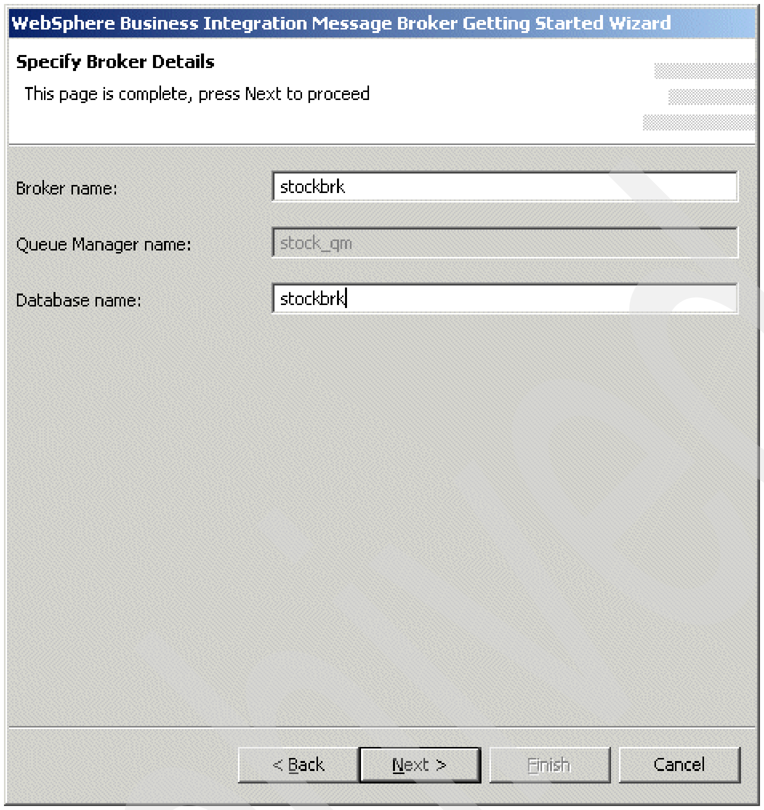
Queue Manager name: stock_qm

Queue Manager port: 1414

Database name: stockcfg

Figure 7-3 Domain details

4. Click **Next** to move to the Broker Details dialogue box. Provide the names for the broker and the database (`stockbrk` and `stockbrk`), as shown in Figure 7-4 on page 108.


WebSphere Business Integration Message Broker Getting Started Wizard

Specify Broker Details

This page is complete, press Next to proceed

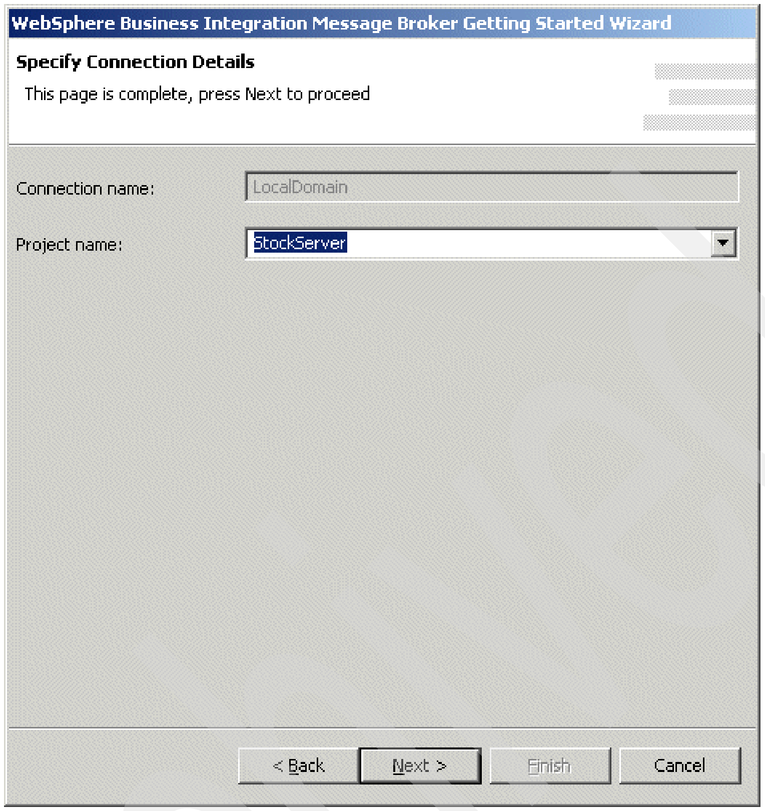
Broker name: stockbrk

Queue Manager name: stock_qm

Database name: stockbrk

Figure 7-4 Broker details

5. Click **Next** to move to the next dialogue box and provide the Connection Details by giving the project name (`StockServer`) as shown in Figure 7-5 on page 109.


WebSphere Business Integration Message Broker Getting Started Wizard

Specify Connection Details

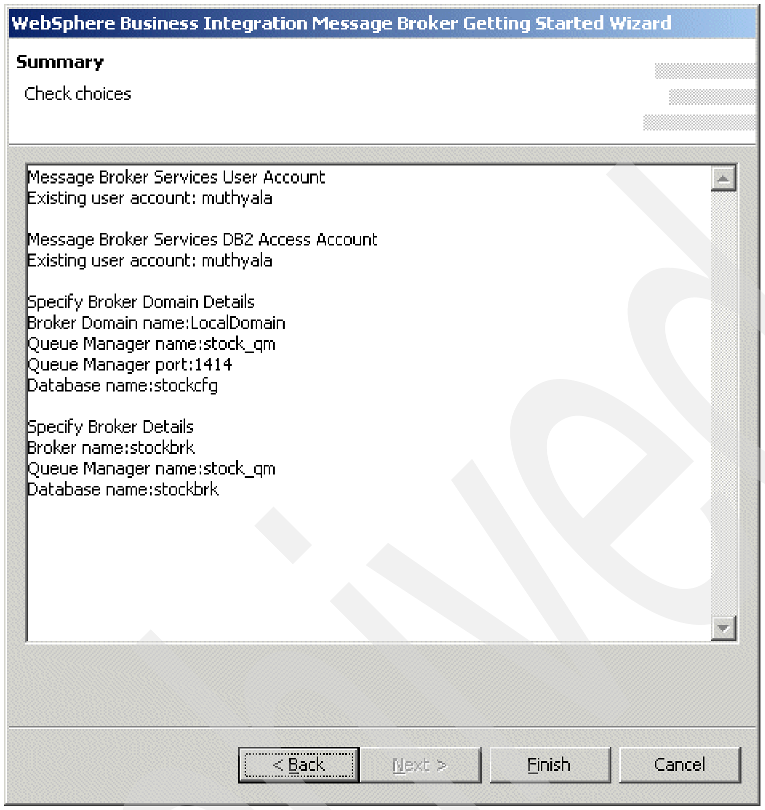
This page is complete, press Next to proceed

Connection name: LocalDomain

Project name: StockServer

Figure 7-5 Connection details

6. The Summary dialogue is displayed as shown in Figure 7-6 on page 110.


WebSphere Business Integration Message Broker Getting Started Wizard

Summary

Check choices

```
Message Broker Services User Account
Existing user account: muthyala

Message Broker Services DB2 Access Account
Existing user account: muthyala

Specify Broker Domain Details
Broker Domain name:LocalDomain
Queue Manager name:stock_qm
Queue Manager port:1414
Database name:stockcfg

Specify Broker Details
Broker name:stockbrk
Queue Manager name:stock_qm
Database name:stockbrk
```

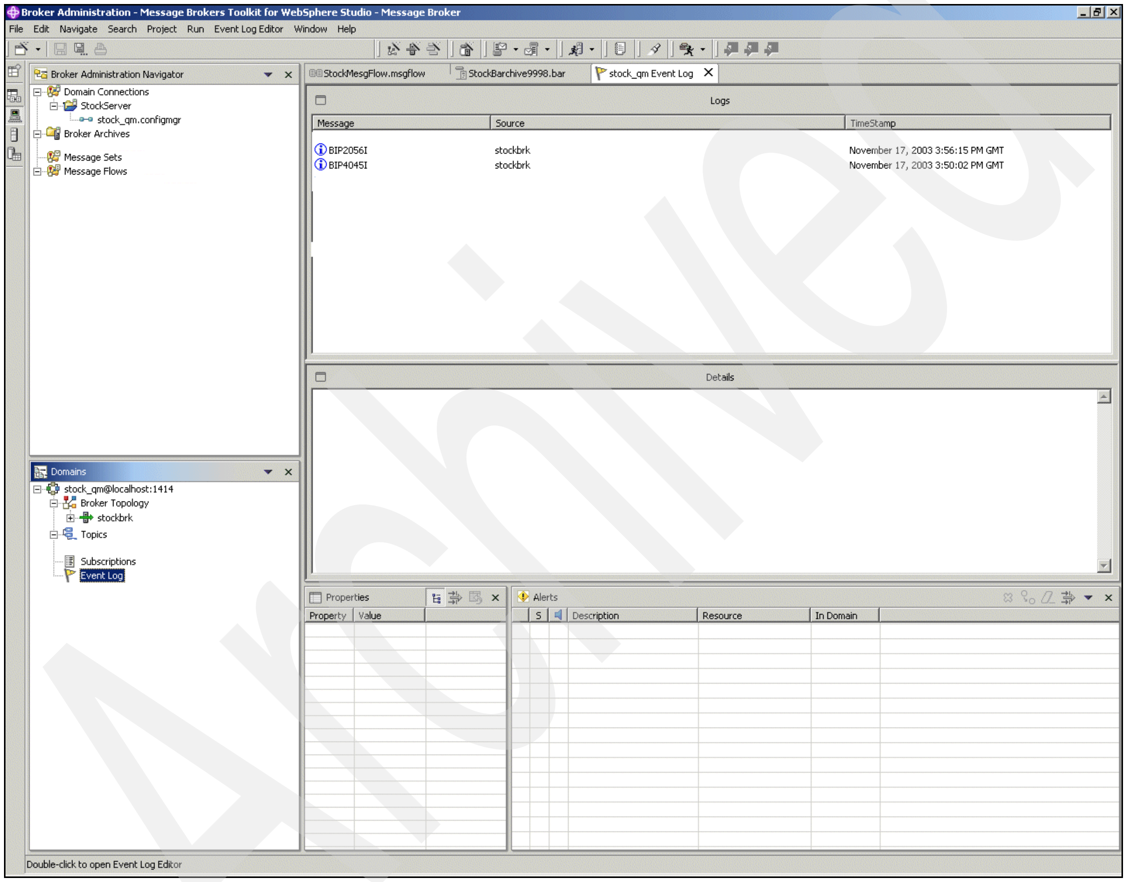
Figure 7-6 Summary

7. Click the **Finish** button to start the configuration process.
8. When the configuration process has completed, a message box is displayed. Any errors encountered are reported here.
9. A message is displayed to ask how the configuration must be deployed. Click the **Complete** button.
10. When the deploy has been initiated, another dialogue shows the status of the deploy. Click the **Details** button to show which resources were deployed.
11. Click **OK** on the Getting Started Wizard Success box to return to the WebSphere Business Integration Message Broker workbench. If the configuration completed successfully a new project named StockServer appears with a connection to the Configuration Manager.
12. Use the **Window -> Open Perspective -> Broker Administration menu** (or click the Open a Perspective icon on the left-hand toolbar) to open the Broker

Administration perspective. This perspective selection and opening of perspectives is shown in detail in Figure 7-15 on page 120.

13. Expand the domain tree in the Domains window to show the broker and the default execution group that has been created.

14. Double click **Event Log** in the Domains window to see the messages associated with the deploy. If the deploy is successful, two information messages (BIP2056I and BIP4045I) indicate this, as shown in Figure 7-7.



Message	Source	TimeStamp
BIP2056I	stockbrk	November 17, 2003 3:56:15 PM GMT
BIP4045I	stockbrk	November 17, 2003 3:50:02 PM GMT

Double-click to open Event Log Editor

Figure 7-7 Event Log to check for the result of deploy operation

7.2 Creating a broker from the command line

Before creating the Configuration Manager and the Broker using the command prompt, create the databases for the Configuration Manager and Broker first by using DB2. Select **Start -> Programs -> IBM DB2 -> General Administrative Tools -> Control Center** and go the Control Center, as shown in Figure 7-8. Create the databases for the Configuration Manager and the broker using the wizard.

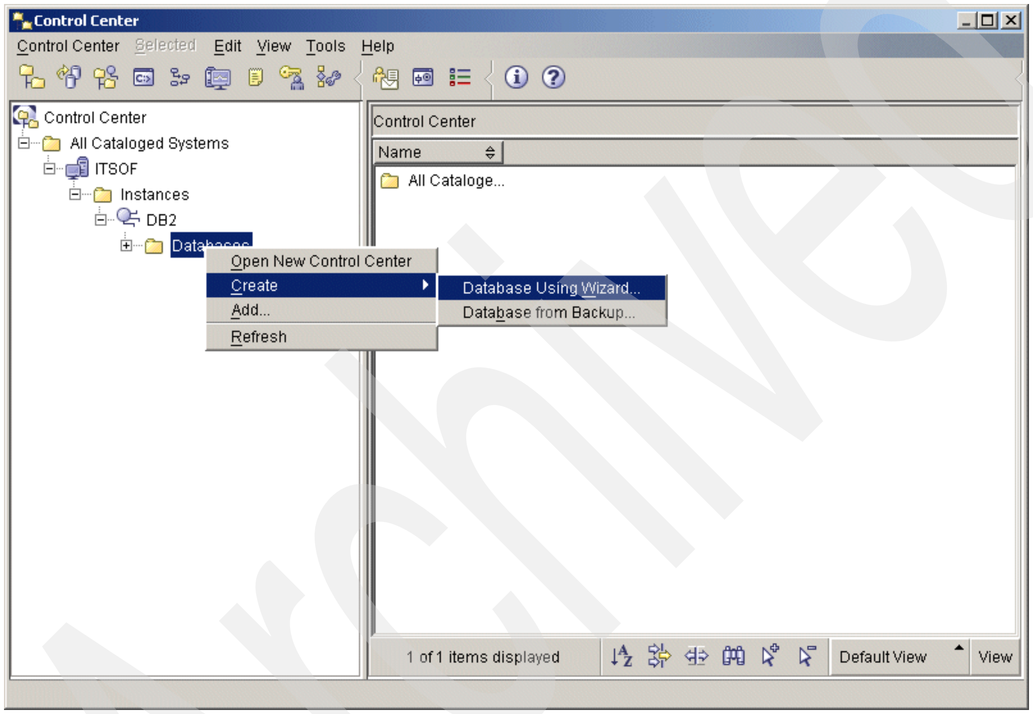
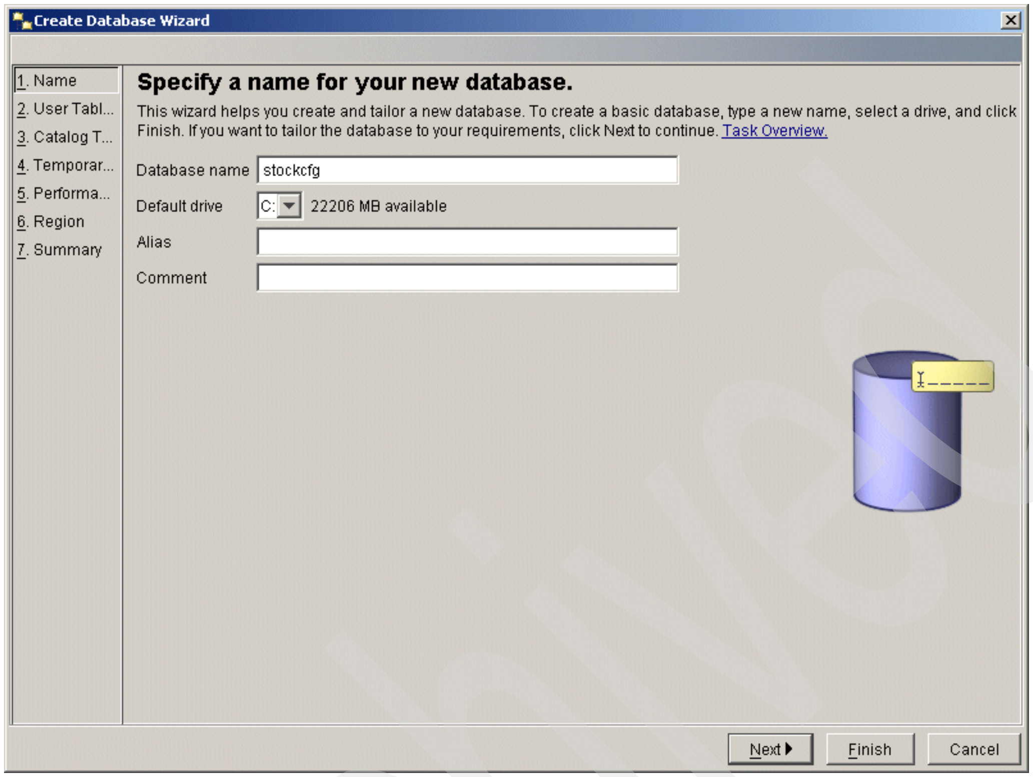


Figure 7-8 DB 2 Control Center

After opening the Control Center go to Database and right click **Database**. Then go to **Create -> Database using Wizard** and select this option. A new dialogue box is presented. Provide the database name and click **Finish**. The database names chosen were stockcfg and stockbrk, as shown in Figure 7-9 on page 113 and Figure 7-10 on page 114.

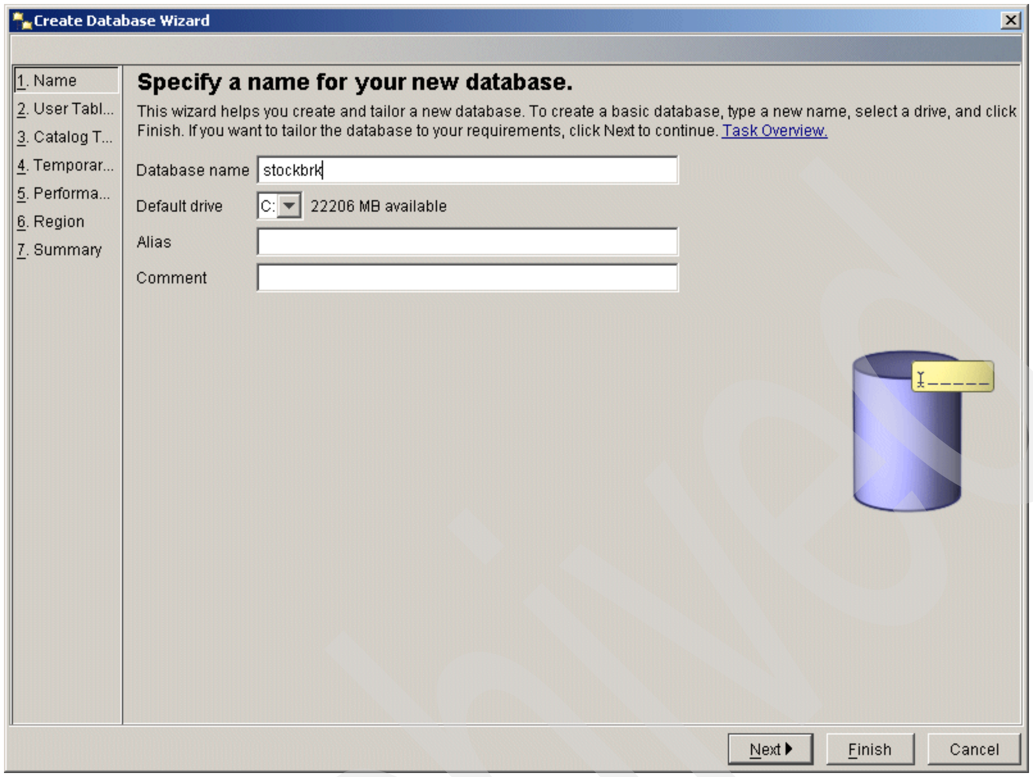

Create Database Wizard

Specify a name for your new database.

This wizard helps you create and tailor a new database. To create a basic database, type a new name, select a drive, and click Finish. If you want to tailor the database to your requirements, click Next to continue. Task Overview.

Database name: stockcfg
Default drive: C: 22206 MB available
Alias:
Comment:

Figure 7-9 Creating a new database


Create Database Wizard

1. Name
2. User Tabl...
3. Catalog T...
4. Temporar...
5. Performa...
6. Region
7. Summary

Specify a name for your new database.

This wizard helps you create and tailor a new database. To create a basic database, type a new name, select a drive, and click Finish. If you want to tailor the database to your requirements, click Next to continue. Task Overview.

Database name: stockbrk

Default drive: C: 22206 MB available

Alias:

Comment:

Figure 7-10 Creating a new database

Then create a new Data Source by opening **ODBC DataSource** and go to **System DSN.** Click the **Add** button and select a driver for the data source to be set up by selecting **IBM DB2 ODBC DRIVER**. Click **Finish** as shown in Figure 7-11 on page 115.

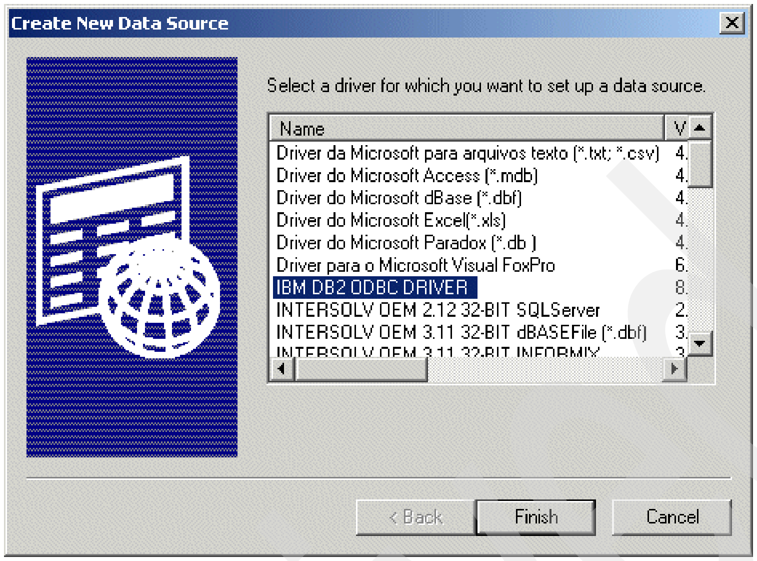
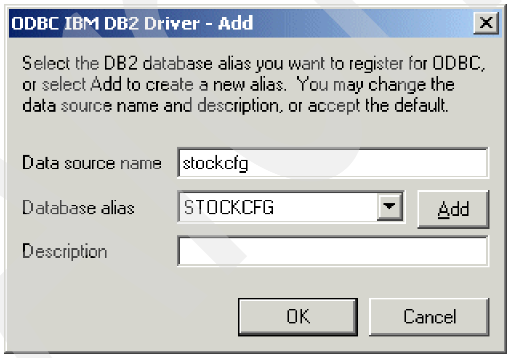

Create New Data Source
Select a driver for which you want to set up a data source.
Name
V
Driver da Microsoft para arquivos texto (*.txt; *.csv) 4.
Driver do Microsoft Access (*.mdb) 4.
Driver do Microsoft dBase (*.dbf) 4.
Driver do Microsoft Excel(*.xls) 4.
Driver do Microsoft Paradox (*.db) 4.
Driver para o Microsoft Visual FoxPro 6.
IBM DB2 ODBC DRIVER 8.
INTERSOLV OEM 2.12 32-BIT SQLServer 2.
INTERSOLV OEM 3.11 32-BIT dBASEFile (*.dbf) 3.
< Back
Finish
Cancel

Figure 7-11 Creating a new data source

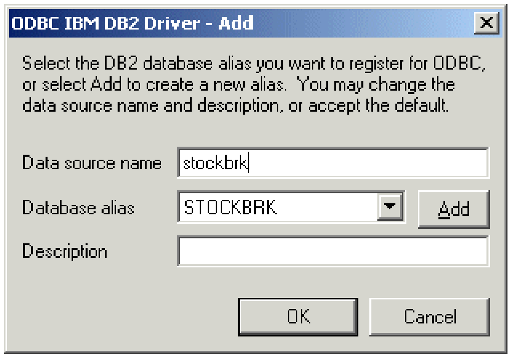
A dialogue box is then presented with DataSourceName and DataBase Alias and the Description. Give the data source name (for the Configuration Manager) `stockcfg` by setting the Database Alias to `stockcfg`. Give the data source name (for the Broker) `stockbrk` by setting the Database Alias to `stockbrk`, as shown in Figure 7-12 and Figure 7-13 on page 116.


ODBC IBM DB2 Driver - Add

Select the DB2 database alias you want to register for ODBC, or select Add to create a new alias. You may change the data source name and description, or accept the default.

Data source name	stockcfg
Database alias	STOCKCFG
Description	

Figure 7-12 Creating a data source for Configuration Manager


ODBC IBM DB2 Driver - Add

Select the DB2 database alias you want to register for ODBC, or select Add to create a new alias. You may change the data source name and description, or accept the default.

Data source name: stockbrk

Database alias: STOCKBRK

Description:

Figure 7-13 Creating a data source for the broker

Note: Database name and data source name must be the same.

Next, open the command prompt and create the Configuration Manager and the Broker.

7.2.1 Creating a Configuration Manager on Windows

A Configuration Manager can only be created on the Windows platforms. The Configuration Manager must be created on the system where the Configuration Manager component is installed, using the **`mqsicreateconfigmgr`** command. The parameters on this command provide the Configuration Manager with all the additional information it requires to be ready for action as soon as it is started, including the identifiers for the configuration repository. Before starting it, the following tasks must be completed:

- Ensure that the user ID has the correct authorizations to perform the task.
- The configuration repository database must be created and configured.

To create a Configuration Manager

Open a command prompt and enter the following command to create the Configuration Manager:

```
mqsicreateconfigmgr -i userid -a pwd -q stock_qm  -n stockcfg -u dbuid -p dbpwd
```

Note: Replace the appropriate values if using different names or values for any parameter on this command.

Where:

- *userid* is the service user ID used to run the Configuration Manager.
- *pwd* is the password for the service user ID.
- *stock_qm* is the name WebSphere MQ queue manager required to host the Configuration Manager. This is created if it does not exist.
- *stockcfg* is the name of the configuration repository database.
- *dbuserid* is the database user ID for the Configuration Manager. This is the user ID that was used when the database instance was created.
- *dbpwd* is the database password for the Configuration Manager. This is the password that was used when the database instance was created.

On completion of this task:

- A Configuration Manager is created and its Windows service is added to the Services (viewable from the Control Panel).
- The WebSphere MQ queue manager called *stock_qm* is created and started.
- WebSphere MQ resources required by the Configuration Manager are defined on the queue manager. This includes the default dead letter queue (DLQ) which is automatically enabled by running the **mqsicreateconfigmgr** command.
- Authorizations required by the Configuration Manager to access the WebSphere MQ resources are set up.
- The database tables required by the Configuration Manager are created and set up in the configuration repository database.

7.2.2 Creating a broker on Windows

Open a command line prompt and enter the following command to create the broker as shown:

```
mqsicreatebroker stockbrk -i userid -a pwd -q stock_qm -n stockbrk -u dbuid -p
dbpwd
```

Note: Replace the appropriate values if using different names or values for any parameter on this command.

Where:

- *stockbrk* is the broker name.
- *userid* is the service user ID that is used to run the broker.
- *pwd* is the password for the service user ID.

- *stock_qm* is the name of the WebSphere MQ queue manager that the broker needs to use. This queue manager is created if it does not exist.
- *stockbrk* is the name of the broker database. The broker tables are created within this database.
- *dbuid* is the user ID that has read, write and create access permissions for the database. This is the user ID used to read and update the broker's persistent store.
- *dbpwd* is the password that is associated with the database user ID.

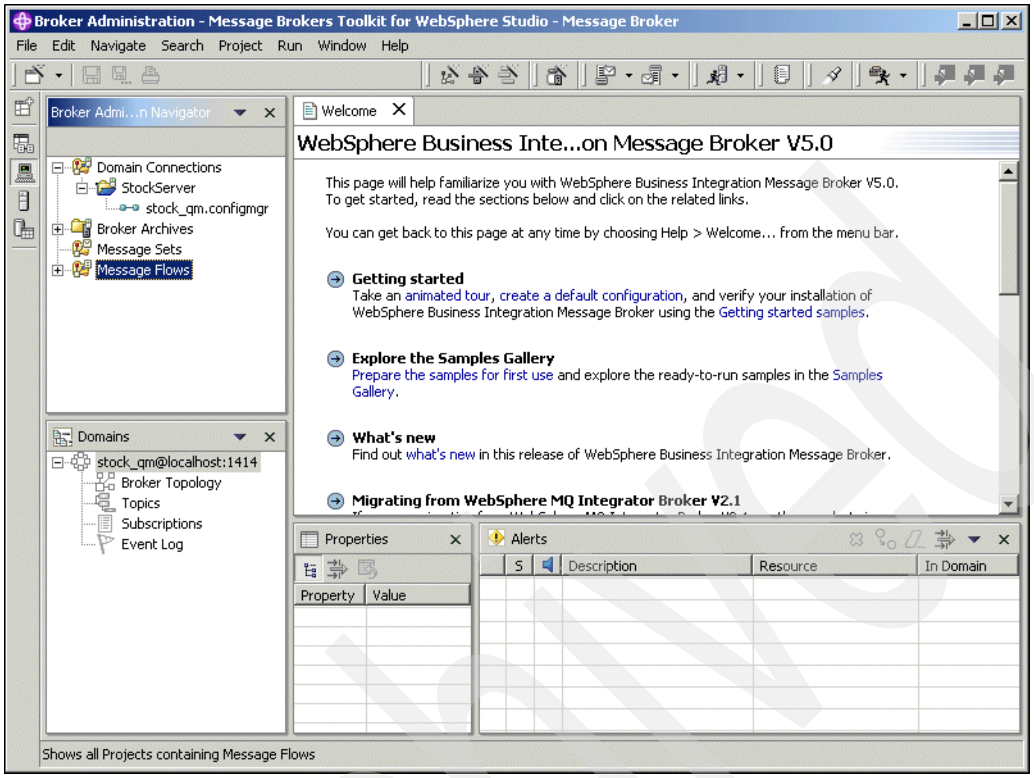
On completion of this task:

- A broker called *stockbrk* is created.
- A WebSphere MQ queue manager called *stock_qm* is created and started.
- The WebSphere MQ resources required by the broker are defined on the queue manager. This includes the default dead letter queue (DLQ), which is automatically enabled by running the `mqsicreatebroker` command.
- The database tables required by the broker were created and set up in the database.

7.3 Configuration of the Broker

Once a Configuration Manager and a broker have been defined, using the toolkit for development purposes is independent of these resources. Only when deploying and testing are these resources required to be active.

If the toolkit is not already active, begin by selecting it from the Windows 2000 desktop, **Start -> Programs -> IBM WebSphere Business Integration Message Brokers -> Message Brokers Toolkit**. The Toolkit displayed must be similar to Figure 7-14 on page 119.



WebSphere Business Inte...on Message Broker V5.0

This page will help familiarize you with WebSphere Business Integration Message Broker V5.0. To get started, read the sections below and click on the related links.

You can get back to this page at any time by choosing Help > Welcome... from the menu bar.

Getting started
Take an animated tour, create a default configuration, and verify your installation of WebSphere Business Integration Message Broker using the Getting started samples.

Explore the Samples Gallery
Prepare the samples for first use and explore the ready-to-run samples in the Samples Gallery.

What's new
Find out what's new in this release of WebSphere Business Integration Message Broker.

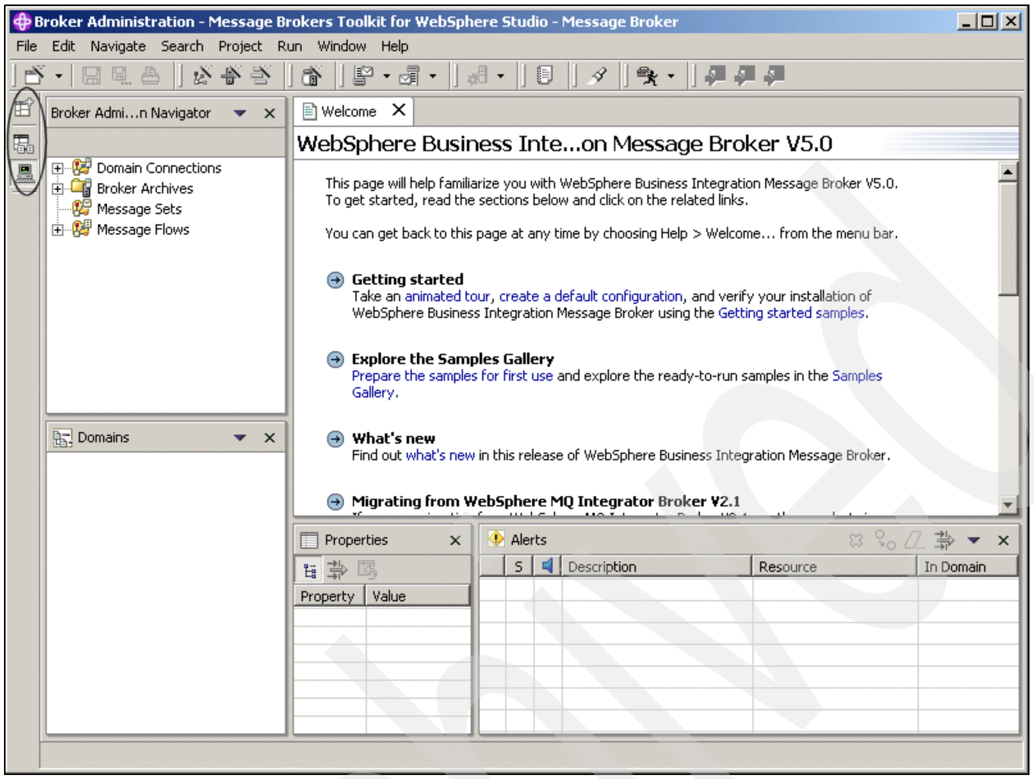
Migrating from WebSphere MQ Integrator Broker V2.1

Shows all Projects containing Message Flows

Figure 7-14 Message Broker Toolkit

7.3.1 Examining the Toolkit view

The toolkit is capable of displaying many perspectives. Icons which are in the upper left corner of the window indicate those available. At a minimum, two icons are visible. The first, Open a Perspective, displays a pull-down of current choices. The remaining icons refer to a specific choice. The view shown in Figure 7-15 on page 120 shows icons for the Broker Application Development Perspective and the Broker Domain Administration Perspective. A depressed icon (see insert in Figure 7-15 on page 120) indicates the current selection with the corresponding title displayed in the title bar.


Broker Administration - Message Brokers Toolkit for WebSphere Studio - Message Broker

File Edit Navigate Search Project Run Window Help

Broker Admi...n Navigator

- Domain Connections
- Broker Archives
- Message Sets
- Message Flows

Domains

Welcome

WebSphere Business Inte...on Message Broker V5.0

This page will help familiarize you with WebSphere Business Integration Message Broker V5.0.
To get started, read the sections below and click on the related links.

You can get back to this page at any time by choosing Help > Welcome... from the menu bar.

Getting started
Take an animated tour, create a default configuration, and verify your installation of WebSphere Business Integration Message Broker using the Getting started samples.

Explore the Samples Gallery
Prepare the samples for first use and explore the ready-to-run samples in the Samples Gallery.

What's new
Find out what's new in this release of WebSphere Business Integration Message Broker.

Migrating from WebSphere MQ Integrator Broker V2.1

Properties

Property	Value

Alerts

S	Description	Resource	In Domain

Figure 7-15 Perspective selection

7.3.2 Creating a ServerProject

Before creating a ServerProject, perform OpenServer Perspective by going to **Windows -> OpenPerspective -> Other**, as shown in Figure 7-16 on page 121. Click **Other**. A dialogue box opens, click **Server**, as shown in Figure 7-17 on page 122, then the Server Perspective Icon can be viewed in the left corner.

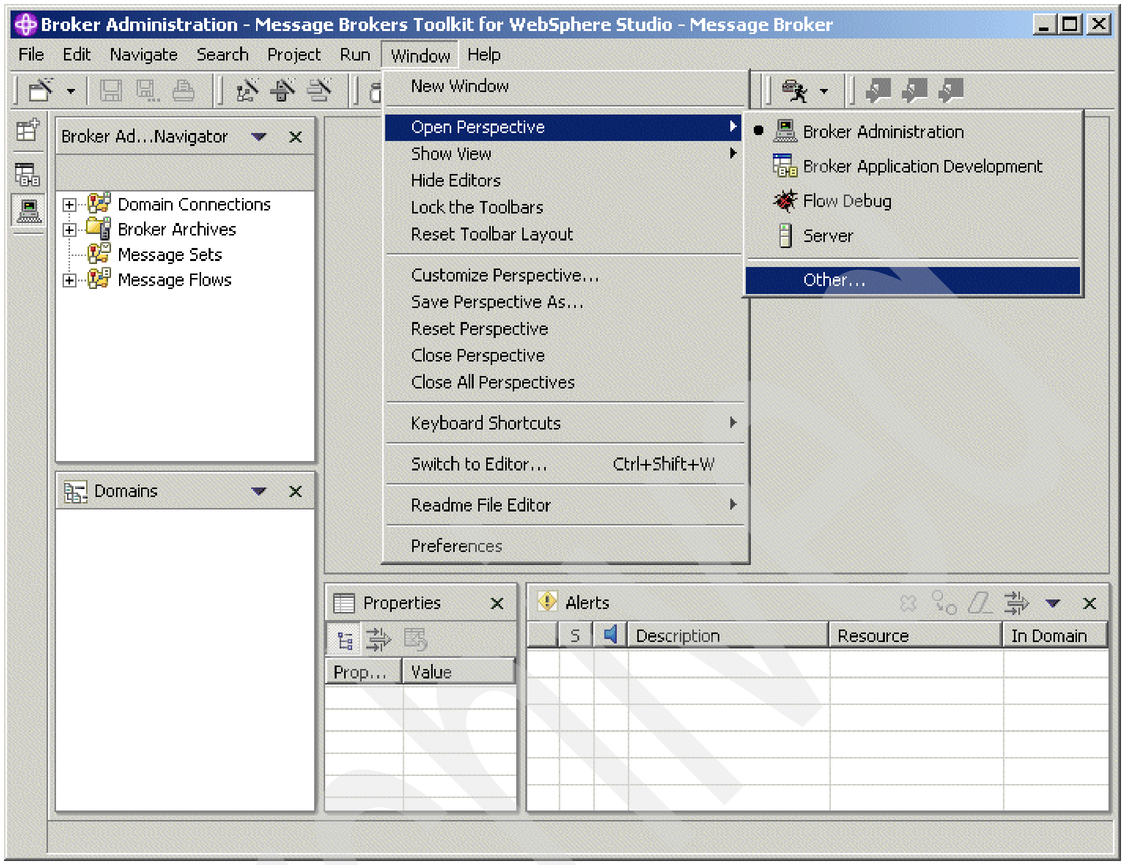


Figure 7-16 Server perspective selection

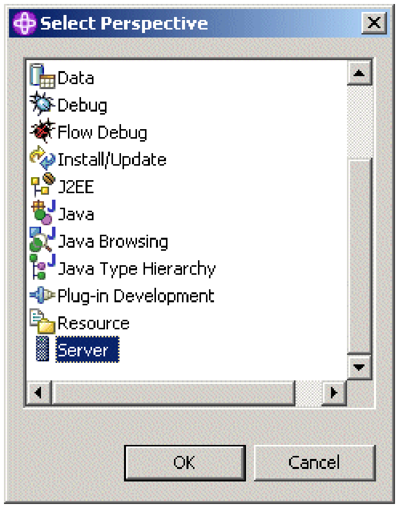


Figure 7-17 Select Server perspective

The Server perspective can be seen on the upper left corner of the window as a depressed icon.

A new server project can be created from the menu bar by selecting **File -> New -> Server Project** as shown in Figure 7-18 on page 123.

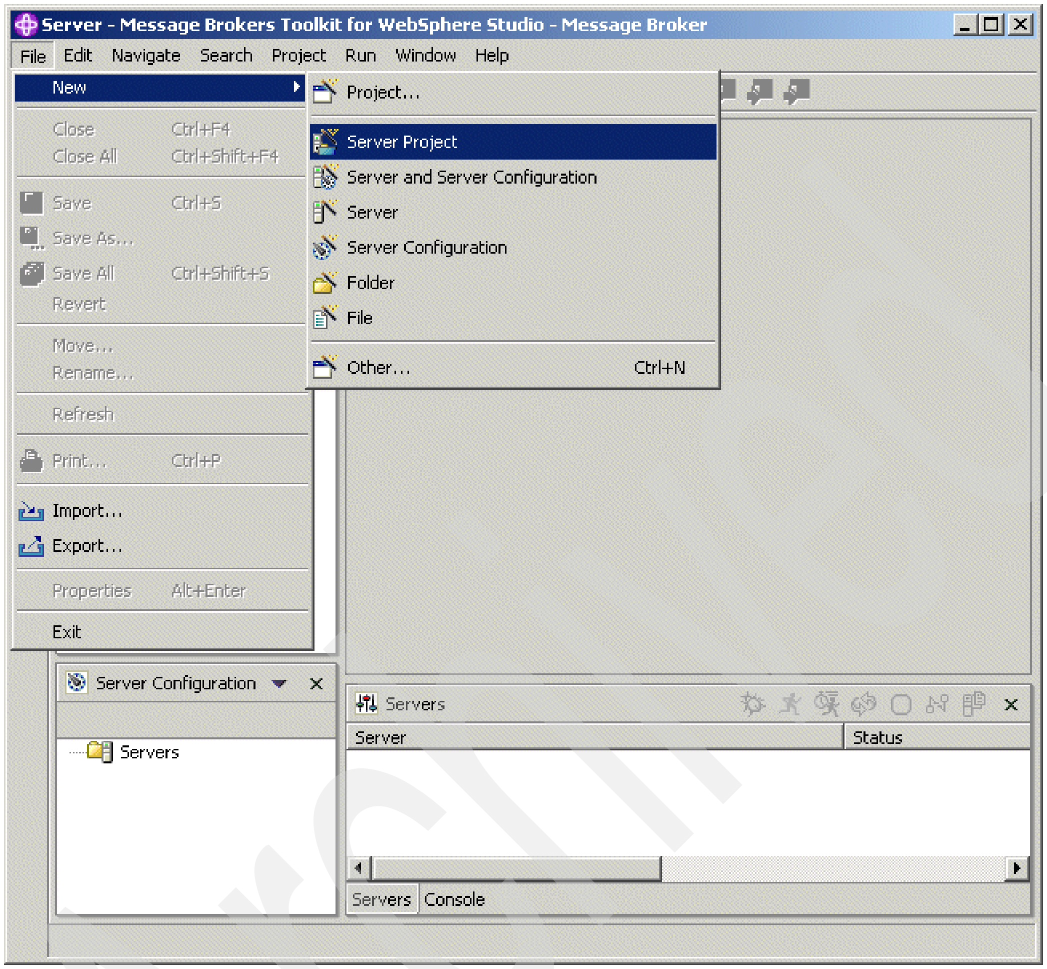
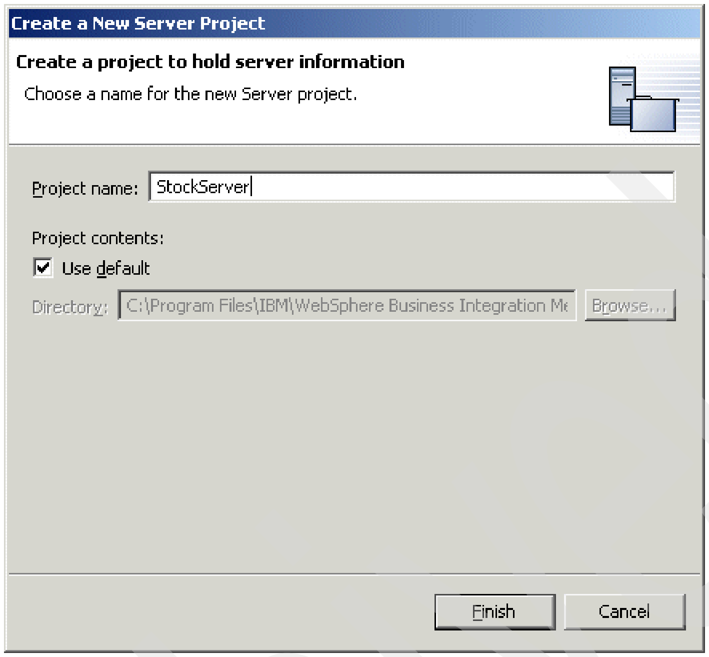


Figure 7-18 Creating new Server Project

Complete the dialogue box by providing the name for the Server Project and click **Finish** as shown in Figure 7-19 on page 124.


Create a New Server Project

Create a project to hold server information

Choose a name for the new Server project.

Project name: StockServer

Project contents:

☑ Use default

Directory: C:\Program Files\IBM\WebSphere Business Integration Me

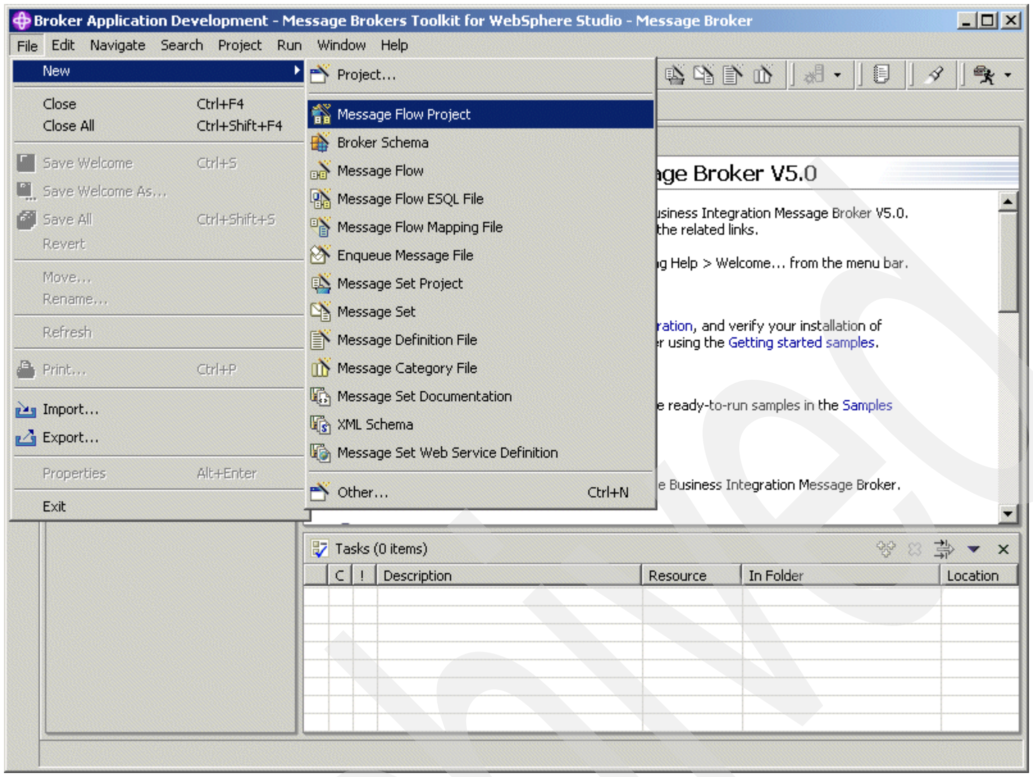
Figure 7-19 Server Project Name

Observe a new development in the Navigator view.

7.3.3 Creating a new message flow project

If needed, close all Editor views by either clicking the X box in the Editor tab or, from the menu bar, select **File -> Close All**.

A new message flow project can be created from the menu bar by selecting **File -> New -> Message Flow Project** as shown in Figure 7-20 on page 125. A dialogue box is then presented as shown in Figure 7-21 on page 126.



age Broker V5.0

usiness Integration Message Broker V5.0.
the related links.

g Help > Welcome... from the menu bar.

ration, and verify your installation of
r using the Getting started samples.

e ready-to-run samples in the Samples

e Business Integration Message Broker.

Figure 7-20 Create Message Flow Project Selection

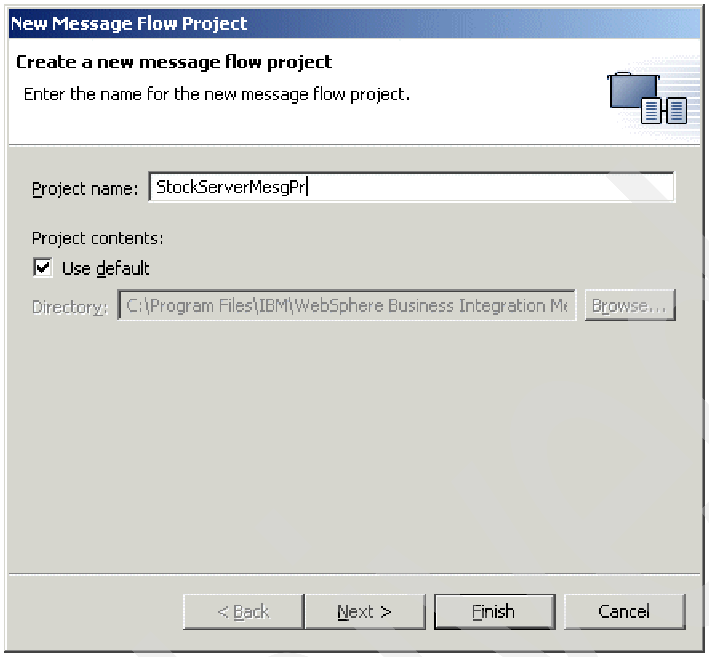

New Message Flow Project
Create a new message flow project
Enter the name for the new message flow project.
Project name: StockServerMesgPr
Project contents:
☑ Use default
Directory: C:\Program Files\IBM\WebSphere Business Integration Me
Browse...
< Back
Next >
Finish
Cancel

Figure 7-21 Name for Message Flow Project

Complete the dialogue box by providing a name for the Message Flow Project. For this example, the Project Name is StockServerMesgPr. Click **Finish** to complete this task.

Observe a new entry in the Resource Navigator view, that is, a file name that matches the newly created message flow project.

7.3.4 Creating a message flow

In the Resource Navigator view, right-click the entry for the message flow Project created earlier called **StockServerMesgPr**, and from the pull-down select **New -> Message Flow** as shown in Figure 7-22 on page 127.

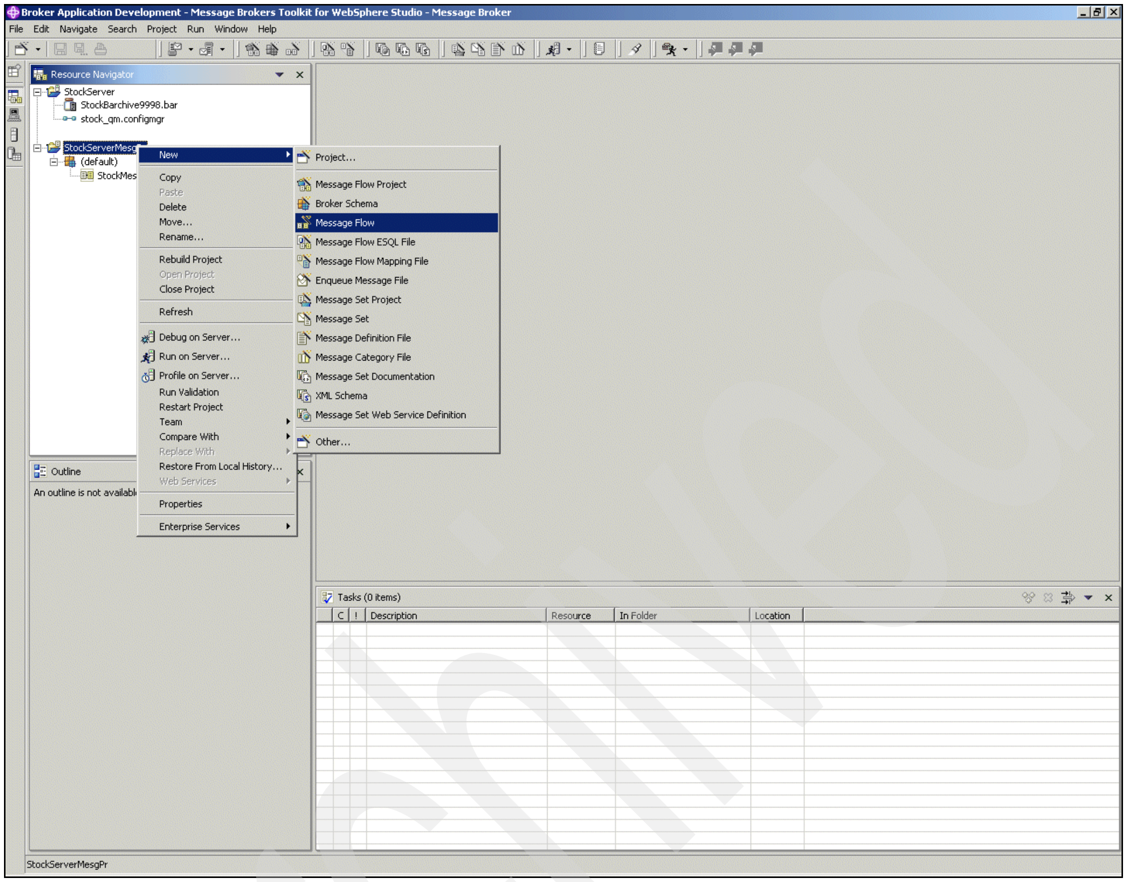
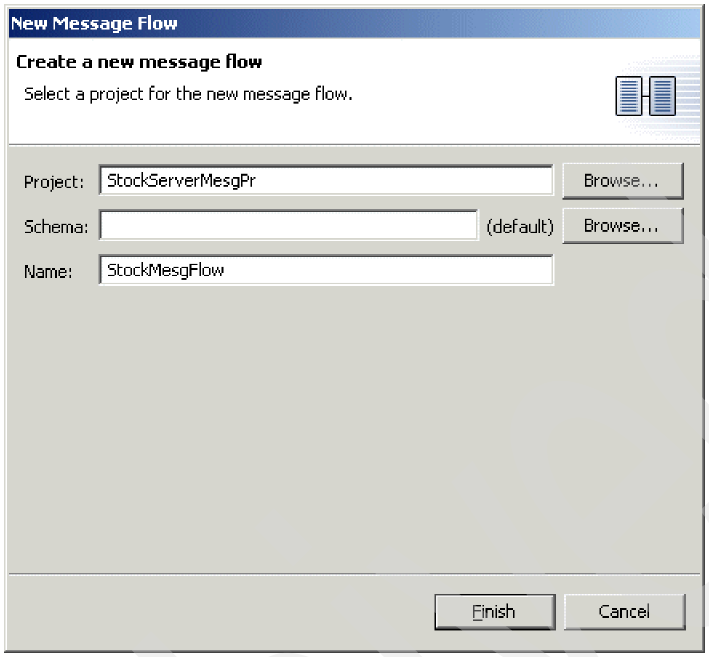


Figure 7-22 Creating message flow selection

Another dialogue box is presented to name the message flow. For example, the name StockMesgFlow shown in Figure 7-23 on page 128. For the time being, the Schema value is left unspecified.

Click **Finish.**



New Message Flow

Create a new message flow

Select a project for the new message flow.

Project: StockServerMesgPr

Schema: (default)

Name: StockMesgFlow

Figure 7-23 Creation of new message flow

Observe the addition of a new message flow file in the Resource Navigator view and the appearance of the Built-in nodes palette and empty "canvas" in the Editor view as shown in Figure 7-24 on page 129.

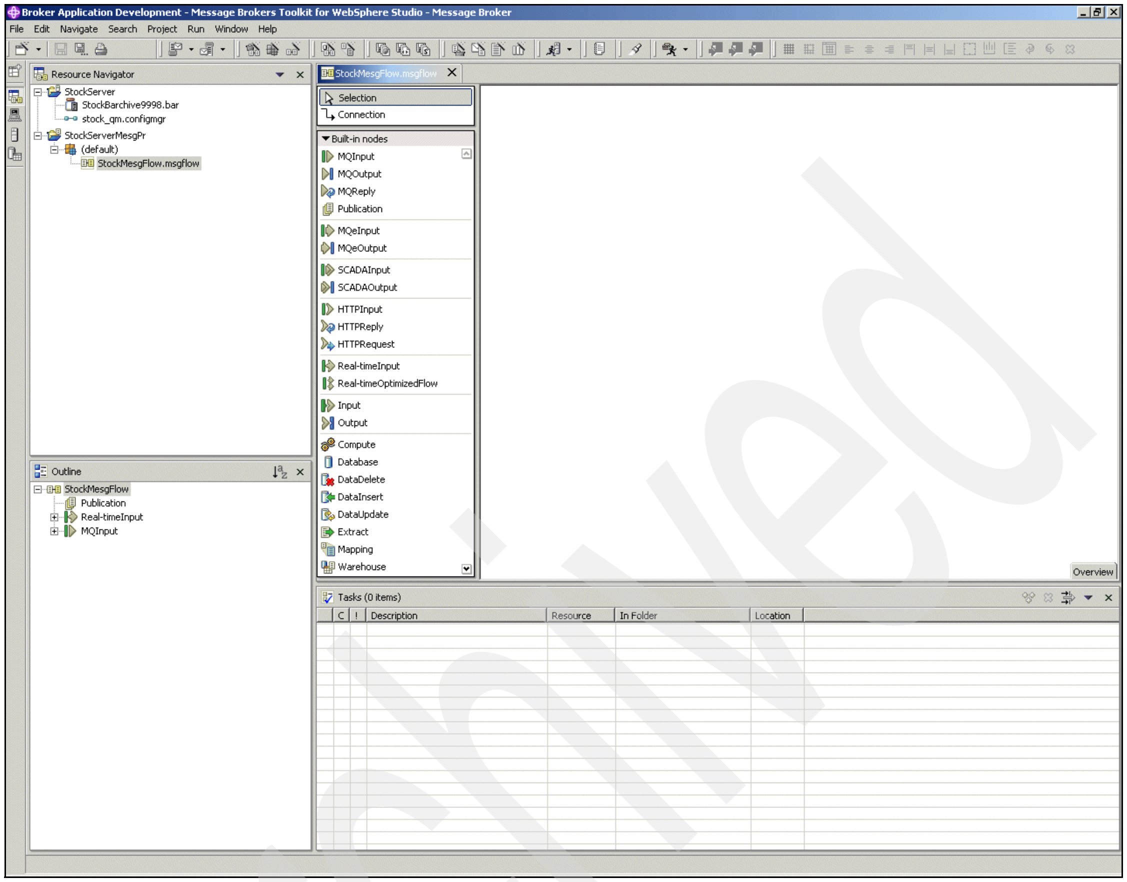


Figure 7-24 Message flow development view

7.3.5 Building the message flow

Populating the canvas is a process whereby instances of nodes that appear on the node palette are positioned on the canvas. They are selected by a click-and-click technique as described below:

1. Position the mouse pointer over the desired node instance in the node palette and select it.
2. Then position the mouse pointer in the desired location on the canvas where the node is to appear and click again.

An instance of the node appears on the canvas in approximately the selected location (see Figure Figure 7-26 on page 131, Figure 7-27 on page 132 and Figure 7-28 on page 133).

The selection of nodes from the palette is shown below in Figure 7-25.

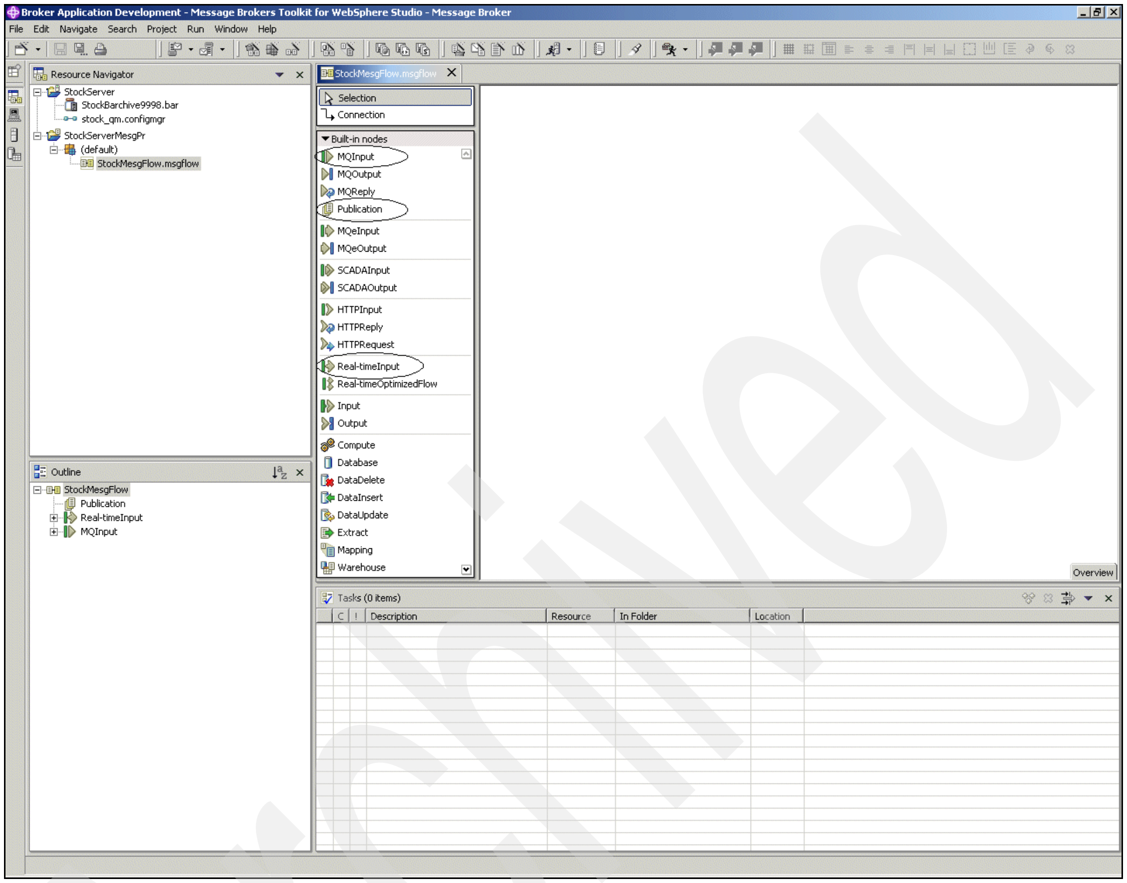


Figure 7-25 Node selection from the node palette

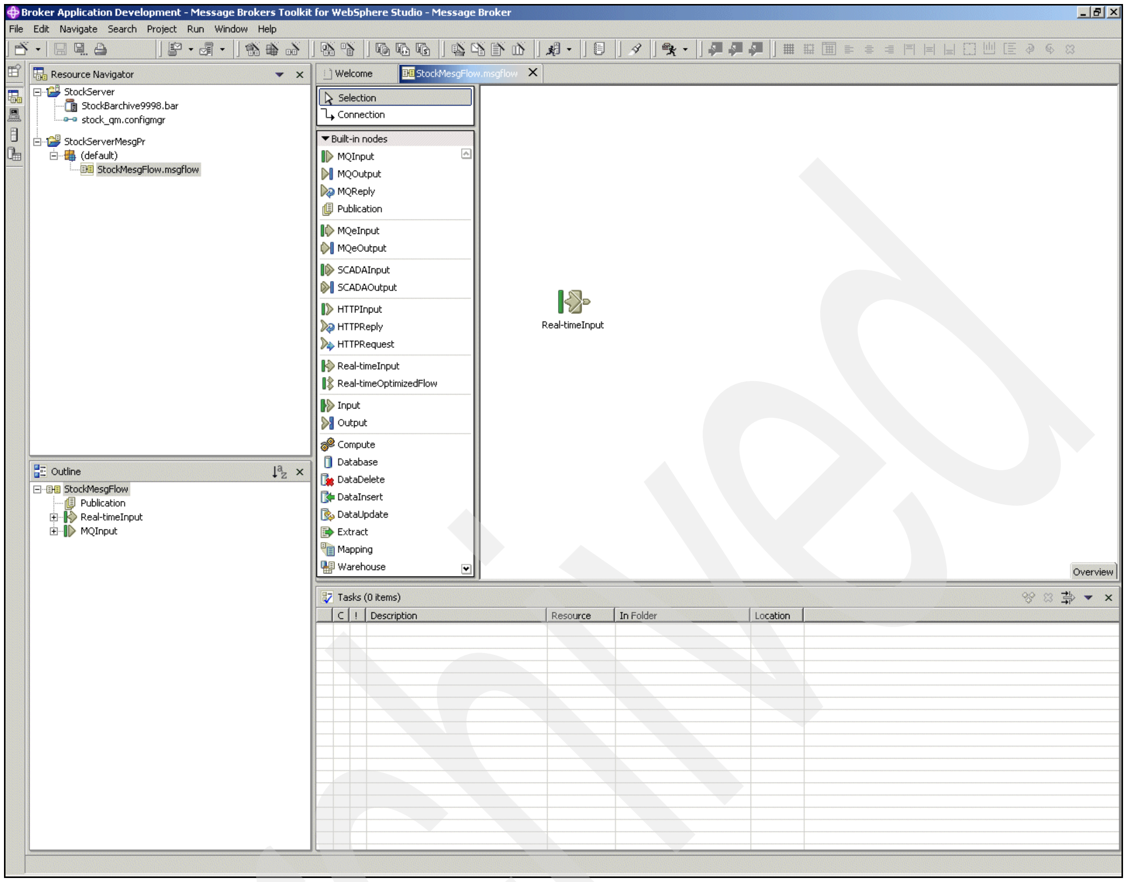


Figure 7-26 Positioning Real Time Input node

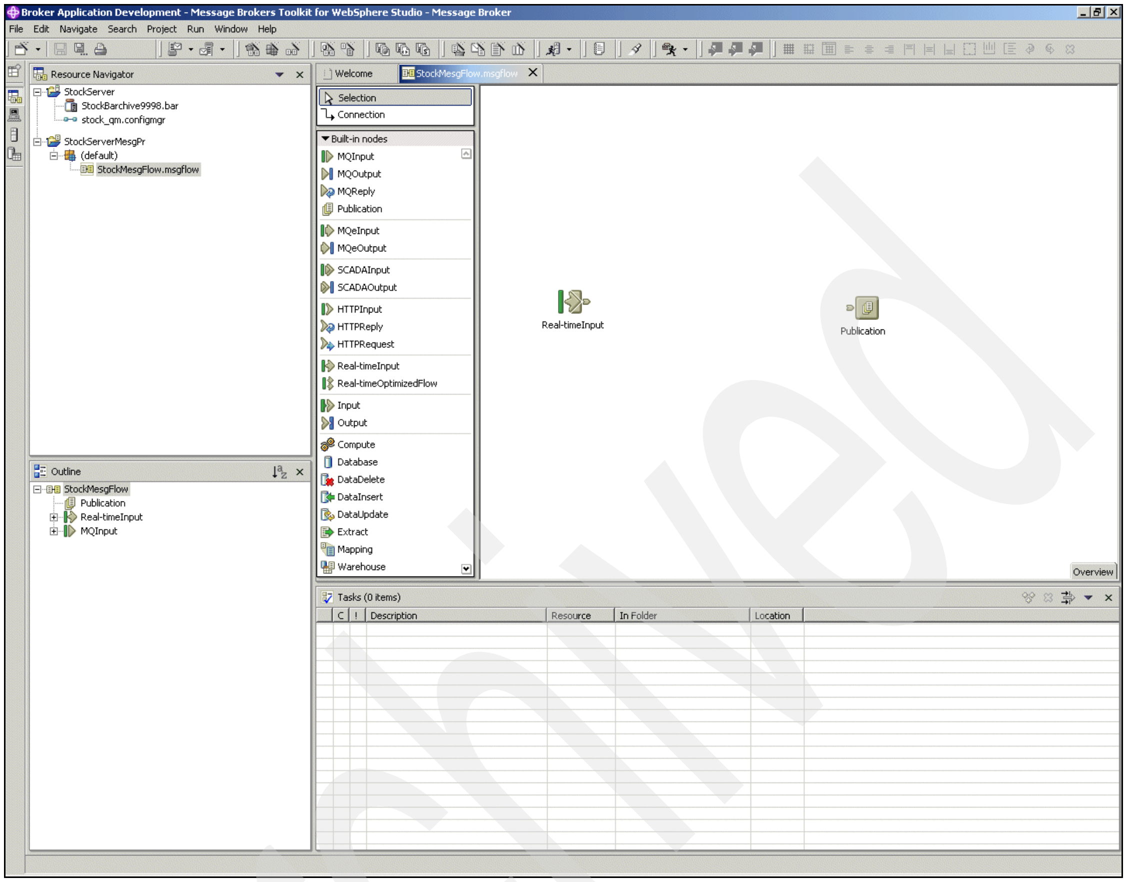


Figure 7-27 Positioning Publication node

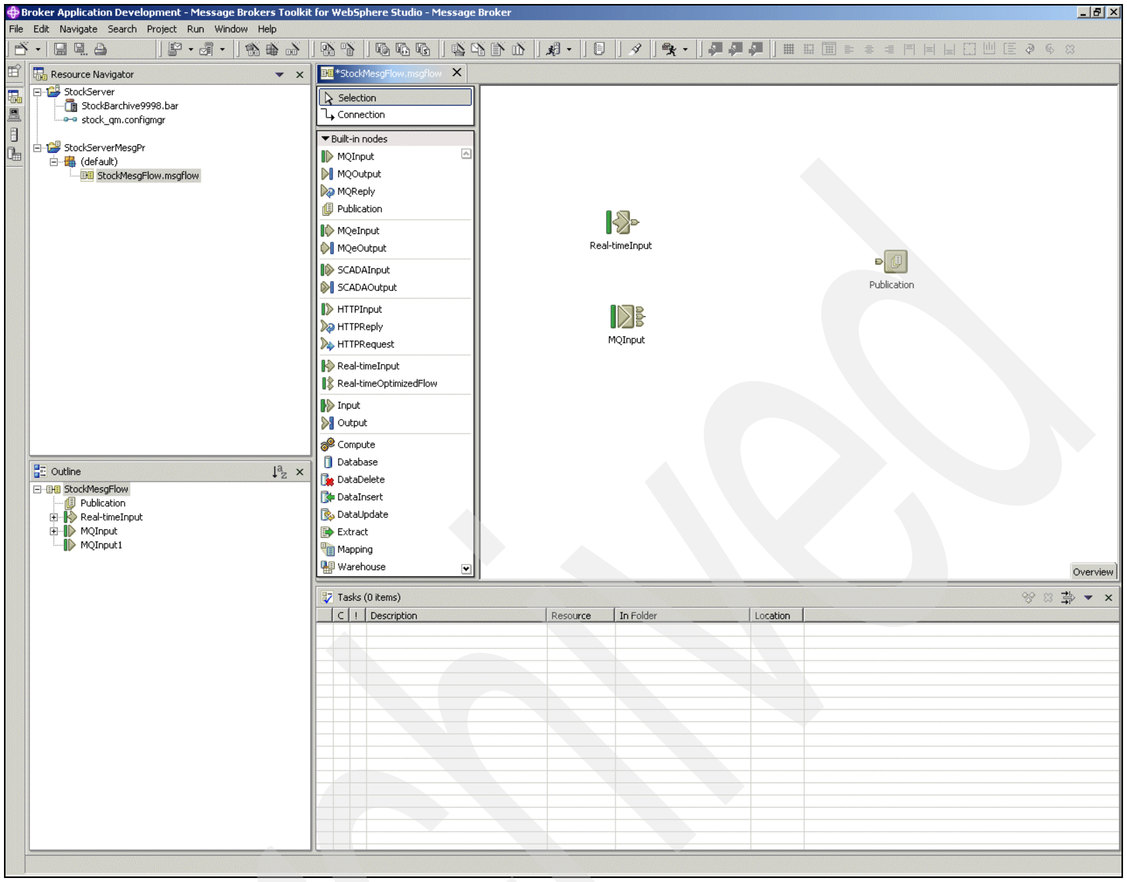


Figure 7-28 Positioning the MQInput node

7.3.6 Connecting the nodes

Nodes can be connected together in two ways:

The first technique is as follows:

1. Select **Connection** at the top of the node palette. When selected, use the left mouse button to select the source of the connection, that is, a node's output terminal, and the destination of the connection, that is, a node's input terminal.
2. Use the click-and-click technique to make the desired connections. When the connection choice is selected, other operations on the canvas are disabled.

The second technique is as follows:

1. Highlight the selection choice by right-clicking a node to produce a pull-down list of options.
2. Select **Create Connection** and a dialogue box appears that lists the node's output terminals.
3. Select an output source (which becomes the origin of the connection), and click **OK** to return to the editor view.
4. To finish the connection, position the mouse pointer over the desired destination input terminal and click again.

Figure 7-29, Figure 7-30 on page 135, Figure 7-31 on page 136, Figure 7-32 on page 137, and Figure 7-33 on page 138 illustrate Connection views.

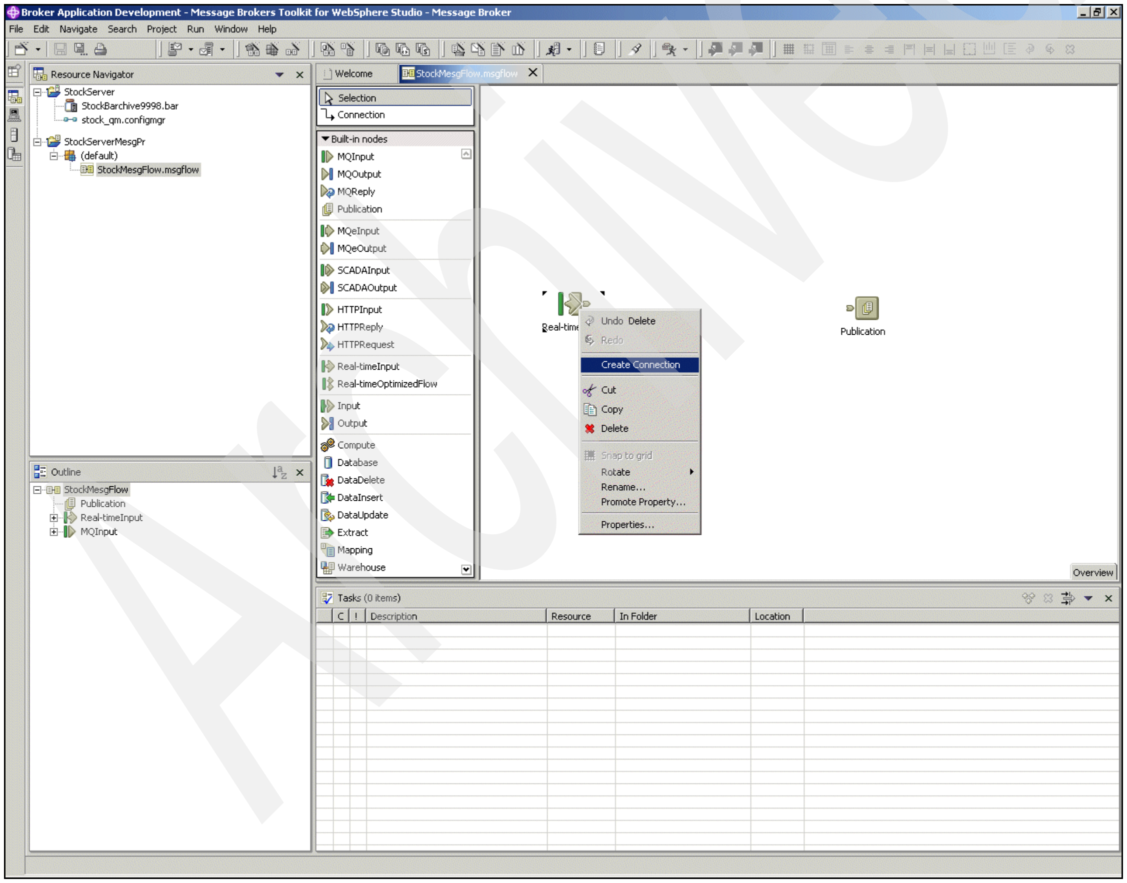


Figure 7-29 Connection selection pull-down for Real-time Input node

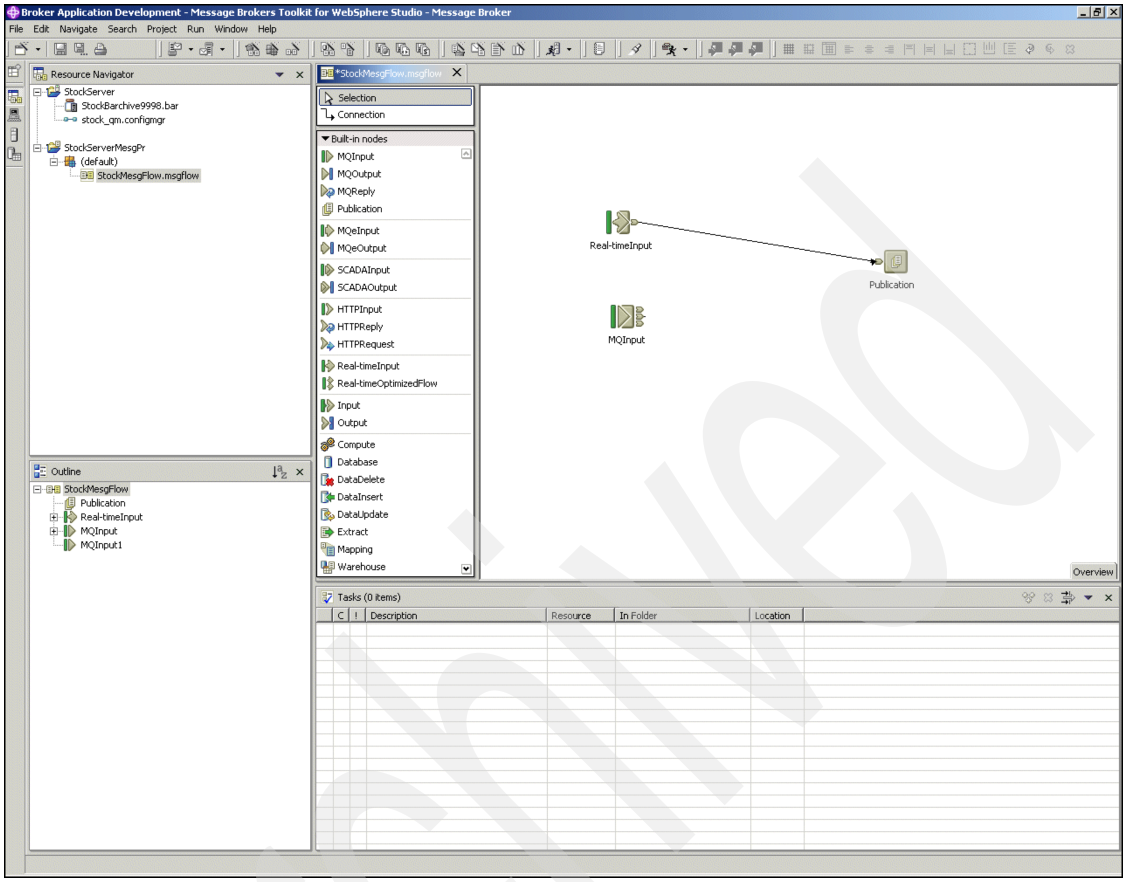


Figure 7-30 Connected nodes

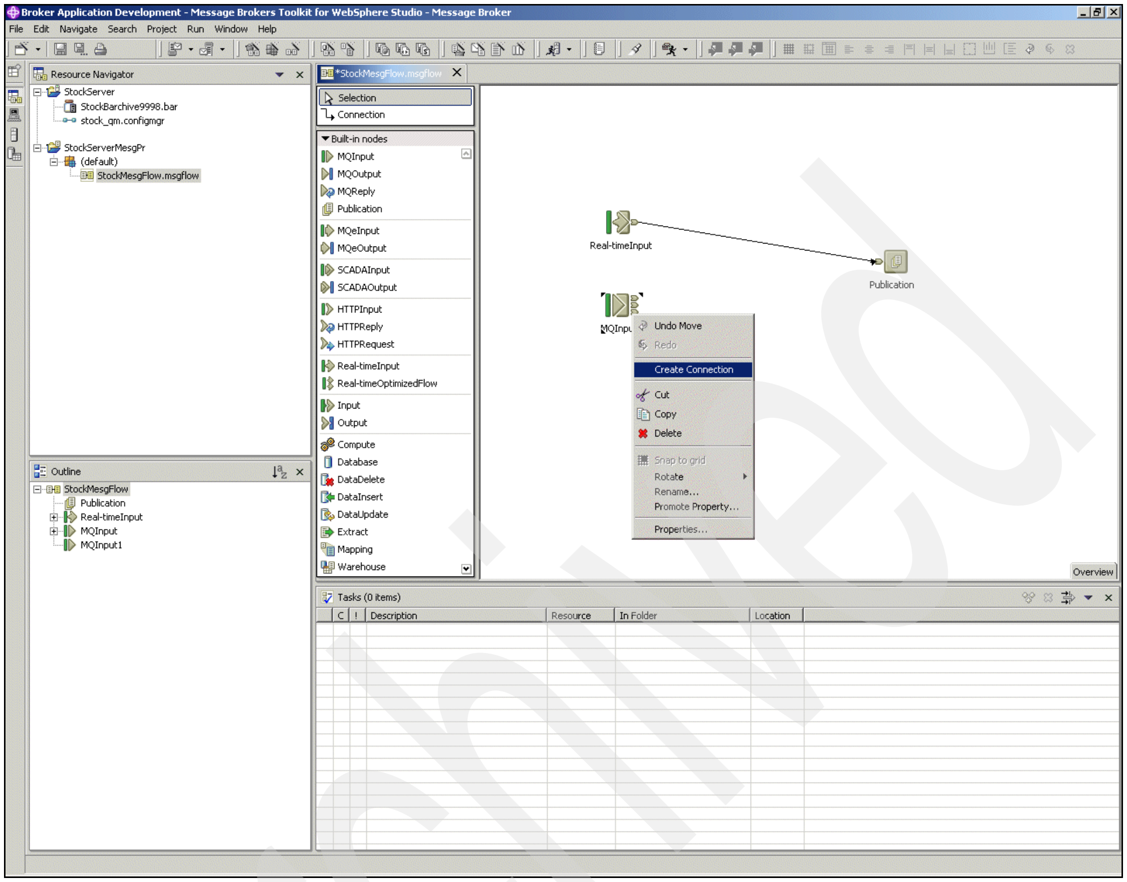


Figure 7-31   Connection selection of MQInput node

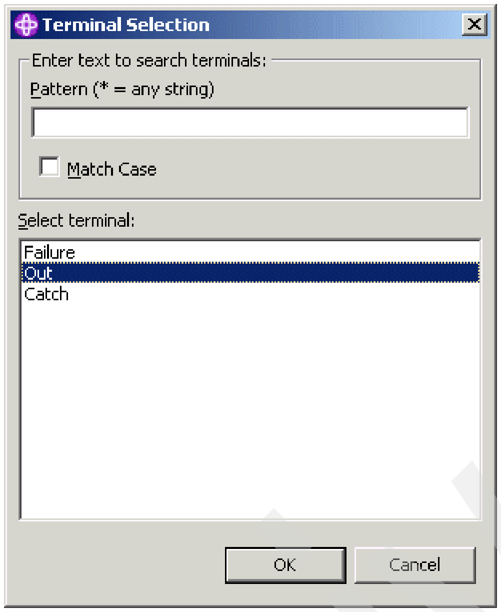


Figure 7-32 Connection selection destination

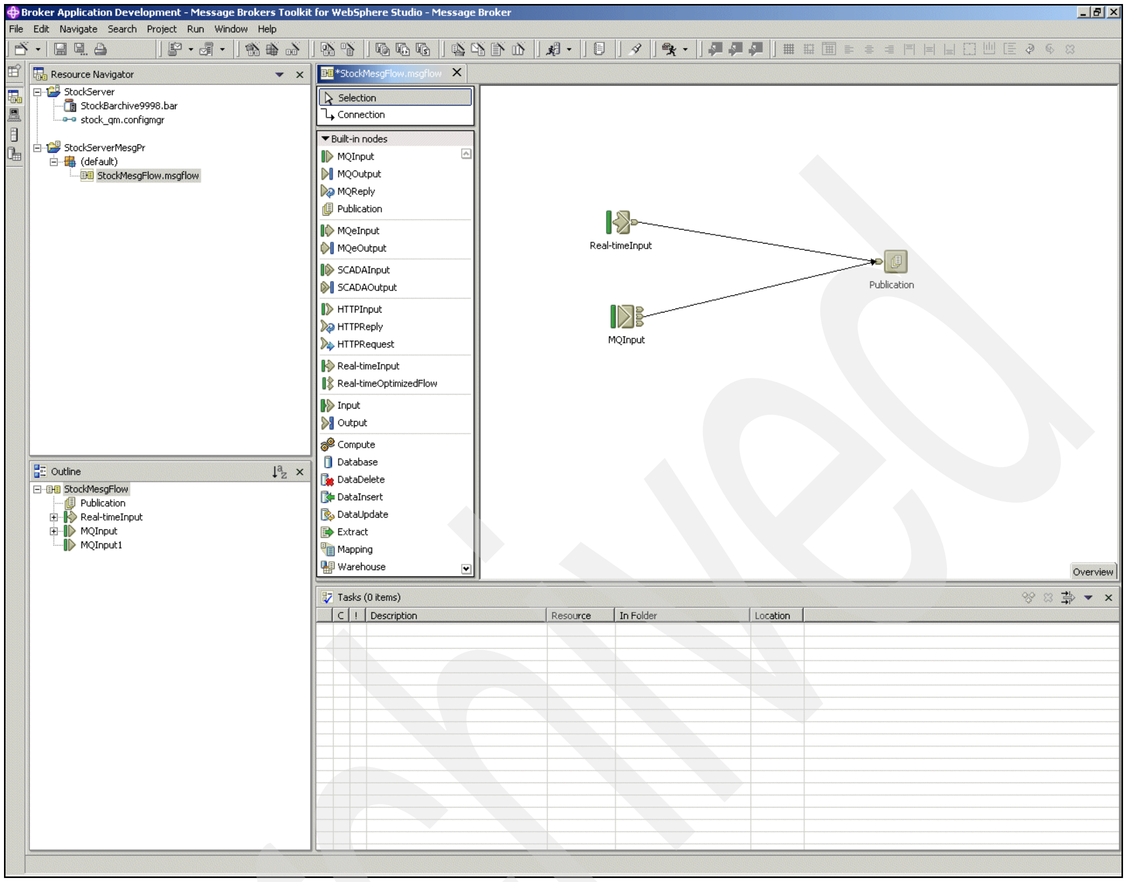


Figure 7-33 Connected nodes

Connections appear by default as lines between output and input terminals. With the Selection choice highlighted, a right-click any point in the white space of the canvas produces a pull-down list to select general appearance options. Observe the Outline view in the lower left of the Toolkit screen. As the connections are made on the canvas, the message flow associations are also represented in a tree diagram. As a message flow becomes more complex, perhaps with the addition of many nodes, it is likely to exceed the space available on the canvas. While scroll bars automatically appear when needed, an encapsulated view of the complete message flow can be obtained by selecting the Overview tab that appears in the lower right corner of the canvas.

7.3.7 Setting node properties

Right-click **Real-TimeInputNode** to produce a pull-down list with a Properties selection. See Figure 7-34. Enter the Port Number 9998 as shown in Figure 7-35 on page 140 and click **OK**.

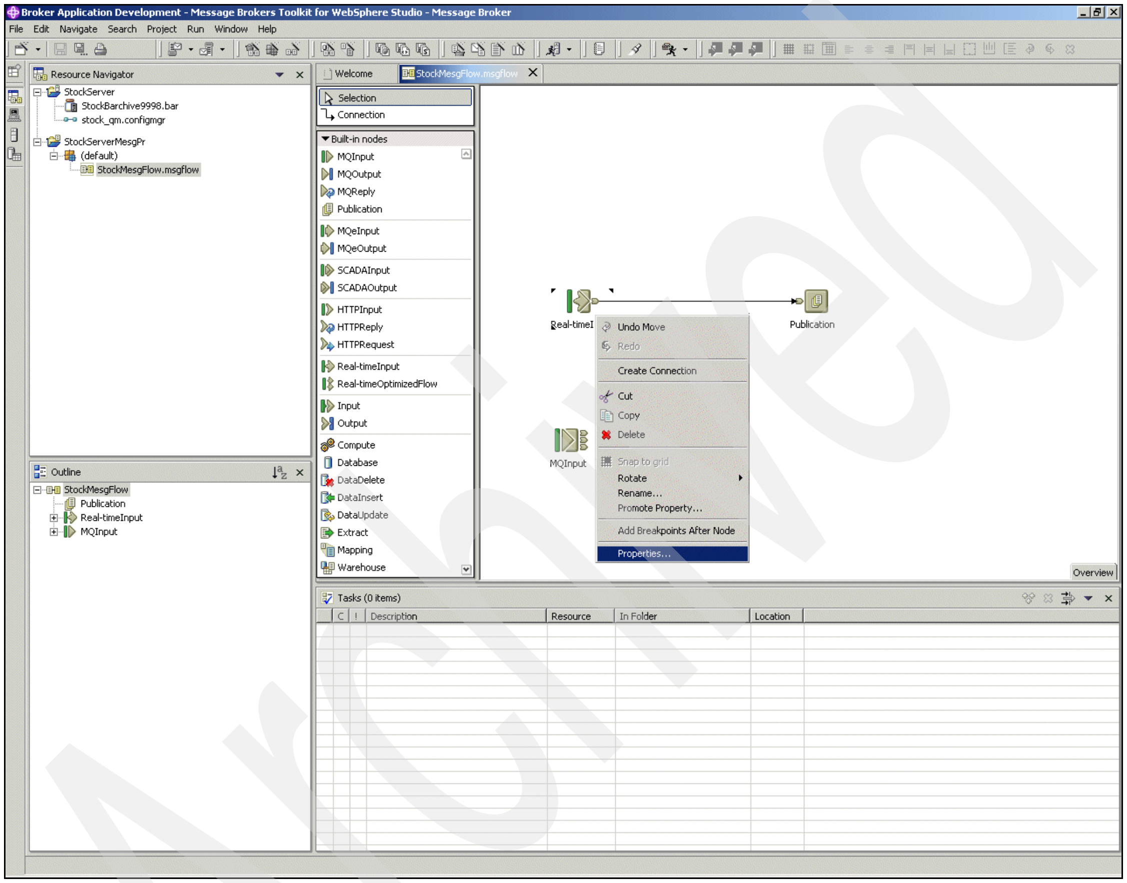
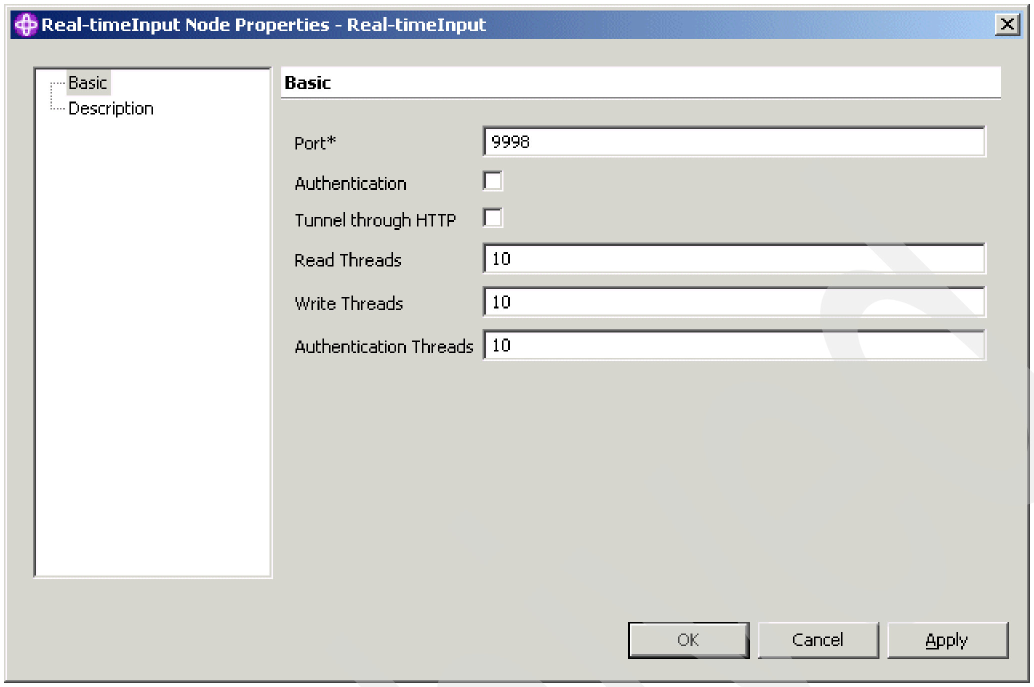


Figure 7-34 Selecting Real-time Input node properties


Real-timeInput Node Properties - Real-timeInput

Basic

Port* 9998

Authentication ☐

Tunnel through HTTP ☐

Read Threads 10

Write Threads 10

Authentication Threads 10

Figure 7-35 Real timeInput node properties

Right-click the **MQInputNode** as shown in Figure 7-36 on page 141, select the properties, and give the queue manager name as shown in Figure 7-37 on page 142.

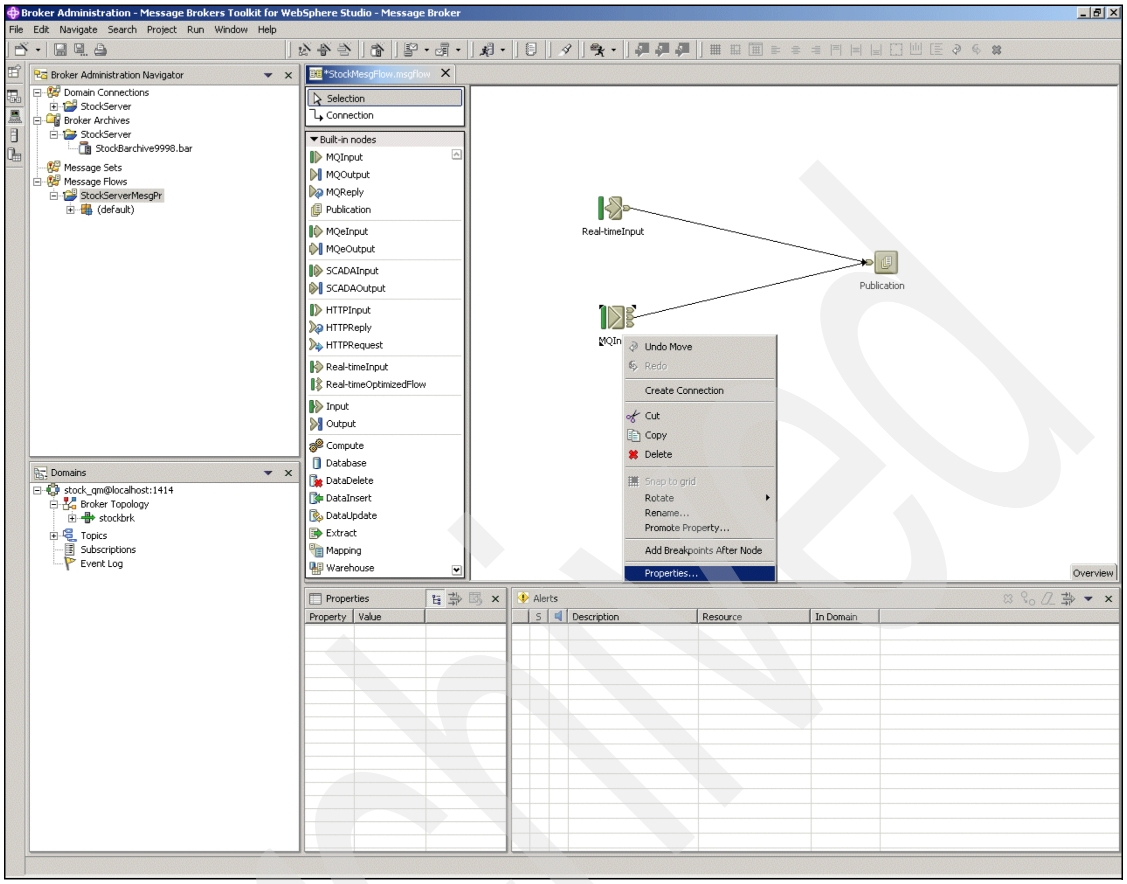
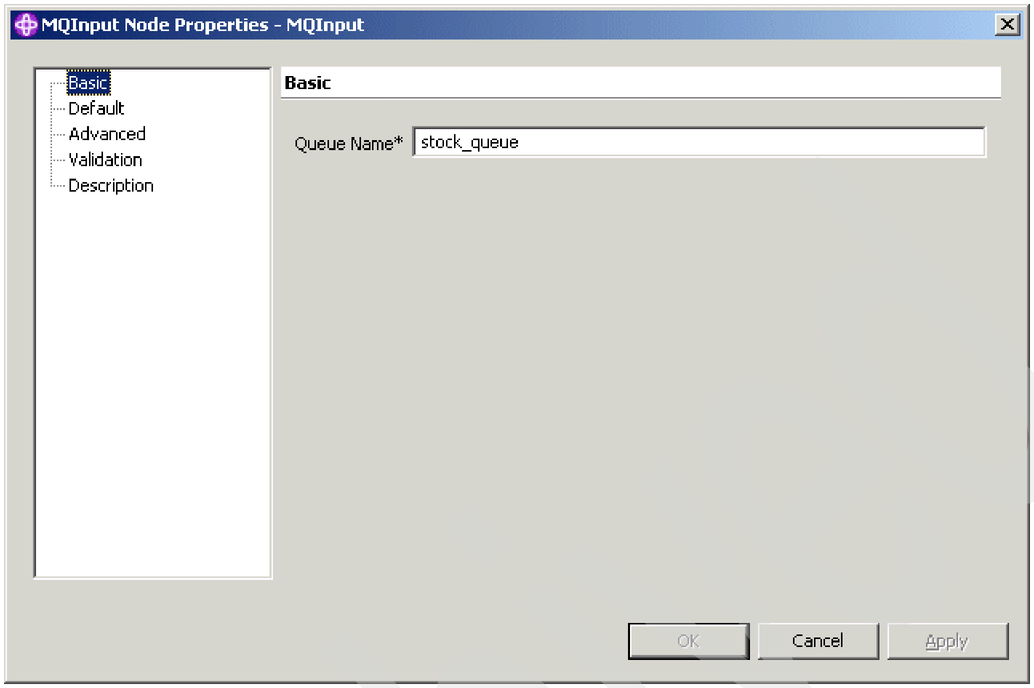


Figure 7-36 Selecting MQ Input node properties



Basic

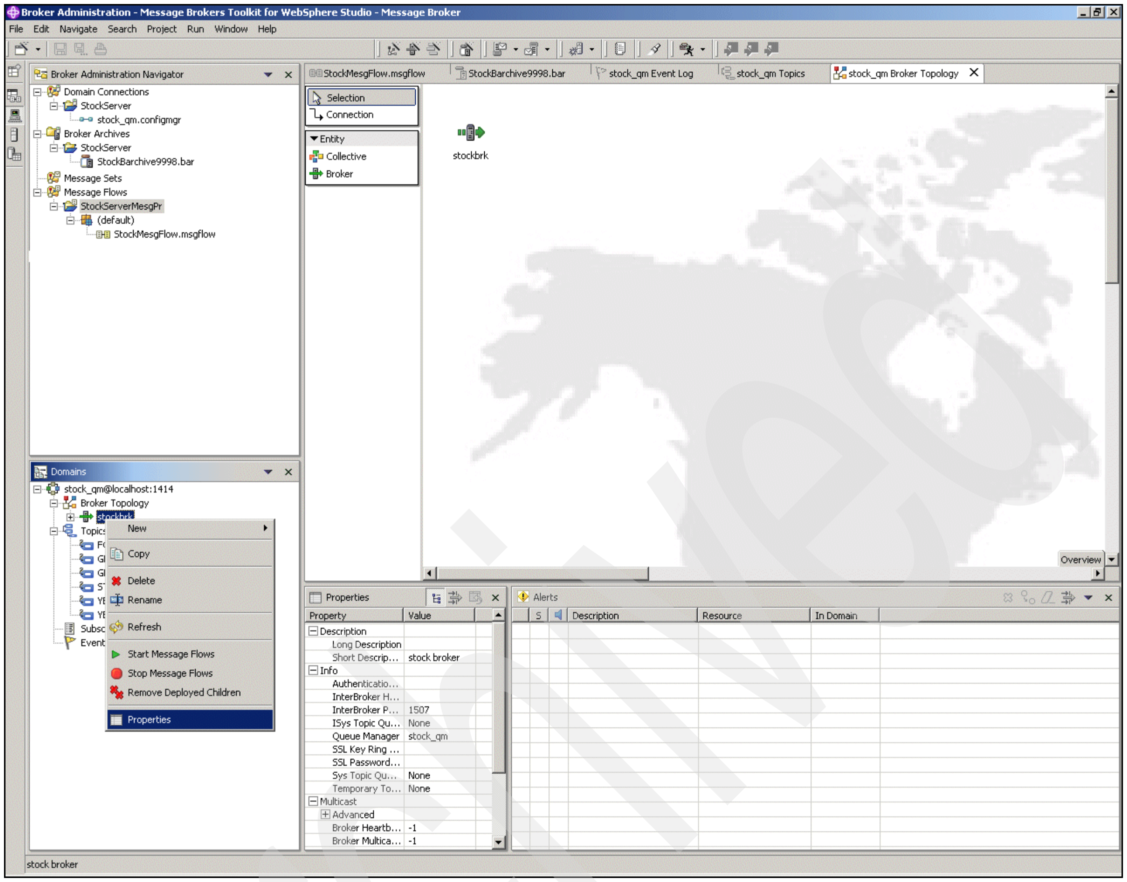
Queue Name* stock_queue

Figure 7-37 MQInput Node properties

7.3.8 Multicasting the brokers

Enabling multicast for a broker can be done in the following way.

Go to **Administration Perspective**. In the lower left corner right click the broker name **stockbrk** under broker topology in the Domain view, as shown in Figure 7-38 on page 143, go to properties.

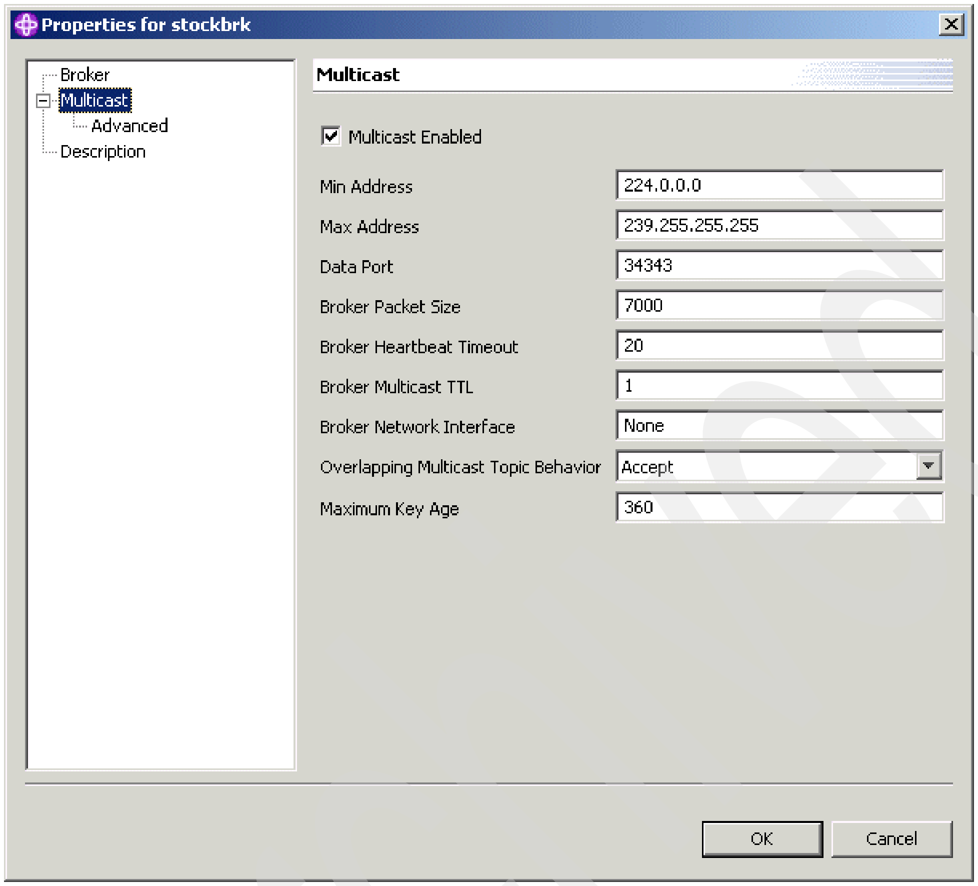


Properties

Property	Value
Description	
Long Description	
Short Descrip…	stock broker
Info	
Authenticatio…	
InterBroker H…	
InterBroker P…	1507
ISys Topic Qu…	None
Queue Manager	stock_qm
SSL Key Ring …	
SSL Password…	
Sys Topic Qu…	None
Temporary To…	None
Multicast	
Advanced	
Broker Heartb…	-1
Broker Multica…	-1

Figure 7-38 Multicast

A dialogue box appears. Click **Multicast**, click the checkbox for the **Multicast Enabled** property and click **OK**, as shown in Figure 7-39 on page 144.



[✓] Multicast Enabled

Min Address	224.0.0.0
Max Address	239.255.255.255
Data Port	34343
Broker Packet Size	7000
Broker Heartbeat Timeout	20
Broker Multicast TTL	1
Broker Network Interface	None
Overlapping Multicast Topic Behavior	Accept
Maximum Key Age	360

Figure 7-39 Multicast enabling

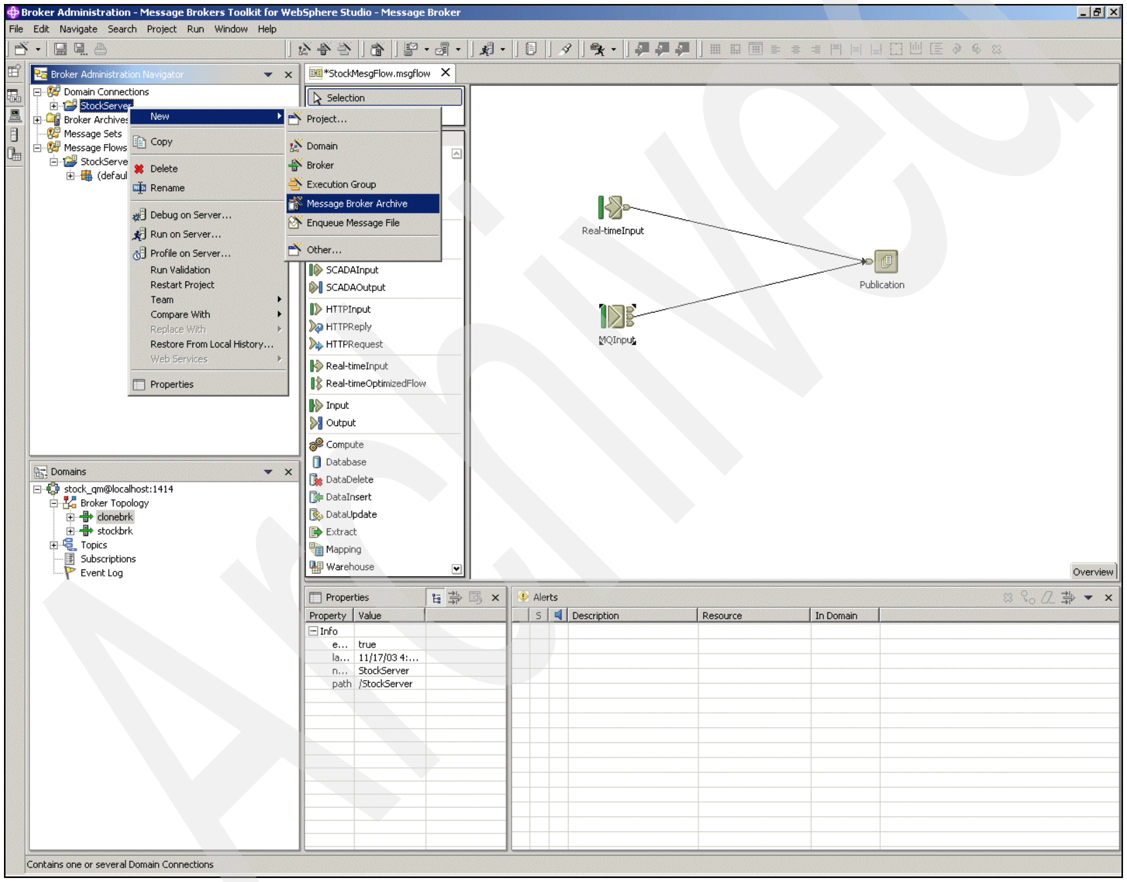
7.3.9 Creating the broker archive file

The broker archive file is the unit of deployment to a broker and contains compiled message flows and optional message set resources. It also contains a deployment descriptor that has information for a specific deployment. There needs to be a separate broker archive file for each deployment configuration that is used.

7.3.10 Creating a Message Broker Archive

Switch to the Broker Administration Perspective by using the toolbar icon or by using the menu bar **Window -> Open Perspective -> Broker Administration**. In the Broker Administration Navigator view, near the top left, there is a folder called Domain Connections. Within this folder is the StockServer that was previously created. This is the server project that contains details of the broker and execution group to which the bar file is deployed. Select the **StockServer** and right-click **New -> Message Broker Archive** as shown in Figure 7-40.

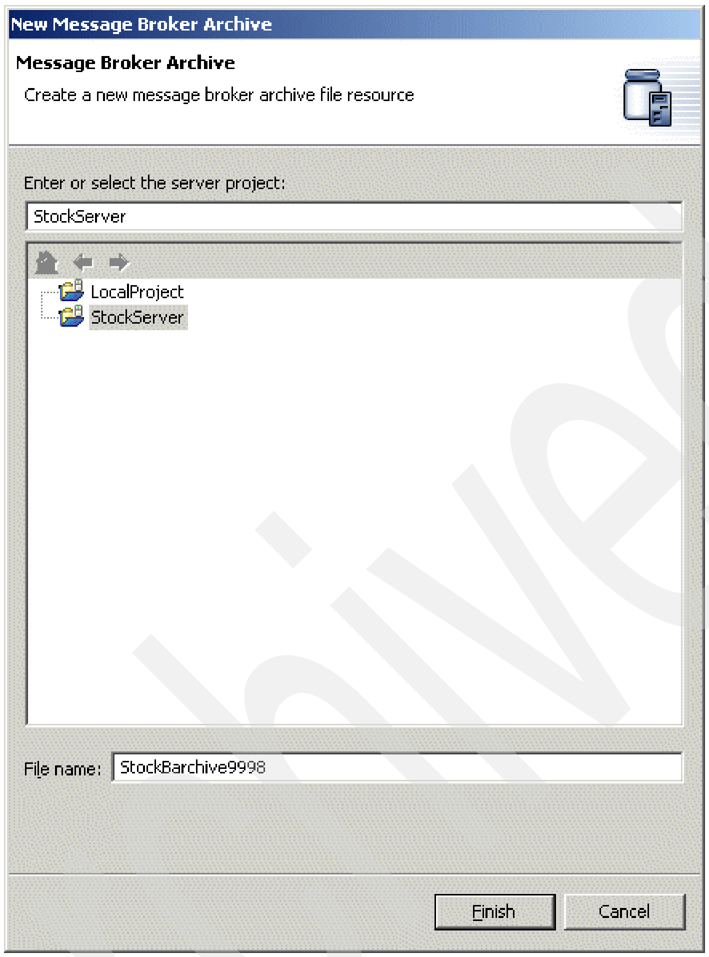
Note: An alternative for creating a new broker archive is to use the menu bar **File -> New -> Message Broker Archive**, but this does not have StockServer project preselected in the next dialogue box.



Property	Value
Info	
e...	true
la...	11/17/03 4:...
n...	StockServer
path	/StockServer

Figure 7-40   Creating a new broker archive within a project

A wizard starts and displays a dialogue box that requires detail about the brokerarchive file name and the associated server project. Select the **StockServer project** and enter `StockBarchive9998` into the file name field and click **Finish**, as shown in Figure 7-41.


New Message Broker Archive

Message Broker Archive

Create a new message broker archive file resource

Enter or select the server project:

StockServer

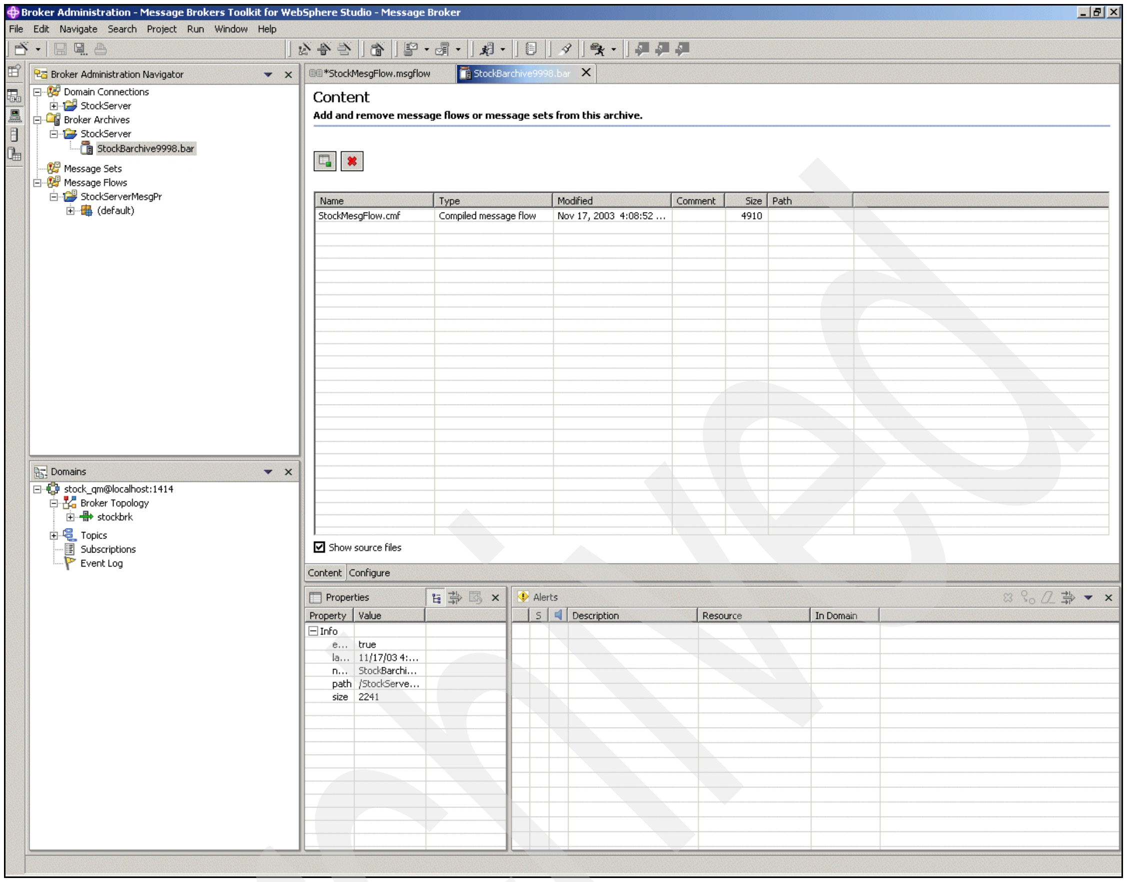
LocalProject

StockServer

File name: StockBarchive9998

Figure 7-41 Message Broker Archive Wizard

After the new broker archive is created, a new file StockBarchive9998.bar is created in the StockServer within the Broker Archives folder, and a new broker archive editor is opened for this file, as shown in Figure 7-42 on page 147.



Name	Type	Modified	Comment	Size	Path
StockMesgFlow.cmf	Compiled message flow	Nov 17, 2003 4:08:52 ...		4910	

Property	Value
Info	
e...	true
la...	11/17/03 4:...
n...	StockBarchi...
path	/StockServe...
size	2241

Figure 7-42 New broker archive editor

The broker archive editor has two tabs called Content and Configure. The Content view is used to add and delete message flows and message sets, and the Configure view is used to modify the deployment descriptor. The editor is initially in the Content view.

To add a new message file into the bar file, click the Add icon near the top of the editor. This displays an Add to Broker Archive dialogue box where all flows associated with StockMesgPr are chosen by selecting the check box next to the project and clicking **OK**, as shown in Figure 7-43 on page 148.

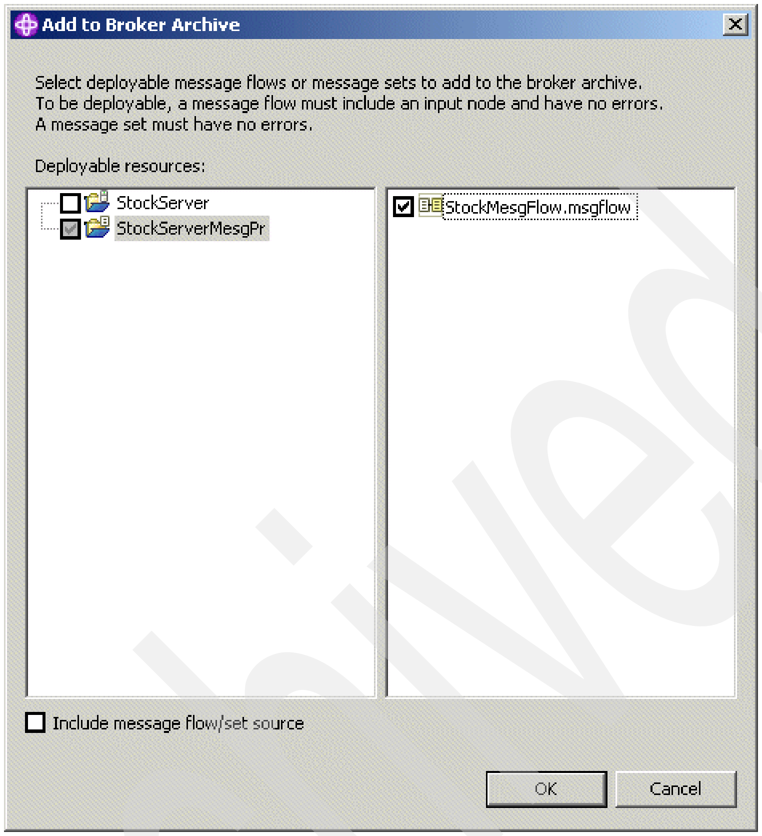
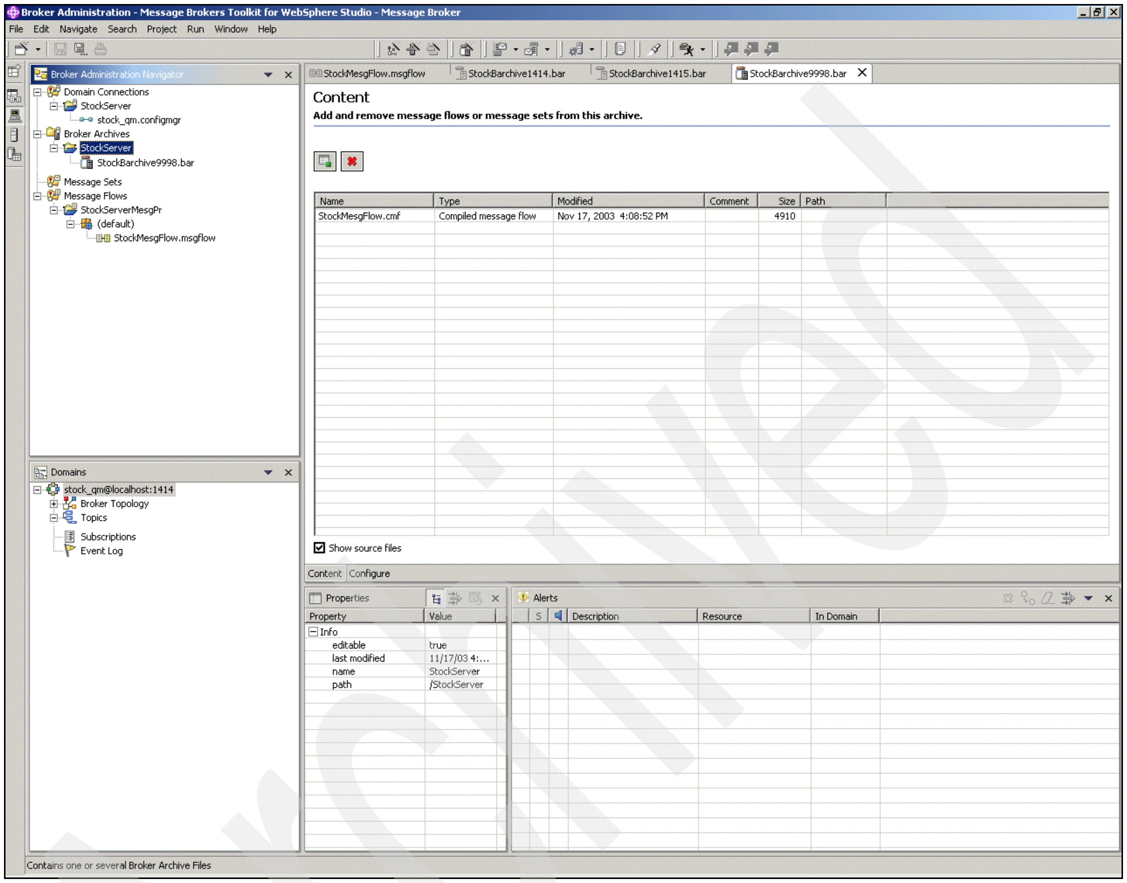

Add to Broker Archive
Select deployable message flows or message sets to add to the broker archive.
To be deployable, a message flow must include an input node and have no errors.
A message set must have no errors.
Deployable resources:
☐ StockServer
☑ StockServerMesgPr
☑ StockMesgFlow.msgflow
☐ Include message flow/set source
OK
Cancel

Figure 7-43 Add to Broker Archive dialog box

Tip: To choose only specific flows within a project, the project must be selected on the left pane and the individual flows checked in the right pane of this dialogue box.

An information dialogue box with a progress bar is shown as the message flow is compiled and added to the bar file. On successful completion, `Operation completed` is displayed. There is a Details>> button that can be used to see further information. If the message flow was not in a deployable state (such as a missing input node), this dialogue box shows where to locate the problem.

After the message flow has been successfully added to the bar file, it shows in the list as `StockMesgFlow.cmf;` with type as compiled message flow and the last modified time and size, as shown in Figure 7-44 on page 149. At this point save the bar file by using Ctrl+S so that changes are saved prior to deployment. The bar file must be saved.


Broker Administration - Message Brokers Toolkit for WebSphere Studio - Message Broker

Broker Administration Navigator
- Domain Connections
  - StockServer
    - stock_qm.configmgr
- Broker Archives
  - StockServer
    - StockBarchive9998.bar
- Message Sets
- Message Flows
  - StockServerMesgPr
    - (default)
      - StockMesgFlow.msgflow

Domains
- stock_qm@localhost:1414
  - Broker Topology
  - Topics
  - Subscriptions
  - Event Log

Content
Add and remove message flows or message sets from this archive.

Name	Type	Modified	Comment	Size	Path
StockMesgFlow.cmf	Compiled message flow	Nov 17, 2003 4:08:52 PM		4910	

Show source files

Properties

Property	Value
Info	
editable	true
last modified	11/17/03 4:...
name	StockServer
path	/StockServer

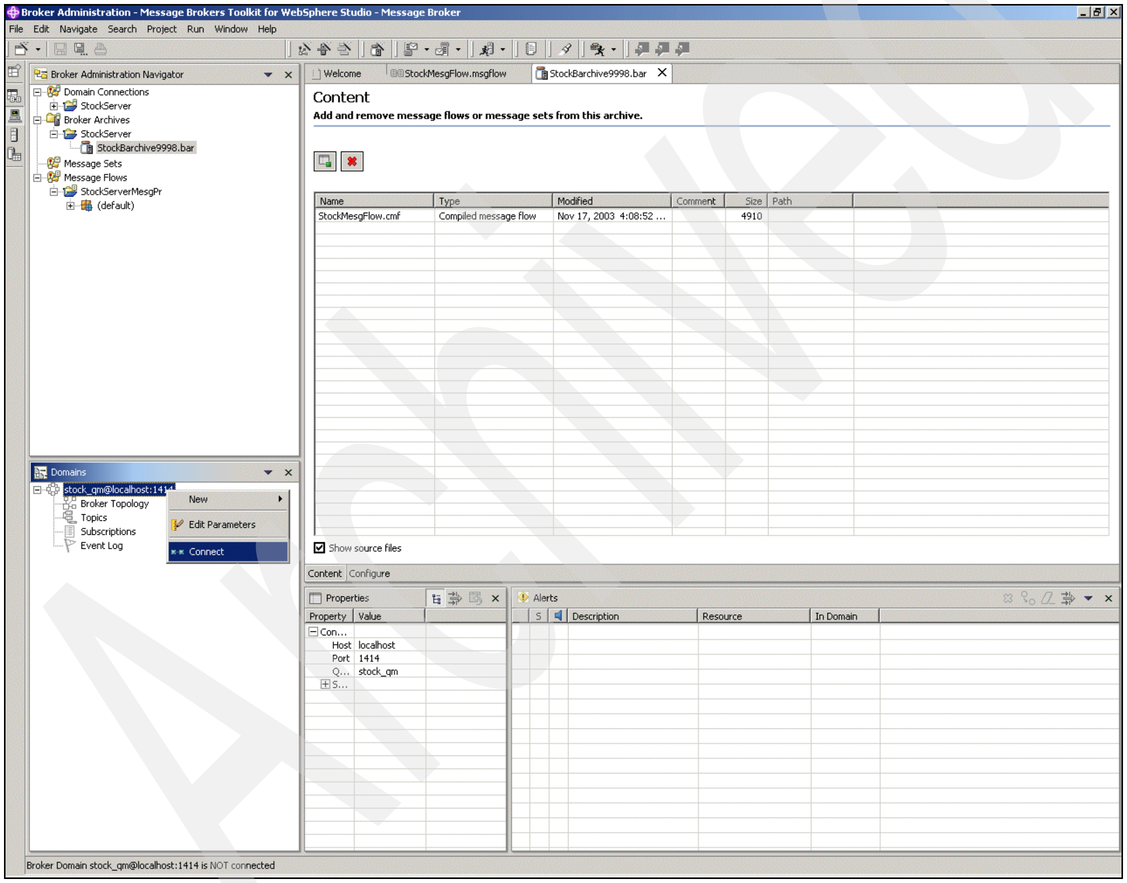
Contains one or several Broker Archive Files

Figure 7-44 Message Flow successfully added to broker archive

Note: It is particularly important to make sure that the message flows or message sets being added to a bar file have been saved prior to being added to the bar file. Changing the message flow source after it has been compiled in a bar file requires repeating several steps described in this section on how to create a new bar file. A compiled message flow source is a static entity and does not automatically change concurrently with changes to the message flow source. While it may seem redundant to create iterations of a bar file, bear in mind that each could account for a different version of the associated message flow. It is important to save a bar file prior to deployment; however, if it is not saved, a prompt is presented for saving the file.

7.3.11 Connecting to a Configuration Manager

At this stage it is necessary to be connected to a Configuration Manager in order to deploy the new bar file into an execution group. While an implicit connect is initiated by an attempt to deploy, an explicit connect is also possible. The process for an explicit connect is described here. The Domains view in the lower left of the Broker Administration perspective shows the Configuration Managers to which access is granted. Select the local Configuration Manager and right-click **Connect**, as shown in Figure 7-45. If you have already been connected to this Configuration Manager then only the Disconnect option is shown.



Content

Add and remove message flows or message sets from this archive.

Name	Type	Modified	Comment	Size	Path
StockMesgFlow.cmf	Compiled message flow	Nov 17, 2003 4:08:52 ...		4910	

Property	Value
⊟ Con...	
Host	localhost
Port	1414
Q...	stock_qm
⊞ S...	

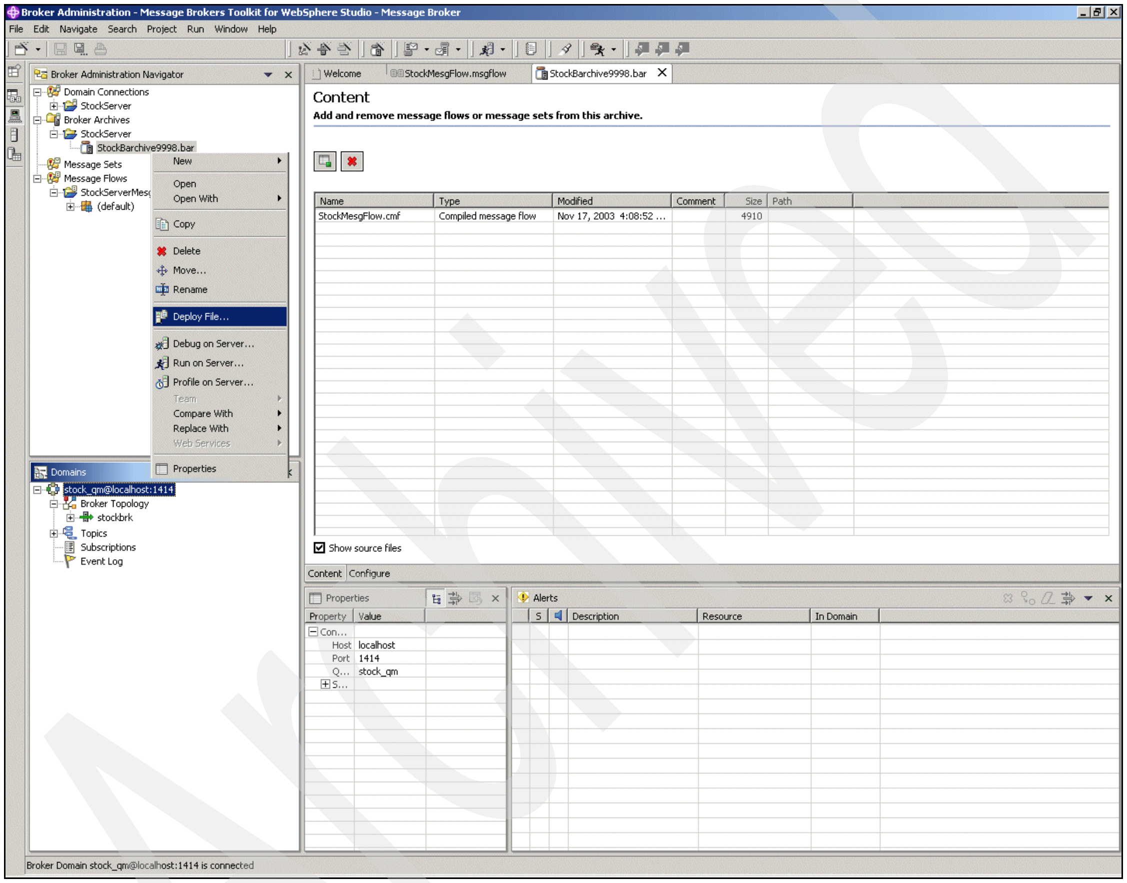
S		Description	Resource	In Domain

Broker Domain stock_qm@localhost:1414 is NOT connected

Figure 7-45 Connecting to Configuration Manager

7.3.12 Deploying a bar file

Once connected to the Configuration Manager, files can be deployed to the brokers managed by this Configuration Manager by selecting the **StockBarchive9998.bar** file in the StockServer project of the Broker Archives folder and right-clicking **Deploy File** as shown in Figure 7-46.



Name	Type	Modified	Comment	Size	Path
StockMesgFlow.cmf	Compiled message flow	Nov 17, 2003 4:08:52 ...		4910	

Property	Value
Con...	
Host	localhost
Port	1414
Q...	stock_qm
S...	

S		Description	Resource	In Domain

Broker Domain stock_qm@localhost:1414 is connected

Figure 7-46 Deploying a bar file

The dialogue box to select the execution group for deployment (shown in Figure 7-47 on page 152) is displayed. Select the execution group that was previously created and click **OK**. The deploy is initiated and messages are sent between the Configuration Manager and the execution group. The Alerts and Domains views dynamically refresh as the deployment is progressing. A Broker Administration information box is shown with the final result of the deploy. It has a Details>> button that can be used to show more details, as shown in Figure 7-48 on page 153. Click **OK** to close this window.

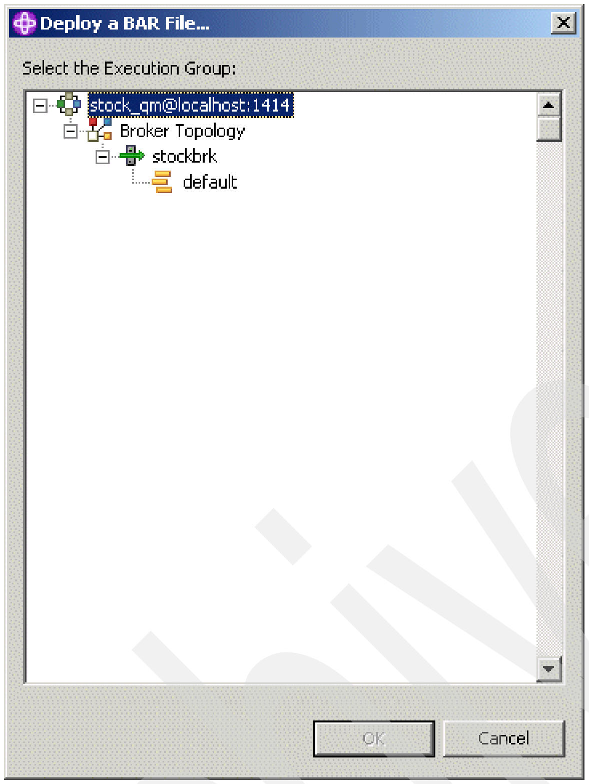
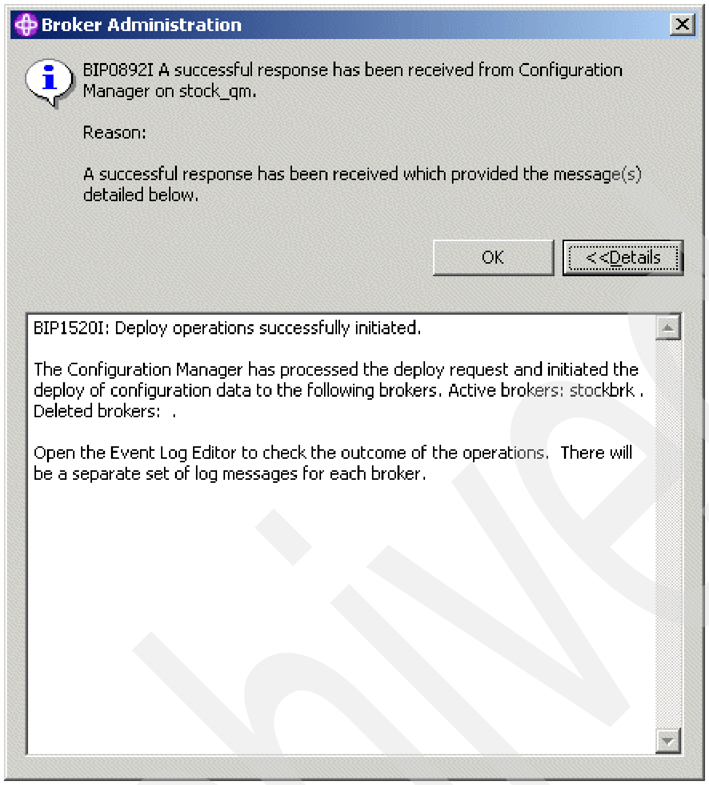


Figure 7-47 Selecting the execution group for a deploy



BIP0892I A successful response has been received from Configuration Manager on stock_qm.

Reason:

A successful response has been received which provided the message(s) detailed below.

BIP1520I: Deploy operations successfully initiated.

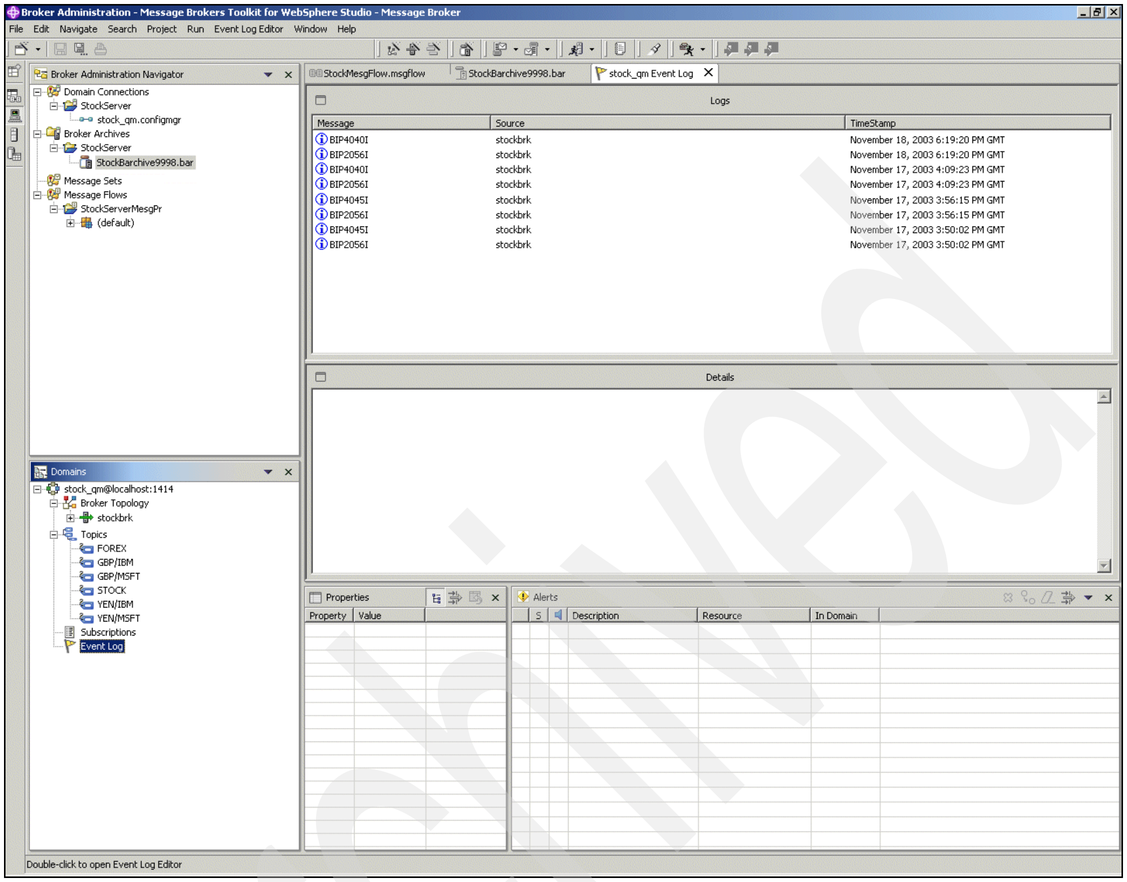
The Configuration Manager has processed the deploy request and initiated the deploy of configuration data to the following brokers. Active brokers: stockbrk . Deleted brokers: .

Open the Event Log Editor to check the outcome of the operations. There will be a separate set of log messages for each broker.

Figure 7-48 Result of successful deploy with details

7.3.13 Checking the deployment

As deploy occurs, event messages are stored in the configuration repository managed by the Configuration Manager. These messages can be viewed using the Event Log editor of the Configuration Manager to determine the success of the deploy. Launch the Event Log editor by double-clicking **Event Log** for the Configuration Manager in the Domains view. The Event Log editor launches, as shown in Figure 7-49 on page 154. The editor has two panes called Logs and Details. As messages are selected in the Logs pane, their details are shown in the Details pane. Messages BIP4040I and BIP2056I indicate that the deploy was successful.



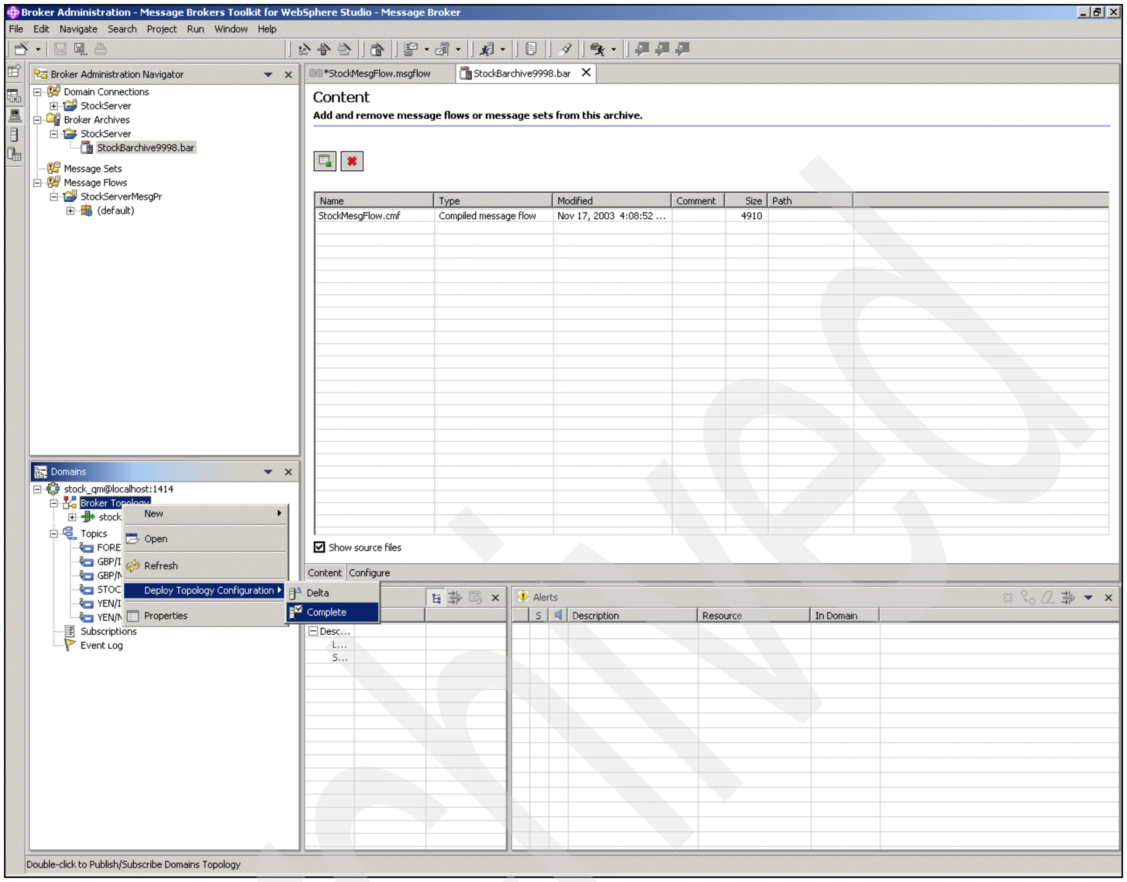
Message	Source	TimeStamp
BIP4040I	stockbrk	November 18, 2003 6:19:20 PM GMT
BIP2056I	stockbrk	November 18, 2003 6:19:20 PM GMT
BIP4040I	stockbrk	November 17, 2003 4:09:23 PM GMT
BIP2056I	stockbrk	November 17, 2003 4:09:23 PM GMT
BIP4045I	stockbrk	November 17, 2003 3:56:15 PM GMT
BIP2056I	stockbrk	November 17, 2003 3:56:15 PM GMT
BIP4045I	stockbrk	November 17, 2003 3:50:02 PM GMT
BIP2056I	stockbrk	November 17, 2003 3:50:02 PM GMT

Figure 7-49 Event Log editor

Deploying topology configuration

This section shows how to deploy broker configuration to the server.

1. Right click **Broker topology -> Deploy Topology Configuration -> Complete**, as shown in Figure 7-50 on page 155.
2. If any error is encountered, then right click (**stock_qm@localhost**) **-> Cancel deployment**.
3. Deploy again by right-clicking **Broker topology -> Deploy Topology Configuration -> Complete**.
4. If it is successful, a message is displayed informing of successful deployment, as shown in Figure 7-51 on page 155.



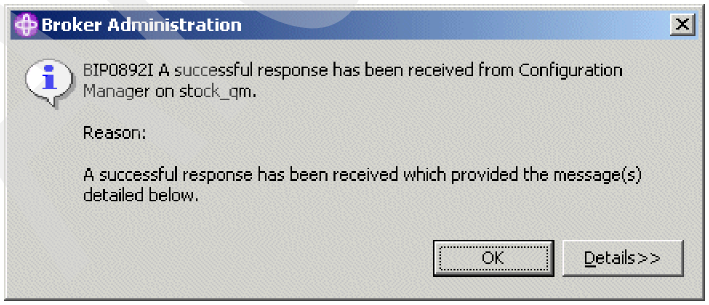
Content

Add and remove message flows or message sets from this archive.

Name	Type	Modified	Comment	Size	Path
StockMesgFlow.cmf	Compiled message flow	Nov 17, 2003 4:08:52 ...		4910	

S		Description	Resource	In Domain

Figure 7-50   Deploying broker topology


Broker Administration

BIP0892I A successful response has been received from Configuration Manager on stock_qm.

Reason:

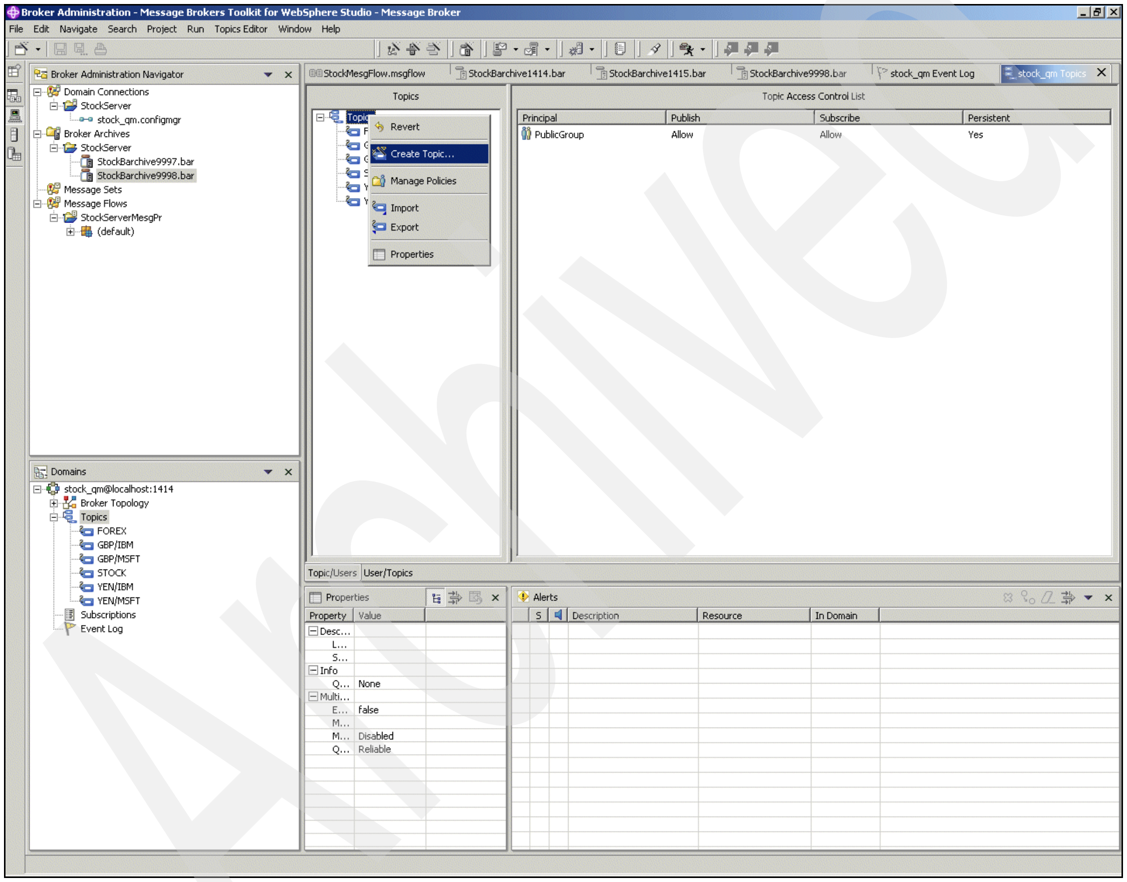
A successful response has been received which provided the message(s) detailed below.

Figure 7-51   Successful deployment

7.3.14 Creation of topics

The steps for the creation of topics are as follows (only required for topics that are to be multicast-enabled or require security control).

Click the **Broker Administrative** perspective in the Domains view. In the lower left corner double click **Topics**. A new development can be viewed. Go to the Topics in that view, right-click the topic, and click **Create topic**, as shown in Figure 7-52.



Topic Access Control List

Principal	Publish	Subscribe	Persistent
PublicGroup	Allow	Allow	Yes

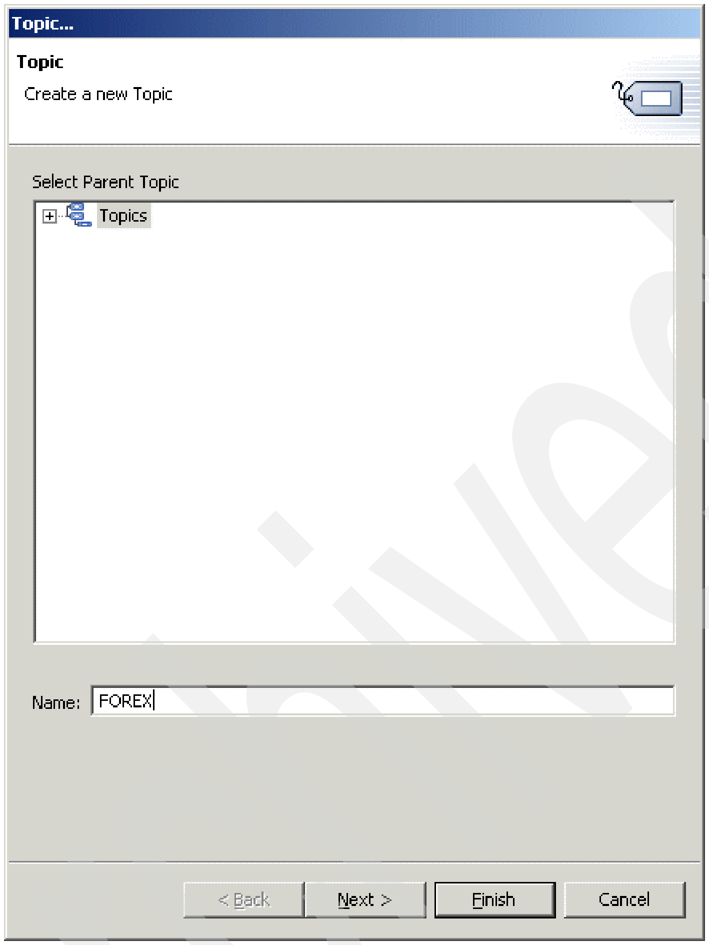
Properties

Property	Value
Desc...	
L...	
S...	
Info	
Q...	None
Multi...	
E...	false
M...	
M...	Disabled
Q...	Reliable

S		Description	Resource	In Domain

Figure 7-52 Step 1: Creation of topics

A dialogue box is presented for creating the Topics by providing the name of the topic, as shown in Figure 7-53 on page 157. In the Domains view a new topic is created with the name given.



Name: FOREX

Figure 7-53 Step 2: Creation of topic

8

Overview of basic scenario

This chapter covers the peripherals of the basic scenario that is used to demonstrate the various configurations of WebSphere Business Integration Event Broker. The same publisher and subscriber applications are used for all the scenarios and no code alteration is needed, demonstrating the power of JMS to provide portable applications. Instead, the differences between the environments are managed in a JNDI namespace, as described below.

The purpose of using the same applications for all the scenarios is to highlight the separation between application programming and infrastructure programming.

In order to use any of the command line programs below, the system's environment needs to be configured. On Windows, the setting of two environment variables is required:

- CLASSPATH needs to include the following files from the C:\Program Files\IBM\WebSphere MQ\Java\lib directory, as well as the directory itself: com.ibm.mq.jar, com.ibm.mqjms.jar, rmm.jar, jms.jar, jndi.jar, jta.jar, connector.jar, providerutil.jar, and fscontext.jar.
- PATH needs to include the C:\Program Files\IBM\WebSphere MQ\Java\lib directory.

These settings have to be available for the command line programs discussed in this chapter.

8.1 JMS Administered Objects

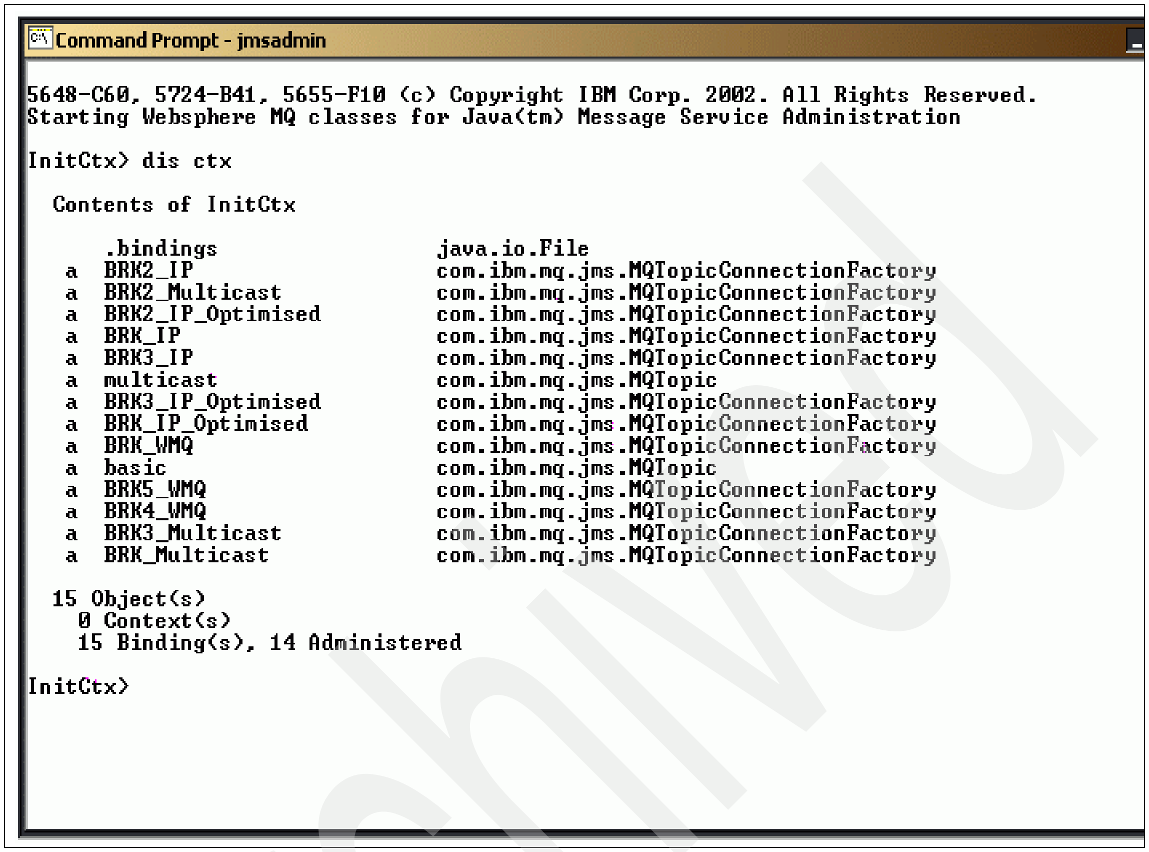
The differences between the brokers required by the different scenarios are masked by the use of different TopicConnectionFactory objects. These are created and managed in a JNDI namespace.

The Java Naming and Directory Interface (JNDI) has a similar concept to the JMS, in that it provides an abstract Java interface. In the case of JMS the interface is to messaging, but JNDI offers an interface to a storage medium for objects that can be serialized to disk. There are many implementations of JNDI, including both Local Directory Access Protocol (LDAP) and the naming concepts used in the WebSphere Application Server. There is also a simple file system implementation provided with the WebSphere MQ JMS (fscontext.jar), and for the purposes of this demonstration, this simple file system context is used.

The tool provided for accessing and managing the WebSphere MQ JMS JNDI objects is called jmsadmin, and is a text-based tool found in the C:\Program Files\IBM\WebSphere MQ\Java\bin directory. It has a configuration file called jmsadmin.config in the same place, and this needs to be edited and the following parameters set:

- INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
- PROVIDER_URL=file:/C:/jndi

A corresponding directory called jndi needs to be created on the C:\ drive. Once this has been done, typing `jmsadmin` at the command line brings up the tool, as shown in Figure 8-1 on page 161.



```
5648-C60, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2002. All Rights Reserved.
Starting Websphere MQ classes for Java(tm) Message Service Administration

InitCtx> dis ctx

  Contents of InitCtx

      .bindings                java.io.File
   a  BRK2_IP                  com.ibm.mq.jms.MQTopicConnectionFactory
   a  BRK2_Multicast           com.ibm.mq.jms.MQTopicConnectionFactory
   a  BRK2_IP_Optimised        com.ibm.mq.jms.MQTopicConnectionFactory
   a  BRK_IP                   com.ibm.mq.jms.MQTopicConnectionFactory
   a  BRK3_IP                  com.ibm.mq.jms.MQTopicConnectionFactory
   a  multicast                com.ibm.mq.jms.MQTopic
   a  BRK3_IP_Optimised        com.ibm.mq.jms.MQTopicConnectionFactory
   a  BRK_IP_Optimised         com.ibm.mq.jms.MQTopicConnectionFactory
   a  BRK_WMQ                  com.ibm.mq.jms.MQTopicConnectionFactory
   a  basic                    com.ibm.mq.jms.MQTopic
   a  BRK5_WMQ                 com.ibm.mq.jms.MQTopicConnectionFactory
   a  BRK4_WMQ                 com.ibm.mq.jms.MQTopicConnectionFactory
   a  BRK3_Multicast           com.ibm.mq.jms.MQTopicConnectionFactory
   a  BRK_Multicast            com.ibm.mq.jms.MQTopicConnectionFactory

  15 Object(s)
    0 Context(s)
    15 Binding(s), 14 Administered

InitCtx>
```

Figure 8-1 JNDI File System Context

8.2 Publisher and subscriber applications

The publisher and subscriber applications are the same for all scenarios, and are simple single-threaded Java applications. The code is fully commented for readability and can be examined in Appendix A, "Code used in the business case scenario" on page 223, but there are a couple of important general JMS coding points which need to be stressed in general JMS terms:

- JMS Sessions are single-threaded objects. In a multi-threaded application, care must be taken to ensure either that each thread has its own Session, or that access to the Session object is serialized.
- JMS objects hold state that is beyond the ownership of the JVM, and therefore cannot be garbage-collected. Therefore, when an application terminates, every effort must be made to ensure that all JMS objects are

closed correctly. In particular, error cases that lead to early program termination must have appropriate JMS closure code.

The code in Appendix A, “Code used in the business case scenario” on page 223 can be compiled with the following commands (assuming the CLASSPATH has been set as above):

```
javac Publisher.java

javac Subscriber.java
```

8.3 Topic creation for the scenarios

There are two topics used for the scenarios, and definitions for these need to be made in both the JNDI namespace and in the Broker Toolkit.

8.3.1 Topic creation in JNDI

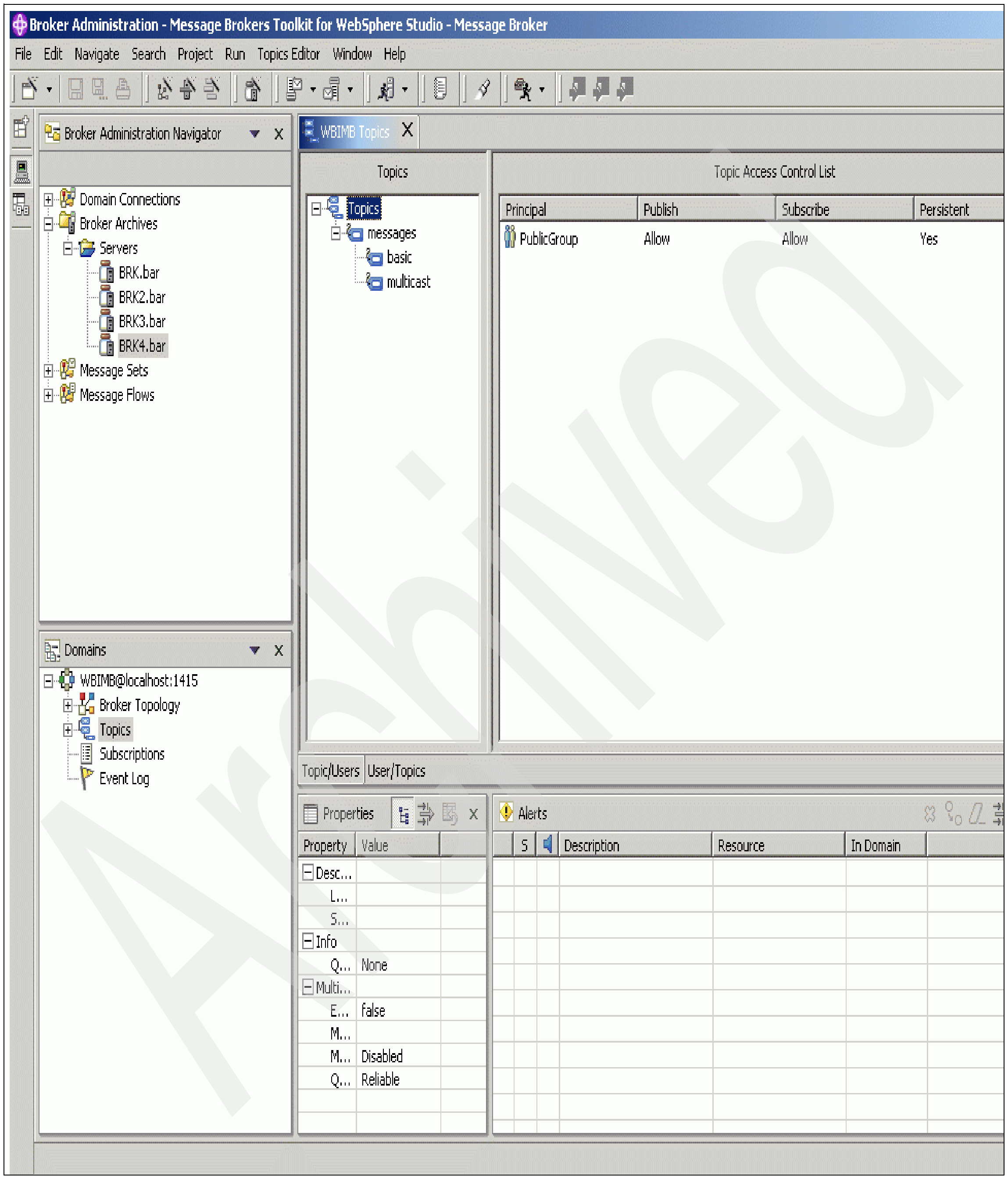
With the JNDI environment set up as described in “JMS Administered Objects” on page 160 above, start the jmsadmin tool and enter the two following commands:

```
def t(basic) topic(messages/basic) brokerver(v2)

def t(multicast) topic(messages/multicast) brokerver(v2)
```

This creates two JNDI lookup keys, *basic* and *multicast*, which JMS applications can then use to find the topics *messages/basic* and *messages/multicast* in the Pub/Sub environment.

8.3.2 Topic creation in the Broker Toolkit

In order for the multicast topic to be correctly multicast-enabled, the topic tree needs to be created in the WebSphere Business Integration Event Broker environment. This is done by double-clicking **Topics** under the local domain connection, and selecting the **Topics/Users**, tag as shown in 8.2, “Publisher and subscriber applications” on page 161.



Principal	Publish	Subscribe	Persistent
PublicGroup	Allow	Allow	Yes

Property	Value
Desc...	
L...	
S...	
Info	
Q...	None
Multi...	
E...	false
M...	
M...	Disabled
Q...	Reliable

Figure 8-2 Topics in the Broker Toolkit

To create the required topics from the root, complete the following:

1. Right-click **Topics**, select **Create Topic...**, enter `messages` as the name, and click **OK**.
2. Right-click the **Messages** topic, select **Create Topic**..., enter `basic` as the name, and click **OK**.
3. Right-click the **Messages** topic, select **Create Topic**..., enter `multicast` as the name, and click **OK**.
4. Right-click the **Multicast** topic, select **Properties** and open the **Multicast** panel. Select **Enabled** for multicast in the pull-down menu and leave the remaining properties as their default values.

These topics match the two JNDI definitions created in the previous section.

Security control could be implemented at this point too, but this is beyond the scope of these demonstrations.

9

Real-time transport for single broker performance

This scenario shows how to configure a single broker for high performance using the WebSphere MQ real-time transport for Pub/Sub.

9.1 Preparation

The preparations necessary for this demonstration are contained in Chapter 7, "Configuration and administration of WBI Event Broker" on page 103, and Chapter 8, "Overview of basic scenario" on page 159. These are as follows:

- Create a queue manager for the broker and Configuration Manager to share.
- Create databases for the Configuration Manager and broker.
- Create a Configuration Manager and a broker.
- Create the topics required for the demonstrations in the broker toolkit.
- Create a JNDI context and store entries for the topics.

9.2 Broker configuration

Open the Broker Toolkit, and select the **Topology** panel. If this is a fresh configuration of WebSphere Business Integration Event Broker, then the panel is empty.

Select **Broker** from the palette and add one to the panel, then right-click the broker icon and select **Properties**. The Broker panel is displayed by default, and this contains an entry for Queue Manager Name, which must be filled in to match the name of the queue manager created for the broker.

Right-click the broker icon again and select **Rename**. The broker's name in the tooling must match that of the broker created in the steps above.

Once these tasks have been completed, save the topology and initiate a complete deploy of the configuration. This must also deploy the topics created earlier to the broker. Select the **Event Log** panel to view the results of the deployment operation; a successful deployment results in messages `BIP2056I` and `BIP4045I`.

9.2.1 Message flows

This scenario uses two message flows. One is for publishing only, and the other is for both publishing and subscribing. They can both only be used by the WebSphere MQ real-time transport (or multicast transport).

The first flow is called IP_Publish_Flow and is shown in Figure 9-1 on page 167.

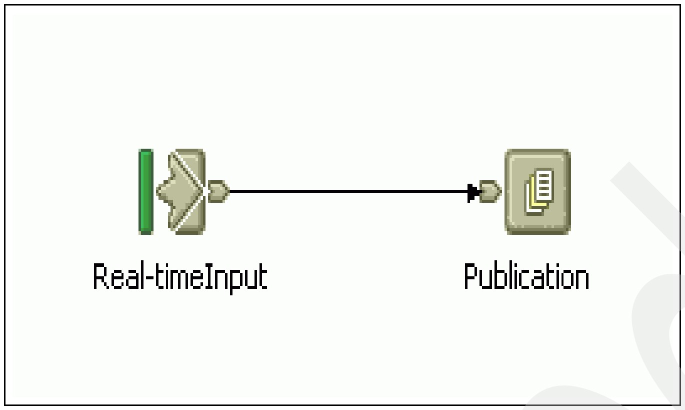
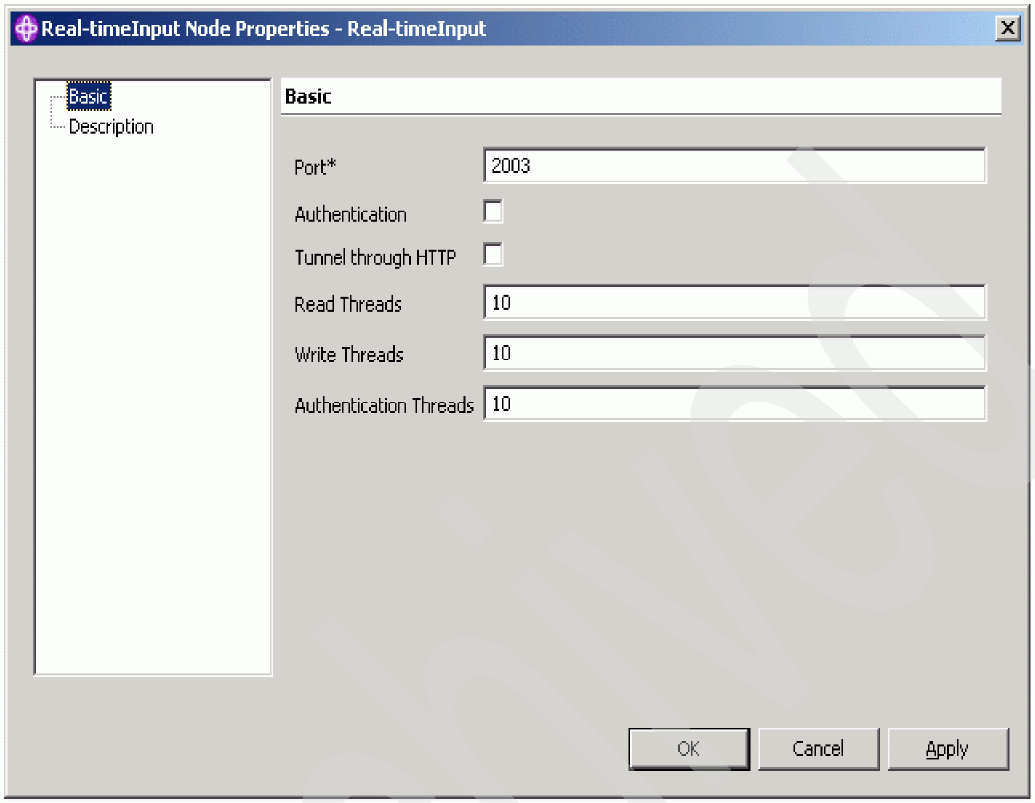


Figure 9-1 IP_Publish_Flow

This needs very little configuration; the only required setting is the Port setting in the Real-timeInputNode. This is accessed by right-clicking the **Real-timeInputNode** and selecting the **Basic** panel. In this case, the port has been set to 2003, as shown in Figure 9-2 on page 168.


Real-timeInput Node Properties - Real-timeInput

- Basic
- Description

Basic

Port*	2003
Authentication	☐
Tunnel through HTTP	☐
Read Threads	10
Write Threads	10
Authentication Threads	10

Figure 9-2 Real-timeInputNode basic properties

The second flow is called IP_Optimised_Flow and is shown in Figure 9-3 on page 169.

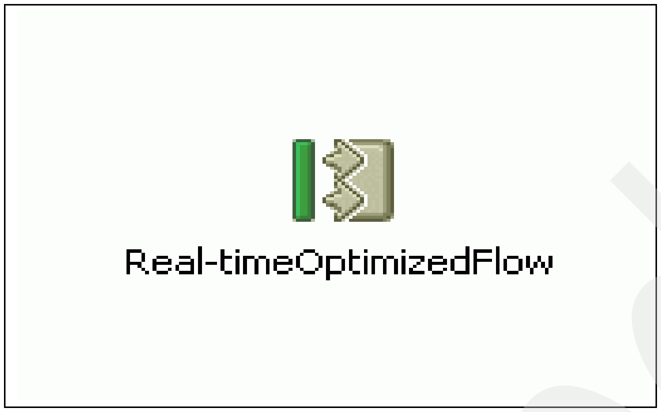
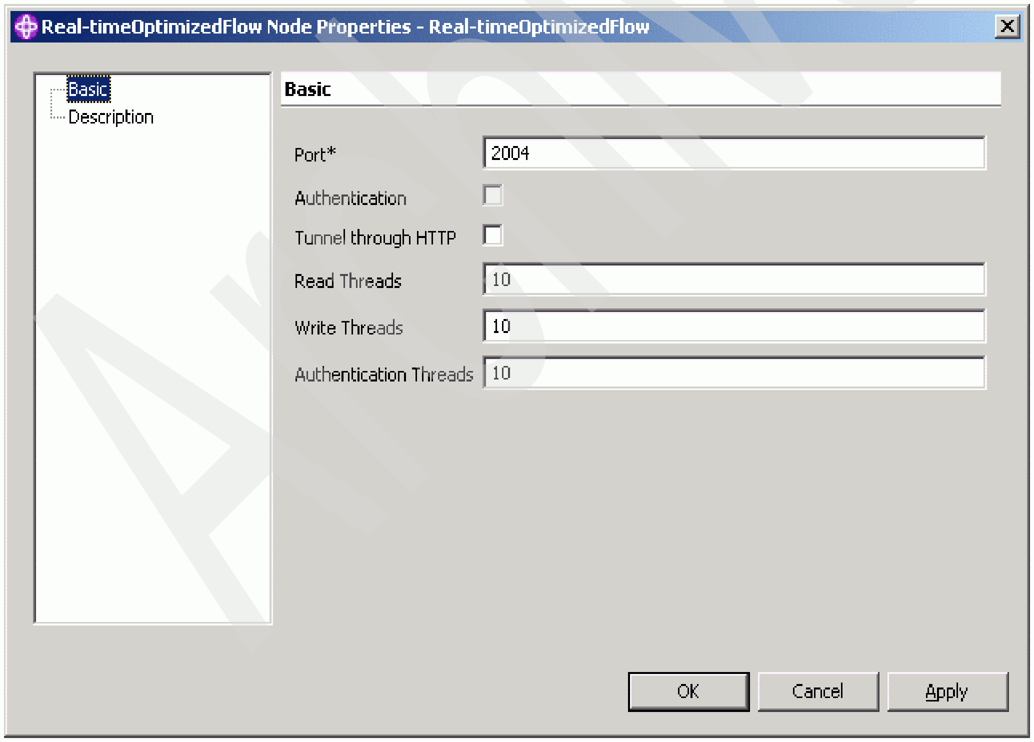


Figure 9-3 IP_Optimized_Flow

This also needs very little configuration; only the Port needs to be set. This is accessed in the same way as described above, and as shown in Figure 9-4, has been set to 2004.


Real-timeOptimizedFlow Node Properties - Real-timeOptimizedFlow
Basic
Description
Basic
OK
Cancel
Apply

Port*	2004
Authentication	☐
Tunnel through HTTP	☐
Read Threads	10
Write Threads	10
Authentication Threads	10

Figure 9-4 Real-timeOptimizedFlow basic properties

Once the message flows have been created and configured, create a new Message Broker Archive file for use in deploying them to the broker. Add the two flows to the new .bar file, as shown in Chapter 7, "Configuration and administration of WBI Event Broker" on page 103, and deploy the .bar file to the broker's default execution group. Check the Event Log as before, this time for BIP2056I and BIP4040I messages showing success.

Note: IP_Publish_Flow can be completely substituted with IP_Optimised_Flow, which is the recommended design for Pub/Sub using the WebSphere MQ real-time transport. This publication uses two flows to demonstrably separate the publishing and subscribing applications across different flows, to show that matching of messages to subscriptions is not flow-dependant.

9.3 JNDI configuration

TopicConnectionFactory references are needed to match both message flows created above. Access the JNDI context as shown in Chapter 8, "Overview of basic scenario" on page 159, and execute the following commands:

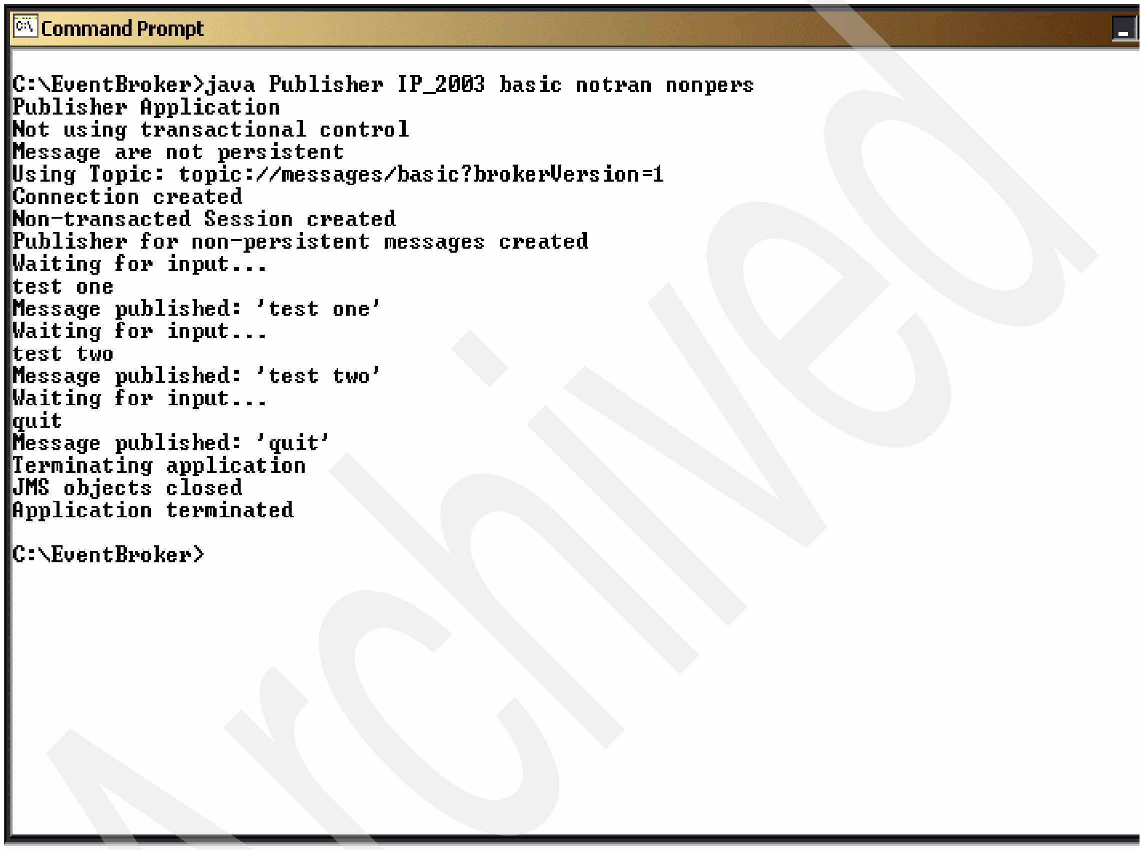
```
def tcf(IP_2003) brokerver(v2) transport(direct) hostname(localhost) port(2003)

def tcf(IP_2004) brokerver(v2) transport(direct) hostname(localhost) port(2004)
```

The publishing and subscribing applications use these objects to determine how to connect to the broker. In this case, the important points are hostname and port, which correspond with the port settings in the flows above. The transport setting ensures that the WebSphere MQ real-time transport is used, and the brokerver setting tells the WebSphere MQ JMS implementation that the broker understands Pub/Sub information when held only in a WebSphere MQ RFH2 header structure.

9.4 Publishing and subscribing

One publisher and one subscriber are sufficient to show that the scenario is correctly functioning. Both the publisher and the subscriber could use the IP_2004 TopicConnectionFactory, but using a different flow for publishing demonstrates the power of WebSphere Business Integration Event Broker, in that publications are mapped to subscriptions regardless of the transport type or message flow being used. This is shown further in the stream-crossing demonstration in Chapter 13, "Stream-crossing for single broker flexibility" on page 199.

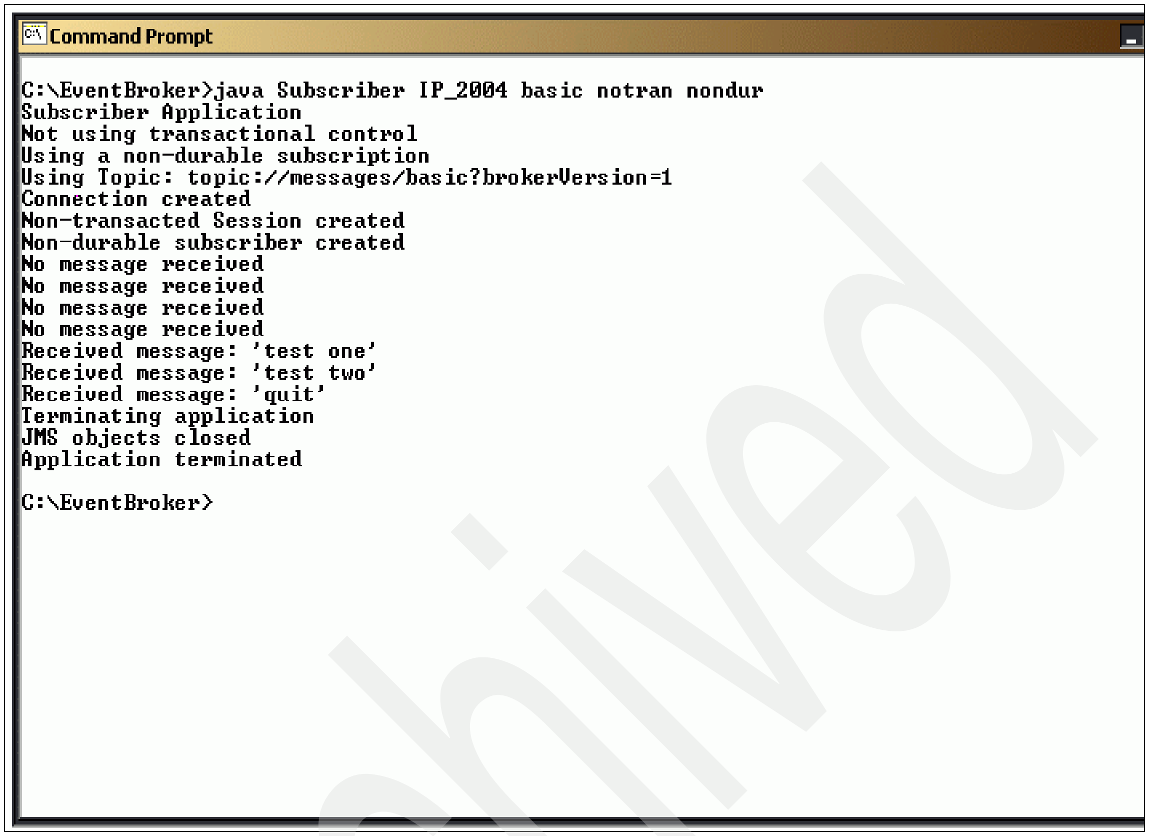
The publisher in this case connects to the broker, sends two publications, and then sends a further publication instructing any subscribers to terminate. This is shown in Figure 9-5. The subscriber must be started before the publications are sent (if not, there are no matching subscriptions for the publications and the broker therefore discards them).



```
C:\EventBroker>java Publisher IP_2003 basic notran nonpers
Publisher Application
Not using transactional control
Message are not persistent
Using Topic: topic://messages/basic?brokerVersion=1
Connection created
Non-transacted Session created
Publisher for non-persistent messages created
Waiting for input...
test one
Message published: 'test one'
Waiting for input...
test two
Message published: 'test two'
Waiting for input...
quit
Message published: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 9-5 The publisher

The subscriber receives the messages, even though it uses a different TopicConnectionFactory. This is because the broker matches publications to subscriptions at a general level underneath the transport and flow, as is shown in Figure 9-6 on page 172.



```
C:\EventBroker>java Subscriber IP_2004 basic notran nondur
Subscriber Application
Not using transactional control
Using a non-durable subscription
Using Topic: topic://messages/basic?brokerVersion=1
Connection created
Non-transacted Session created
Non-durable subscriber created
No message received
No message received
No message received
No message received
Received message: 'test one'
Received message: 'test two'
Received message: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 9-6 The subscriber

10

Multicast transport for single broker performance

This scenario shows how to configure a single broker for high performance using the WebSphere MQ Reliable Multicast Messaging (RMM) transport.

10.1 Preparation

The preparations necessary for this demonstration are contained in Chapter 7, “Configuration and administration of WBI Event Broker” on page 103, and Chapter 8, “Overview of basic scenario” on page 159. These are as follows:

- Create a queue manager for the broker and Configuration Manager to share.
- Create databases for the Configuration Manager and broker.
- Create a Configuration Manager and a broker.
- Create the topics required for the demonstrations in the Broker Toolkit.
- Create a JNDI context and store entries for the topics.

10.2 Broker configuration

Open the Broker Toolkit, and select the **Topology** panel. If this is a fresh configuration of WebSphere Business Integration Event Broker, then the panel is empty.

Select **Broker** from the palette and add one to the panel, then right-click the broker icon and select **Properties**. The Broker panel is displayed by default, and this contains an entry for Queue Manager Name, which must be filled in to match the name of the queue manager created for the broker.

Right-click the broker icon again and select **Rename**. The broker's name in the tooling must match that of the broker created in the steps above.

Once these tasks have been completed, save the topology and initiate a complete deploy of the configuration. This must also deploy the topics created earlier to the broker. Select the **Event Log** panel to view the results of the deployment operation; a successful deployment results in messages `BIP2056I` and `BIP4045I`.

Restart the broker and manually repeat the deployment of the Topics Hierarchy (right-click **Topics** and select **Deploy Topics Configuration -> Complete)**. This step is necessary and required to ensure that the multicast-enabled topic is deployed to the broker when it is also configured for multicast.

10.2.1 Message flows

The configuration and deployment of the required message flows is identical to that in Chapter 9, “Real-time transport for single broker performance” on page 165. See “Message flows” on page 166 for further details.

10.3 JNDI configuration

TopicConnectionFactory references are needed to match both message flows being used, and one of these is different from the previous chapter (despite the fact that the message flows are the same). Access the JNDI context as shown in Chapter 8, “Overview of basic scenario” on page 159, and execute the following commands:

```
def tcf(IP_2003) brokerver(v2) transport(direct) hostname(localhost) port(2003)

def tcf(Multicast_2004) brokerver(v2) transport(direct) hostname(localhost)
port(2004) multicast(reliable)
```

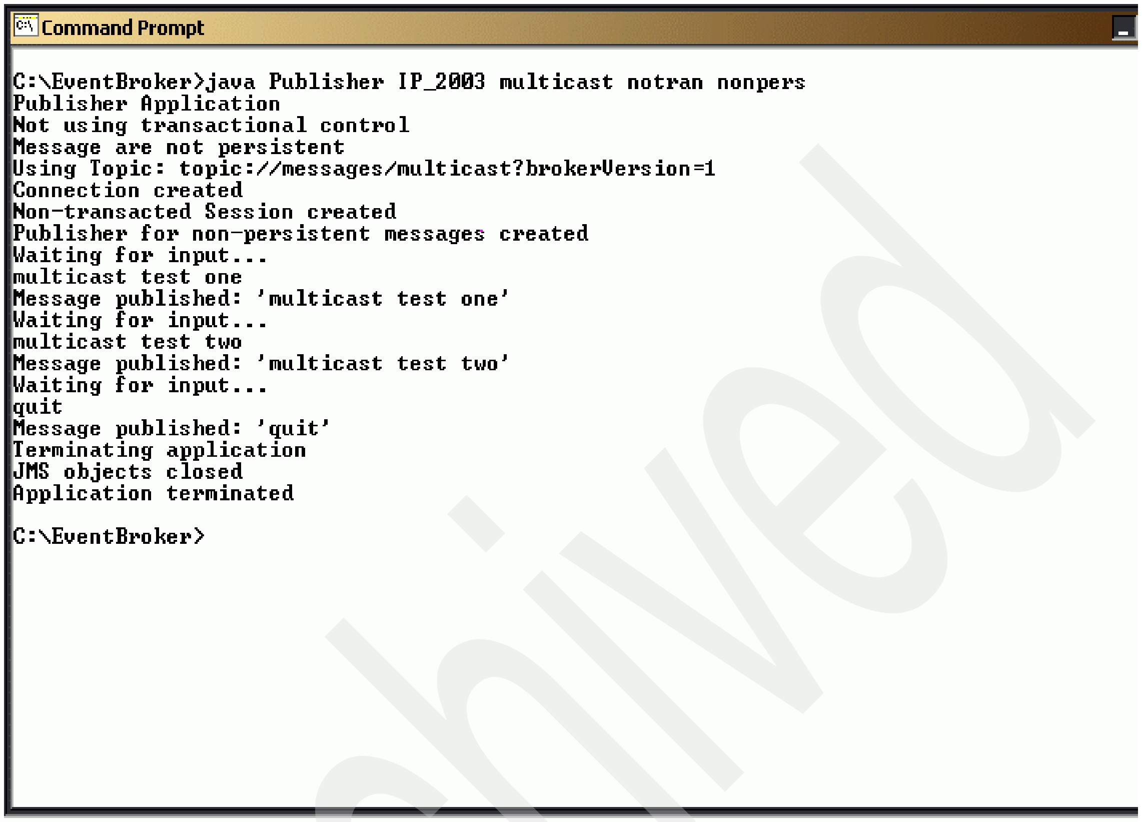
The publishing and subscribing applications use these objects to determine how to connect to the broker. Most of the settings above are detailed in “JNDI configuration” on page 170, and the final parameter, multicast, instructs the WebSphere MQ JMS application to use the multicast transport. This parameter has three settings:

- *Disabled* ensures that only the standard WebSphere MQ real-time transport is used.
- *Enabled* uses multicast if the broker supports it, but falls back to the real-time transport if it does not.
- *Reliable* forces uses of the multicast transport, and if this is not supported by the broker then an error occurs (see Figure 10-3 on page 178 below).

10.4 Publishing and subscribing

One publisher and one subscriber are sufficient to show that the scenario is correctly functioning. Both the publisher and the subscriber could use the IP_2004 TopicConnectionFactory, but using a different flow for publishing demonstrates the power of WebSphere Business Integration Event Broker, in that publications are mapped to subscriptions regardless of the transport type or message flow being used. This is shown further in the stream-crossing demonstration in Chapter 13, “Stream-crossing for single broker flexibility” on page 199.

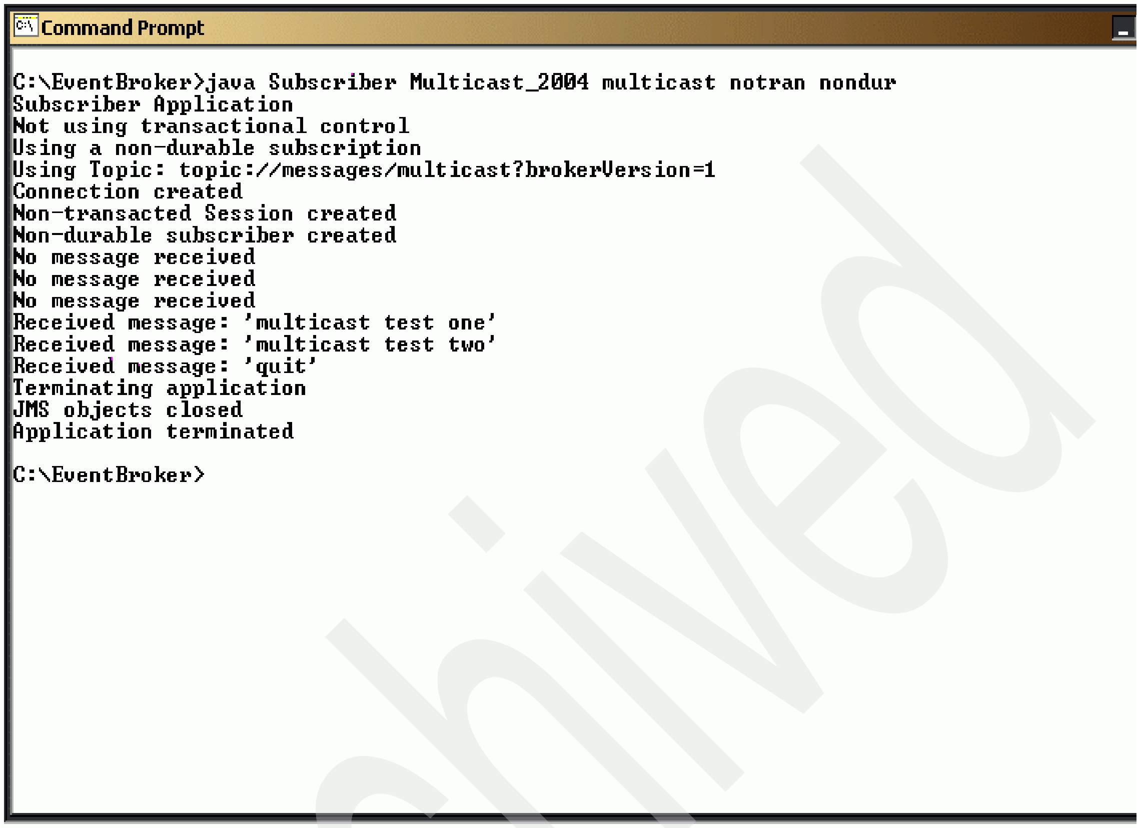
The publisher in this case connects to the broker, sends two publications, and then sends a further publication instructing any subscribers to terminate. This is shown in Figure 10-1 on page 176. The subscriber must be started before the publications are sent (if not, there are no matching subscriptions for the publications and the broker therefore discards them).



```
C:\EventBroker>java Publisher IP_2003 multicast notran nonpers
Publisher Application
Not using transactional control
Message are not persistent
Using Topic: topic://messages/multicast?brokerVersion=1
Connection created
Non-transacted Session created
Publisher for non-persistent messages created
Waiting for input...
multicast test one
Message published: 'multicast test one'
Waiting for input...
multicast test two
Message published: 'multicast test two'
Waiting for input...
quit
Message published: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 10-1 The publisher

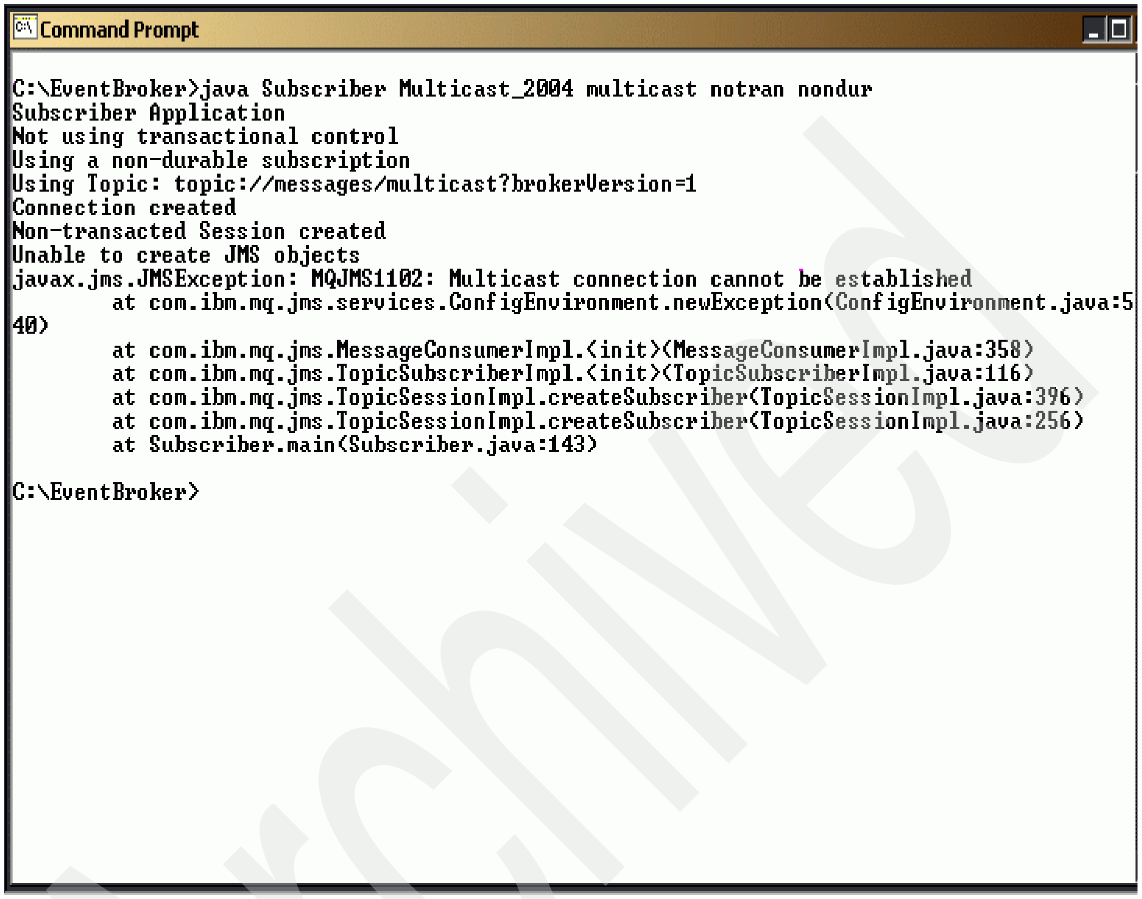
The subscriber receives the messages, even though it uses a different TopicConnectionFactory. This is because the broker matches publications to subscriptions at a general level underneath the transport and flow, and is shown in Figure 10-2 on page 177 below.



```
C:\EventBroker>java Subscriber Multicast_2004 multicast notran nondur
Subscriber Application
Not using transactional control
Using a non-durable subscription
Using Topic: topic://messages/multicast?brokerVersion=1
Connection created
Non-transacted Session created
Non-durable subscriber created
No message received
No message received
No message received
Received message: 'multicast test one'
Received message: 'multicast test two'
Received message: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 10-2 The subscriber

If the broker has not been correctly configured for multicast support, the subscriber application generates an error, as shown in Figure 10-3 on page 178 below.



```
C:\EventBroker>java Subscriber Multicast_2004 multicast notran nondur
Subscriber Application
Not using transactional control
Using a non-durable subscription
Using Topic: topic://messages/multicast?brokerVersion=1
Connection created
Non-transacted Session created
Unable to create JMS objects
javax.jms.JMSException: MQJMS1102: Multicast connection cannot be established
        at com.ibm.mq.jms.services.ConfigEnvironment.newException(ConfigEnvironment.java:5
40)
        at com.ibm.mq.jms.MessageConsumerImpl.<init>(MessageConsumerImpl.java:358)
        at com.ibm.mq.jms.TopicSubscriberImpl.<init>(TopicSubscriberImpl.java:116)
        at com.ibm.mq.jms.TopicSessionImpl.createSubscriber(TopicSessionImpl.java:396)
        at com.ibm.mq.jms.TopicSessionImpl.createSubscriber(TopicSessionImpl.java:256)
        at Subscriber.main(Subscriber.java:143)

C:\EventBroker>
```

Figure 10-3 A failing multicast subscriber

11

Real-time transport for broker collective performance

This scenario shows how to configure a broker collective for high performance using the WebSphere MQ real-time transport for Pub/Sub.

11.1 Preparation

The preparations necessary for this demonstration are contained in Chapter 7, “Configuration and administration of WBI Event Broker” on page 103, and Chapter 8, “Overview of basic scenario” on page 159. These are as follows:

- Create a queue manager for the broker and Configuration Manager to share.
- Create databases for the Configuration Manager and broker.
- Create a Configuration Manager and a broker.
- Create the topics required for the demonstrations in the Broker Toolkit.
- Create a JNDI context and store entries for the topics.

11.2 Broker configuration

First, another broker must be created in order to join with the first broker in the collective. This needs a separate queue manager and database created before the new broker can be created (as shown in Chapter 7, “Configuration and administration of WBI Event Broker” on page 103).

> **Note:** The second queue manager's listener needs a different port to that of the first if they are on the same machine.

The two queue managers need to be connected so that the Configuration Manager can communicate with the second broker. (As an aside, if WebSphere MQ queues were to be used for the subscriber traffic, then the queue manager connections would also be used by the two brokers to share publications.) To connect the queue managers, run the **runmqsc** command for each of them, and define the required objects by the following commands at either end:

```
def ql(<other qmgr name>) usage(xmitq) trigger trigdata(<other qmgr name>)
initq(system.channel.initq)

def chl(<this qmgr name>) chltype(rcvr) trptype(tcp)

def chl(<other qmgr name>) chltype(sdr) trptype(tcp) xmitq(<other qmgr name>)
conname('<hostname>(<port>)')
```

The channels above start automatically when messages are placed on the transmission queue. The **runmqsc** command automatically capitalizes all characters it processes, so if the queue managers do not have names that are all uppercase, then single quotes need to be placed as shown:

```
def ql('exampleQM') usage(xmitq)
```

11.2.1 Broker topology

Open the Broker Toolkit, and select the **Topology** panel. If this is a fresh configuration of WebSphere Business Integration Event Broker, then the panel is empty.

Select **Broker** from the palette and add one to the panel, then right-click the broker icon and select **Properties**. The Broker panel is displayed by default, and this contains an entry for Queue Manager Name, which must be filled in to match the name of the queue manager created for the broker. In the same panel, the Interbroker Host Name must be set to the hostname of the machine that the broker is running on, and the Interbroker Port must be set to a port on that machine. In this example, both brokers are running on the same machine, so the hostname is localhost. The selected ports are 1520 and 1521.

Right-click the broker icon again and select **Rename**. The broker's name in the tooling must match that of the broker created in the steps above.

Repeat all the steps above for the second broker (using different values where required), and once this has been added, select **Collective** from the palette and add one to the topology panel. Select **Connection** and wire the brokers into the collective. This is shown in Figure 11-1 on page 182 below.

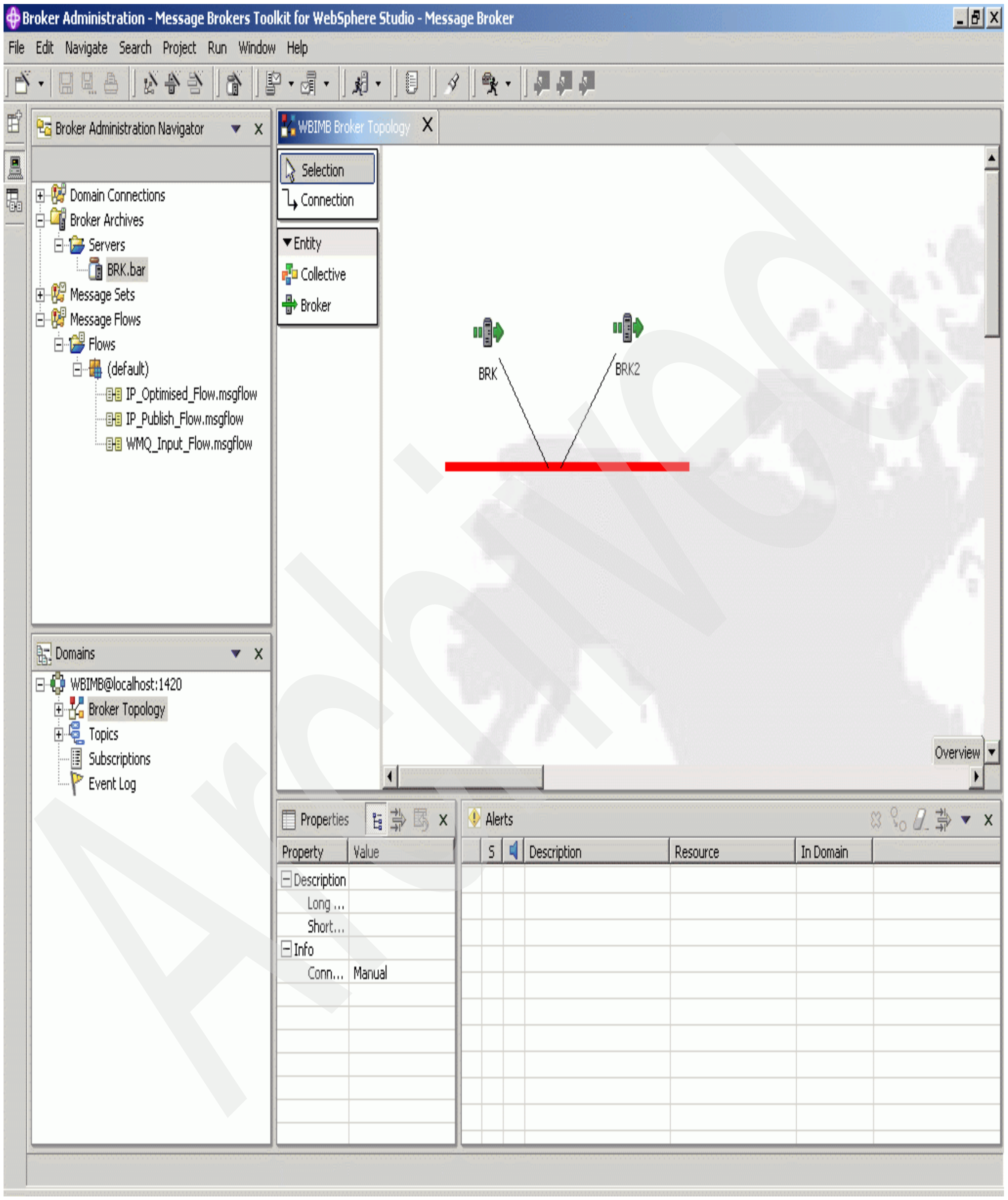
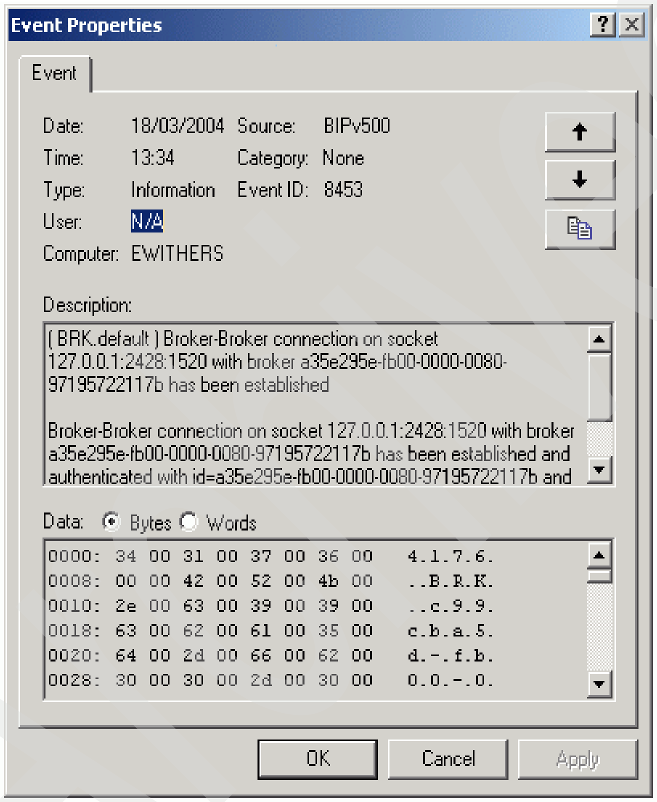


Figure 11-1 A broker collective

Once these tasks have been completed, save the topology and initiate a complete deploy of the configuration. This must also deploy the topics created earlier to the broker. Select the **Event Log** panel to view the results of the deployment operation; a successful deployment results in messages `BIP2056I` and `BIP4045I` (for each broker). The brokers must be restarted for the changes to take effect, and successful initiation of the collective can be verified in the main operating system log. For example, on the Windows platform, two brokers initializing a collective each put a message to the Application Log which can be seen in Event Viewer, as shown in Figure 11-2 below.



Date: 18/03/2004 Source: BIPv500
Time: 13:34 Category: None
Type: Information Event ID: 8453
User: N/A
Computer: EWITHERS

Description:

[BRK.default] Broker-Broker connection on socket 127.0.0.1:2428:1520 with broker a35e295e-fb00-0000-0080-97195722117b has been established

Broker-Broker connection on socket 127.0.0.1:2428:1520 with broker a35e295e-fb00-0000-0080-97195722117b has been established and authenticated with id=a35e295e-fb00-0000-0080-97195722117b and

Data: [✓ Bytes] Words

```
0000: 34 00 31 00 37 00 36 00   4.1.7.6.
0008: 00 00 42 00 52 00 4b 00   ..B.R.K.
0010: 2e 00 63 00 39 00 39 00   ..c.9.9.
0018: 63 00 62 00 61 00 35 00   c.b.a.5.
0020: 64 00 2d 00 66 00 62 00   d.-.f.b.
0028: 30 00 30 00 2d 00 30 00   0.0.-.0.
```

Figure 11-2 A successful collective message

11.2.2 Message flows

The message flows used are the same as those described in "Message flows" on page 166. Once deployment to one broker's default execution group has been successful, edit the broker archive file to change the ports for the flows on the second broker. This is done by selecting the **Configure** tab, expanding the flows down to their configurable parameters, and changing the ports from 2003 and

2004 to 2005 and 2006, respectively. Once this is done, deploy the .bar file to the second broker's default execution group.

11.3 JNDI configuration

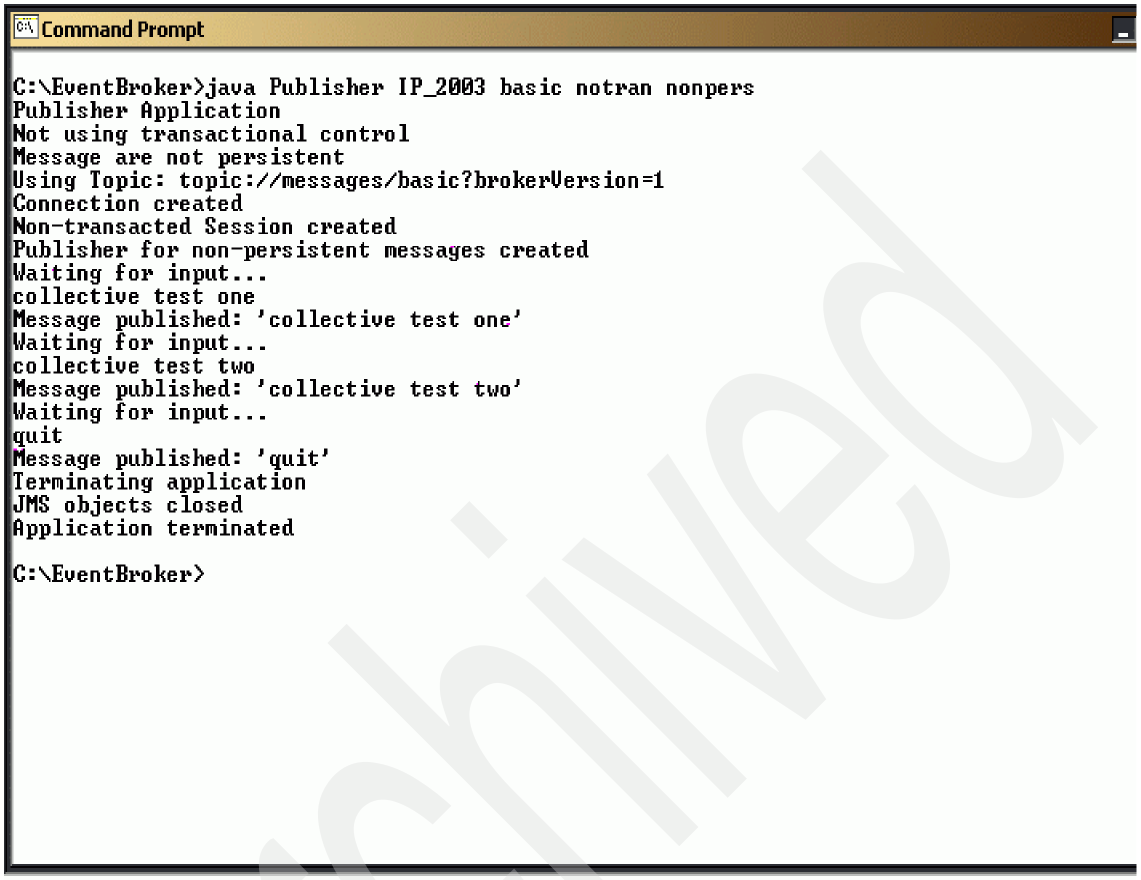
TopicConnectionFactory references are needed to match all message flows created above. Access the JNDI context as shown in Chapter 8, “Overview of basic scenario” on page 159, and execute the following commands:

```
def tcf(IP_2003) brokerver(v2) transport(direct) hostname(localhost) port(2003)

def tcf(IP_2004) brokerver(v2) transport(direct) hostname(localhost) port(2004)

def tcf(IP_2005) brokerver(v2) transport(direct) hostname(localhost) port(2005)

def tcf(IP_2006) brokerver(v2) transport(direct) hostname(localhost) port(2006)
```

The publishing and subscribing applications use these objects to determine how to connect to the broker. The settings above are detailed in “JNDI configuration” on page 170.

11.4 Publishing and subscribing

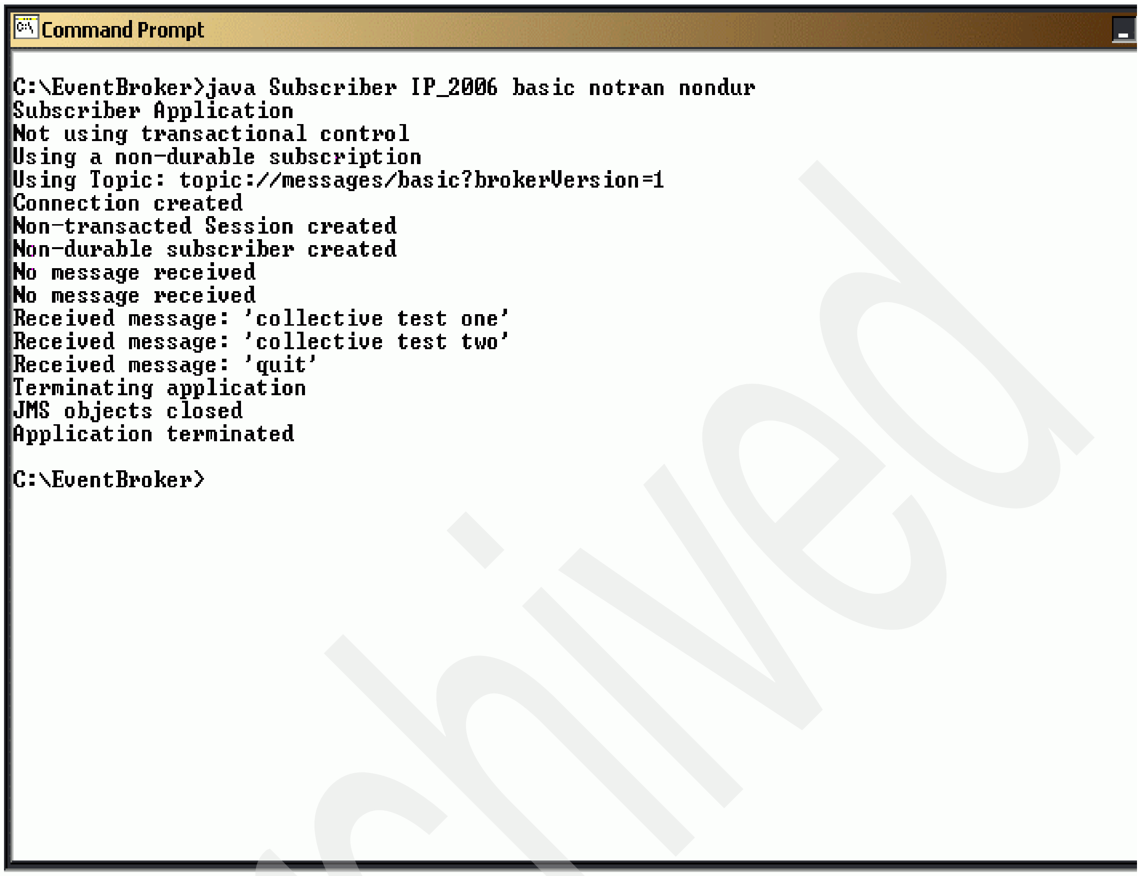
One publisher and one subscriber are sufficient to show that the scenario is correctly functioning. The publisher in this case connects to the first broker, sends two publications and then sends a further publication instructing any subscribers to terminate. This is shown in 11.3, “JNDI configuration” on page 184 below. The subscriber must be started before the publications are sent (if not, there are no matching subscriptions for the publications and the broker therefore discards them).



```
C:\EventBroker>java Publisher IP_2003 basic notran nonpers
Publisher Application
Not using transactional control
Message are not persistent
Using Topic: topic://messages/basic?brokerVersion=1
Connection created
Non-transacted Session created
Publisher for non-persistent messages created
Waiting for input...
collective test one
Message published: 'collective test one'
Waiting for input...
collective test two
Message published: 'collective test two'
Waiting for input...
quit
Message published: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 11-3 The publisher

The subscriber connects to the second broker in the collective, and receives the messages as they are shared across the collective at the broker level. This is shown in 11.4, “Publishing and subscribing” on page 184, and demonstrates that the same application can be used for single and multiple broker configurations, so long as JNDI is correctly used.



```
C:\EventBroker>java Subscriber IP_2006 basic notran nondur
Subscriber Application
Not using transactional control
Using a non-durable subscription
Using Topic: topic://messages/basic?brokerVersion=1
Connection created
Non-transacted Session created
Non-durable subscriber created
No message received
No message received
Received message: 'collective test one'
Received message: 'collective test two'
Received message: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 11-4 The subscriber

12

Persistent messages for single broker reliability

This scenario shows how to configure a single broker for WebSphere MQ message persistence, durable subscriptions and transactional control.

12.1 Preparation

The preparations necessary for this demonstration are contained in Chapter 7, “Configuration and administration of WBI Event Broker” on page 103, and Chapter 8, “Overview of basic scenario” on page 159. These are as follows:

- Create a queue manager for the broker and Configuration Manager to share.
- Create databases for the Configuration Manager and broker.
- Create a Configuration Manager and a broker.
- Create the topics required for the demonstrations in the Broker Toolkit.
- Create a JNDI context and store entries for the topics.

12.2 Broker configuration

First, the broker's queue manager must be configured to use WebSphere MQ queues as the messaging transport. This is accomplished by running a supplied script against the queue manager, by executing the following command:

```
runmqsc WBIMB3 < "C:\Program Files\IBM\WebSphere MQ\Java\bin\MQJMS_PSQ.mqsc"
```

The queue manager also needs a queue on which it receives incoming publications. Historically this has been SYSTEM.BROKER.DEFAULT.STREAM (and is so in this demonstration), but there is no restriction that says this must be the case. The queue is created by the following command in runmqsc:

```
def ql(system.broker.default.stream)
```

Once the queue manager has been configured correctly, complete the broker topology as detailed at the beginning of “Broker configuration” on page 166.

12.2.1 Message flows

This scenario uses one new message flow, which can only be used by the WebSphere MQ queue-based transport. It is called WMQ_Publish_Flow and is shown in Figure 12-1 on page 189.

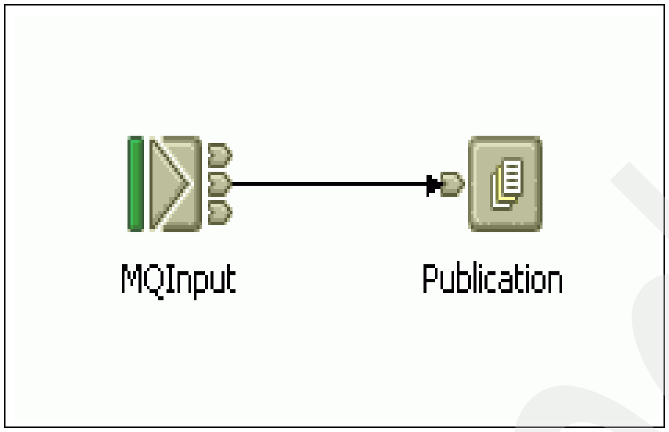
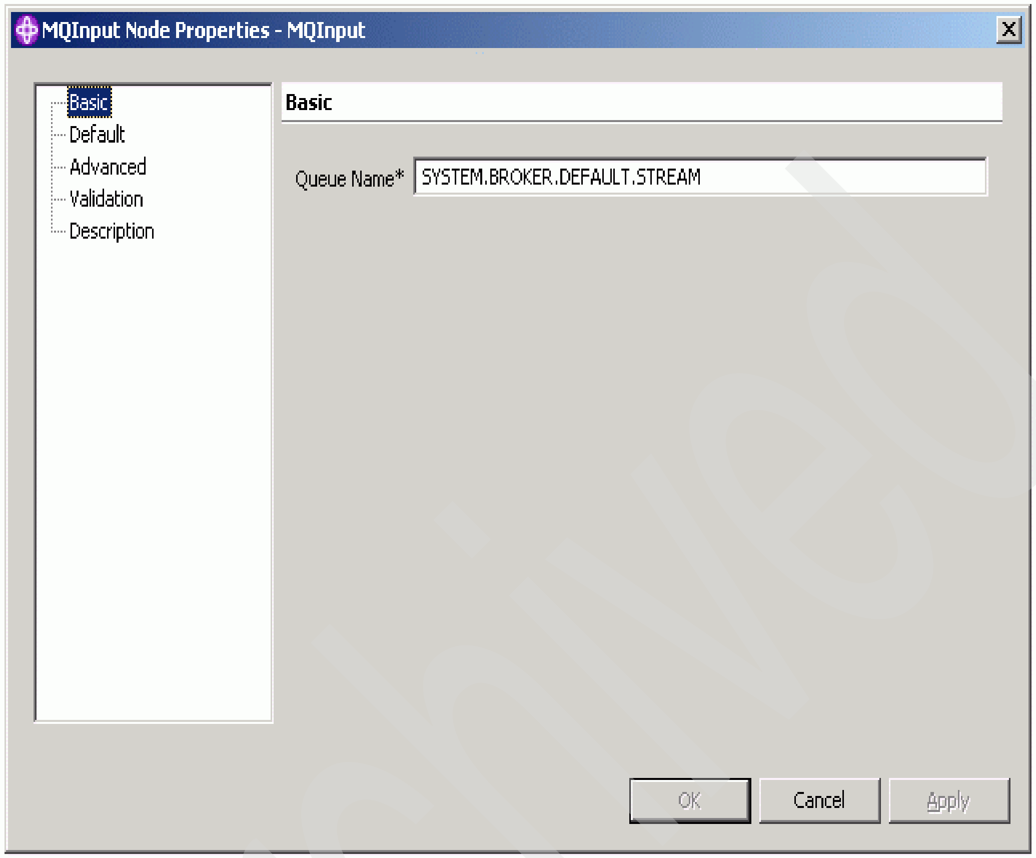


Figure 12-1 WMQ_Publish_Flow

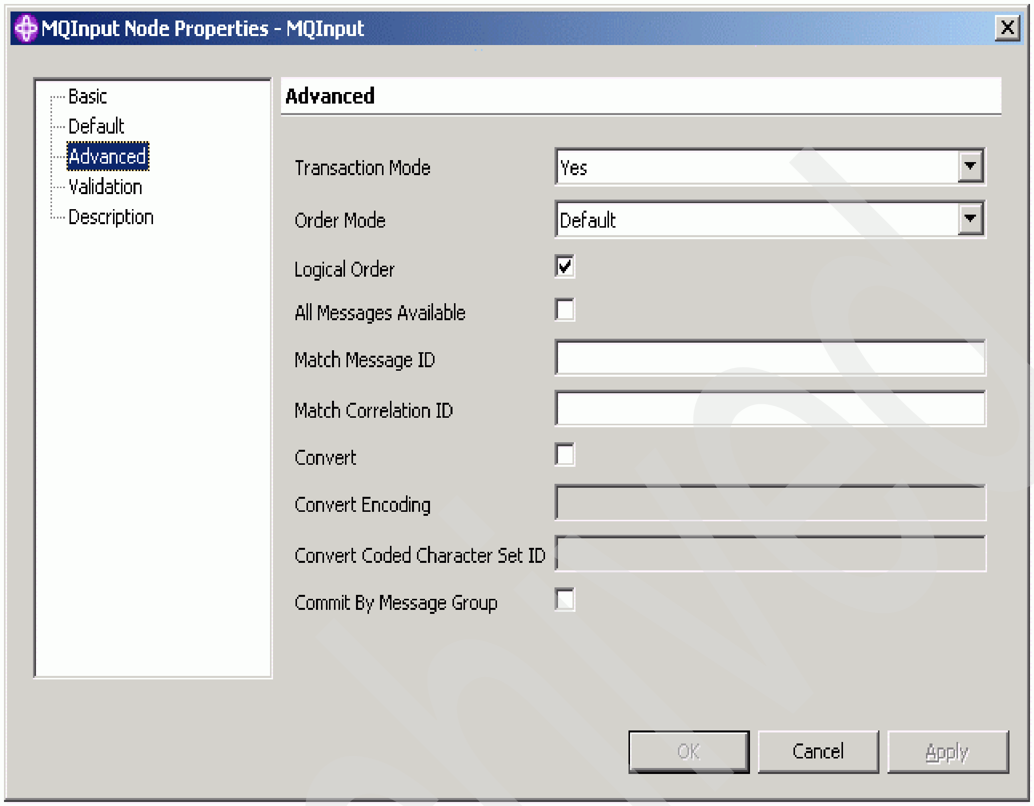
This needs some configuration; the only MQInputNode has two settings that need completing. These are accessed by right-clicking the **MQInputNode**. The first is in the Basic panel, and is called Queue Name. This is the queue that the flow reads messages from for publication, and is set to SYSTEM.BROKER.DEFAULT.STREAM in this example. This is shown in the 12.2, “Broker configuration” on page 188.



Queue Name* SYSTEM.BROKER.DEFAULT.STREAM

Figure 12-2 MQInputNode basic properties

The second setting is in the Advanced panel, and is called Transaction Mode. This must be set to Yes, as shown in the figure below, so that all messages that pass through the flow are controlled under separate transactional contexts.


MQInput Node Properties - MQInput

Advanced

Transaction Mode: Yes
Order Mode: Default
Logical Order: ✓
All Messages Available: ☐
Match Message ID:
Match Correlation ID:
Convert: ☐
Convert Encoding:
Convert Coded Character Set ID:
Commit By Message Group: ☐

Figure 12-3 MQInputNode Advanced properties

Once the message flow has been created and configured, create a new Message Broker Archive file for use in deploying it to the broker. Add the flow to the new .bar file, as shown in Chapter 7, “Configuration and administration of WBI Event Broker” on page 103, and deploy the .bar file to the broker's default execution group. Check the Event Log as before, this time for BIP2056I and BIP4045I messages showing success.

12.3 JNDI configuration

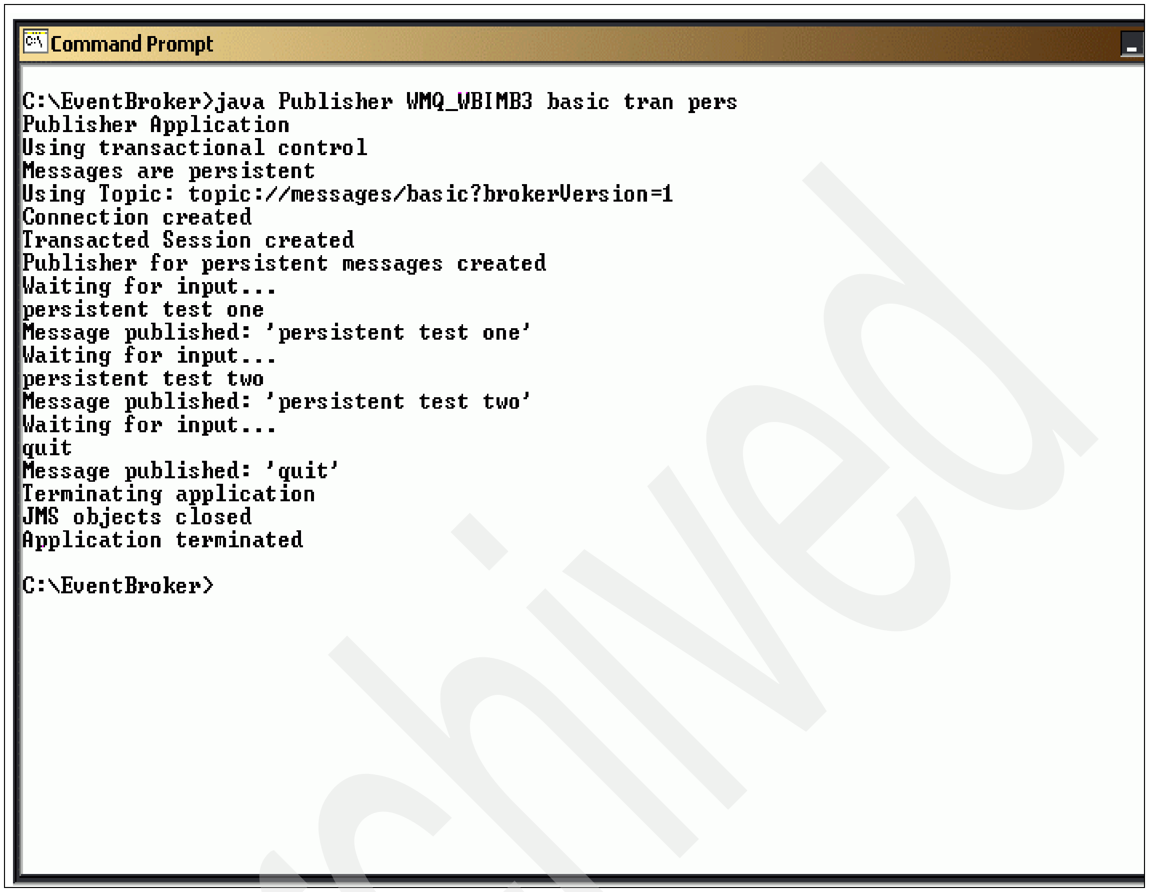
A TopicConnectionFactory reference is needed to match the message flows created above. Access the JNDI context as shown in Chapter 8, “Overview of basic scenario” on page 159, and execute the following command:

```
def tcf(WMQ_WBIMB3) brokerver(v2) qmanager(WBIMB3) brokerqmgr(WBIMB3)
clientid(clientid01)
```

The publishing and subscribing applications use this object to determine how to connect to the broker. In this case, the important points are qmanager and brokerqmgr, which correspond to the name of the queue manager underlying the broker. The clientid setting is required for JMS applications, and the **brokerver** setting tells the WebSphere MQ JMS implementation that the broker understands Pub/Sub information when held only in a WebSphere MQ RFH2 header structure.

12.4 Publishing and subscribing

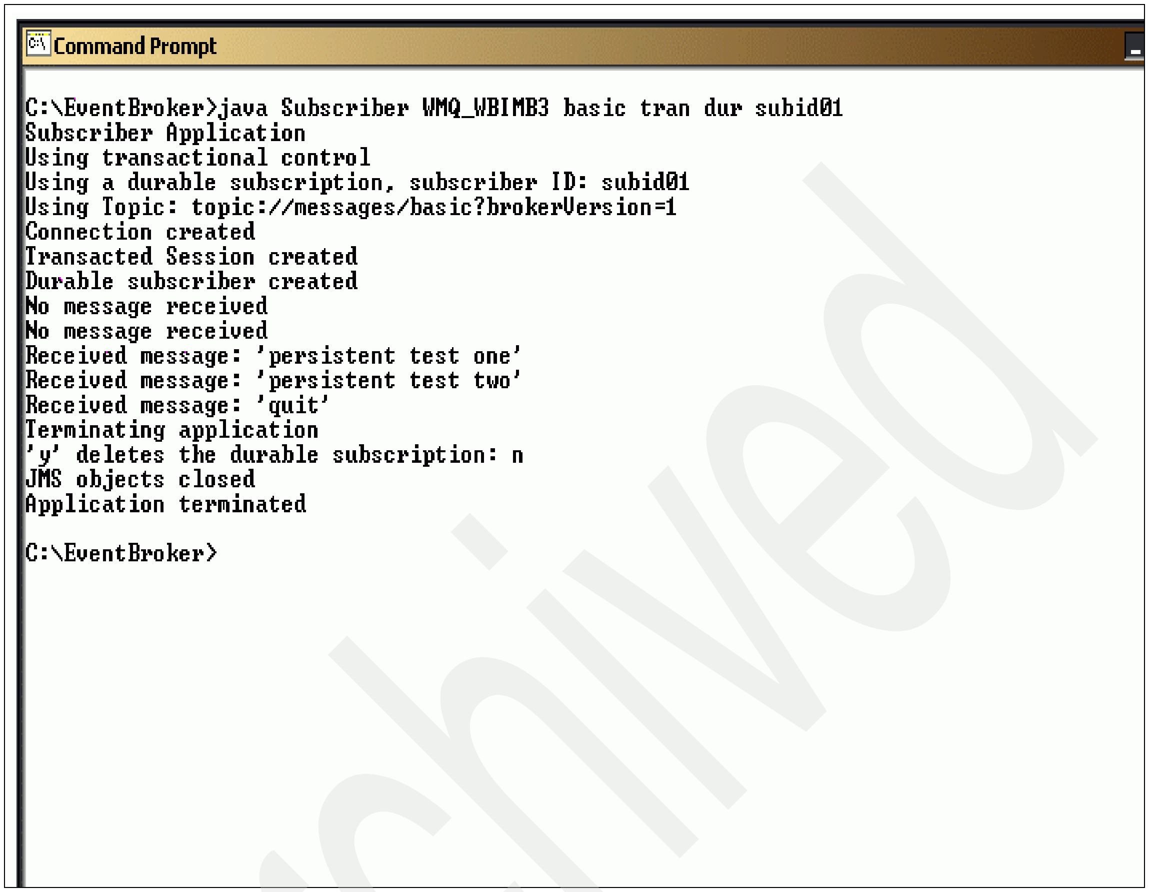
One publisher and one subscriber are sufficient to show that the scenario is correctly functioning. The publisher in this case connects to the broker, sends two publications and then sends a further publication instructing any subscribers to terminate. This is shown in 12.4, “Publishing and subscribing” on page 192 below. The subscriber must be started before the publications are sent (if not, there are no matching subscriptions for the publications and the broker therefore discards them). Both applications use transactions to control their message processing. This provides greater robustness (together with message persistence and durable subscriptions) at the expense of some performance.



```
C:\EventBroker>java Publisher WMQ_WBIMB3 basic tran pers
Publisher Application
Using transactional control
Messages are persistent
Using Topic: topic://messages/basic?brokerVersion=1
Connection created
Transacted Session created
Publisher for persistent messages created
Waiting for input...
persistent test one
Message published: 'persistent test one'
Waiting for input...
persistent test two
Message published: 'persistent test two'
Waiting for input...
quit
Message published: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 12-4 The publisher

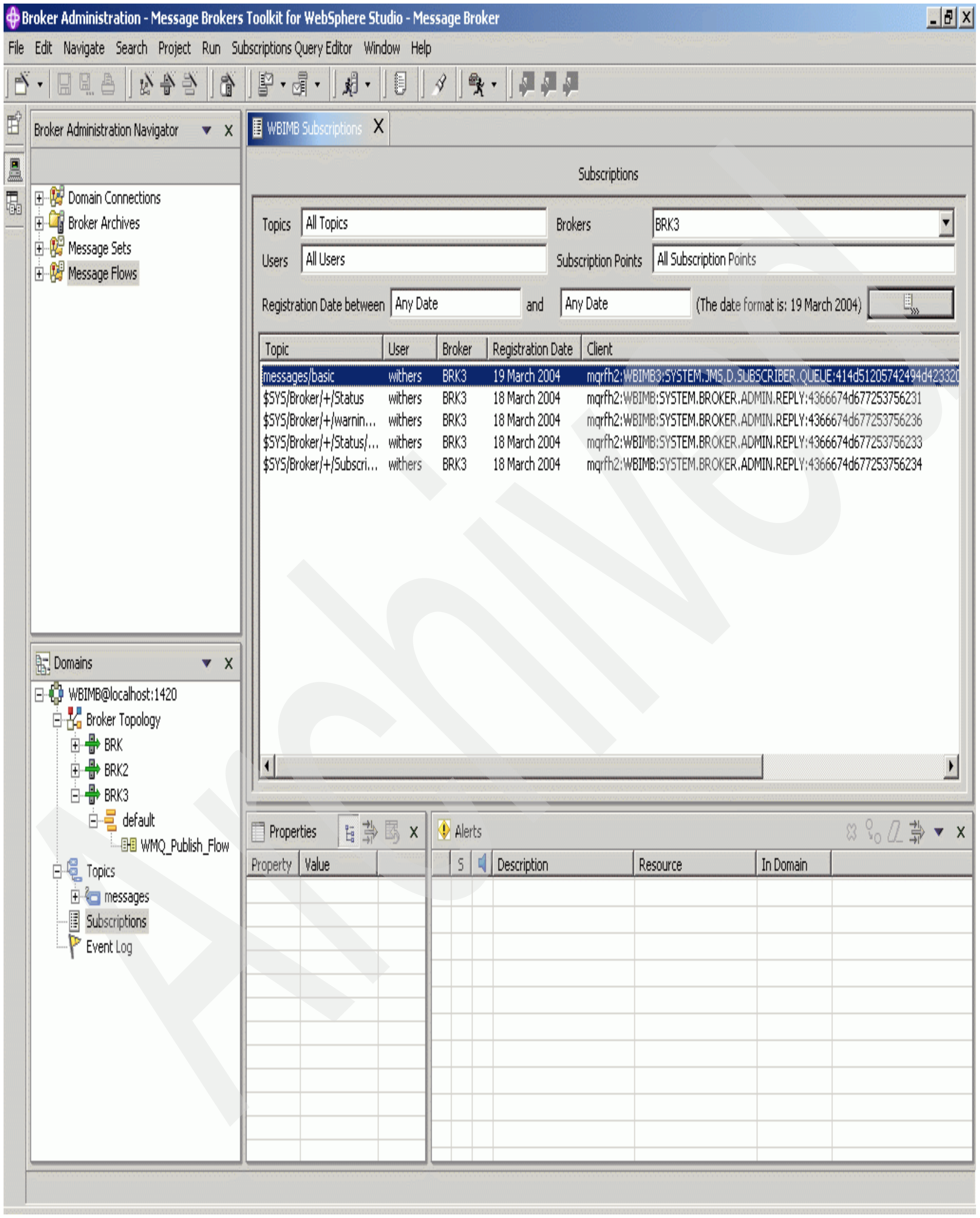
The subscriber receives the messages, but does not deregister its durable subscription when asked. This means that messages are held for it, even when it is not running. The subscriber application is shown in Figure 12-5 on page 194 below, and Figure 12-6 on page 195 shows the broker's subscription store still holding the subscription even though the application is not running.



Command Prompt

```
C:\EventBroker>java Subscriber WMQ_WBIMB3 basic tran dur subid01
Subscriber Application
Using transactional control
Using a durable subscription, subscriber ID: subid01
Using Topic: topic://messages/basic?brokerVersion=1
Connection created
Transacted Session created
Durable subscriber created
No message received
No message received
Received message: 'persistent test one'
Received message: 'persistent test two'
Received message: 'quit'
Terminating application
'y' deletes the durable subscription: n
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 12-5 The durable subscriber



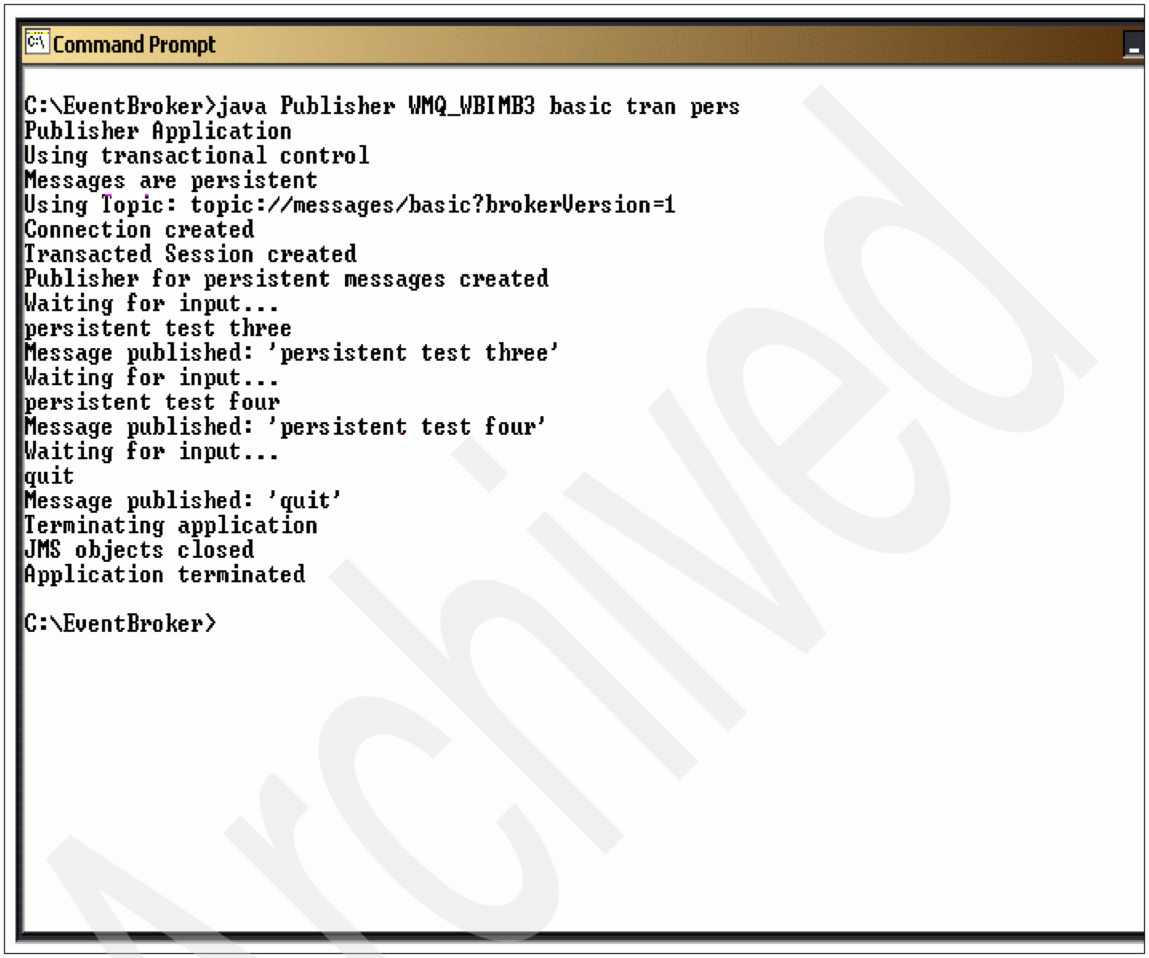
Subscriptions

Topics: All Topics
Brokers: BRK3
Users: All Users
Subscription Points: All Subscription Points
Registration Date between: Any Date and Any Date (The date format is: 19 March 2004)

Topic	User	Broker	Registration Date	Client
messages/basic	withers	BRK3	19 March 2004	mqrfh2:WBIMB3:SYSTEM.JMS.D.SUBSCRIBER.QUEUE:414d51205742494d423320
$SYS/Broker/+/Status	withers	BRK3	18 March 2004	mqrfh2:WBIMB:SYSTEM.BROKER.ADMIN.REPLY:4366674d677253756231
$SYS/Broker/+/warnin...	withers	BRK3	18 March 2004	mqrfh2:WBIMB:SYSTEM.BROKER.ADMIN.REPLY:4366674d677253756236
$SYS/Broker/+/Status/...	withers	BRK3	18 March 2004	mqrfh2:WBIMB:SYSTEM.BROKER.ADMIN.REPLY:4366674d677253756233
$SYS/Broker/+/Subscri...	withers	BRK3	18 March 2004	mqrfh2:WBIMB:SYSTEM.BROKER.ADMIN.REPLY:4366674d677253756234

Figure 12-6 The subscription store

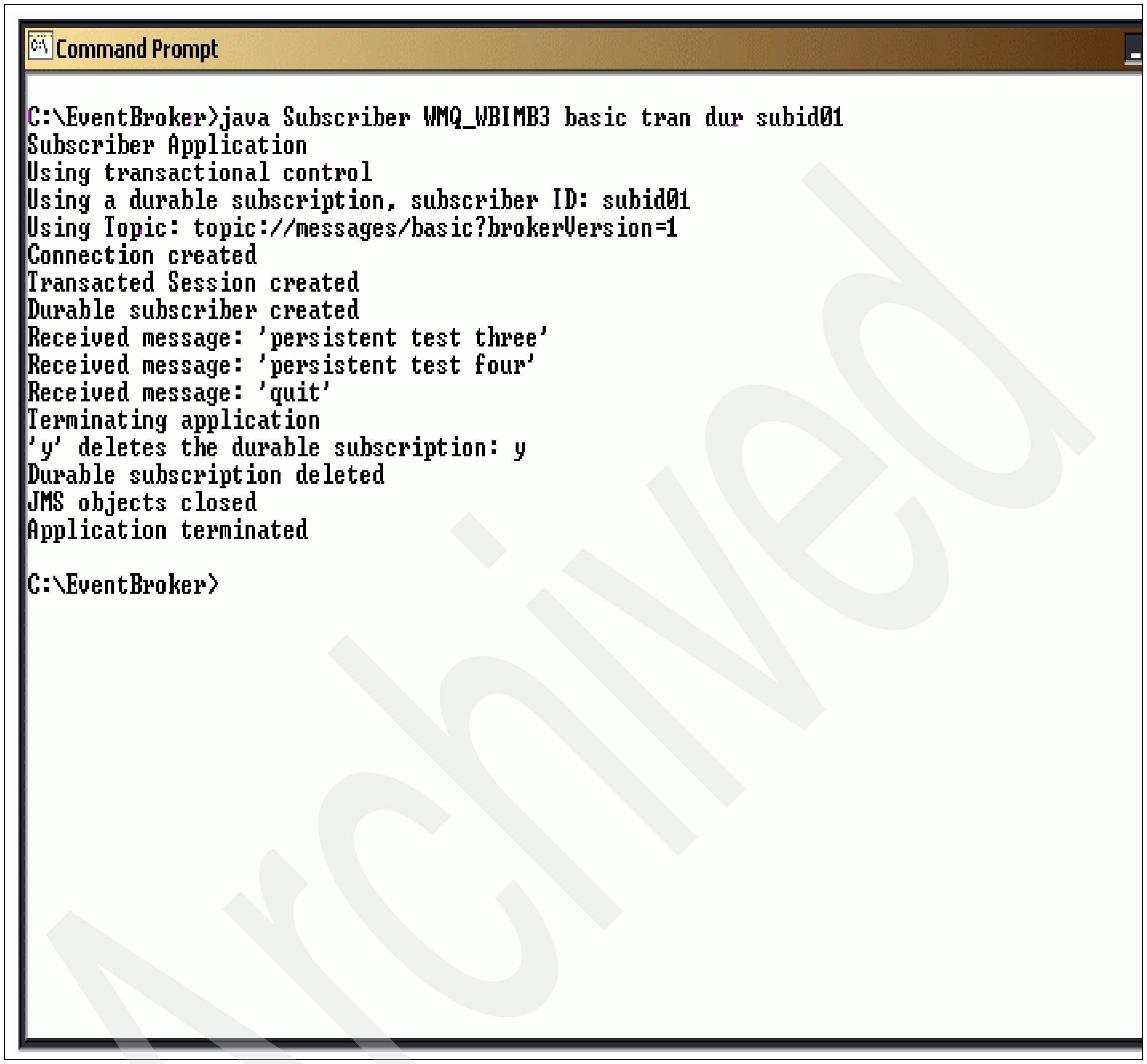
The publisher is restarted (this time without the subscriber running) and publishing two more messages before exiting, as shown in Figure 12-7.



```
C:\EventBroker>java Publisher WMQ_WBIMB3 basic tran pers
Publisher Application
Using transactional control
Messages are persistent
Using Topic: topic://messages/basic?brokerVersion=1
Connection created
Transacted Session created
Publisher for persistent messages created
Waiting for input...
persistent test three
Message published: 'persistent test three'
Waiting for input...
persistent test four
Message published: 'persistent test four'
Waiting for input...
quit
Message published: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 12-7 The Publisher

Finally, the subscriber is started again with the same durable subscription identifier. Even though the publisher published the last two messages while it was deactivated, they are still received because the durable subscription was not de-registered. This is shown in Figure 12-8 on page 197.



```
C:\EventBroker>java Subscriber WMQ_WBIMB3 basic tran dur subid01
Subscriber Application
Using transactional control
Using a durable subscription, subscriber ID: subid01
Using Topic: topic://messages/basic?brokerVersion=1
Connection created
Transacted Session created
Durable subscriber created
Received message: 'persistent test three'
Received message: 'persistent test four'
Received message: 'quit'
Terminating application
'y' deletes the durable subscription: y
Durable subscription deleted
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 12-8 The durable subscriber

13

Stream-crossing for single broker flexibility

This scenario shows how to configure a single broker configured to use both persistent and non-persistent messages in the Pub/Sub architecture, thus providing a higher degree of granularity when matching requirements to infrastructure design.

13.1 Preparation

The preparations necessary for this demonstration are contained in Chapter 7, “Configuration and administration of WBI Event Broker” on page 103, and Chapter 8, “Overview of basic scenario” on page 159. These are as follows:

- Create a queue manager for the broker and Configuration Manager to share.
- Create databases for the Configuration Manager and broker.
- Create a Configuration Manager and a broker.
- Create the topics required for the demonstrations in the Broker Toolkit.
- Create a JNDI context and store entries for the topics.

13.2 Broker configuration

The broker and its underlying queue manager are configured exactly as in “Broker configuration” on page 188, and the message flows used are all those detailed in “Message flows” on page 166 and “Message flows” on page 188. The only difference in this case is the port settings of the two IP flows, which are set to 2007 and 2008, respectively, at deployment time.

13.3 JNDI configuration

TopicConnectionFactory references are needed to match the message flows used above. Access the JNDI context as shown in Chapter 8, “Overview of basic scenario” on page 159, and execute the following commands:

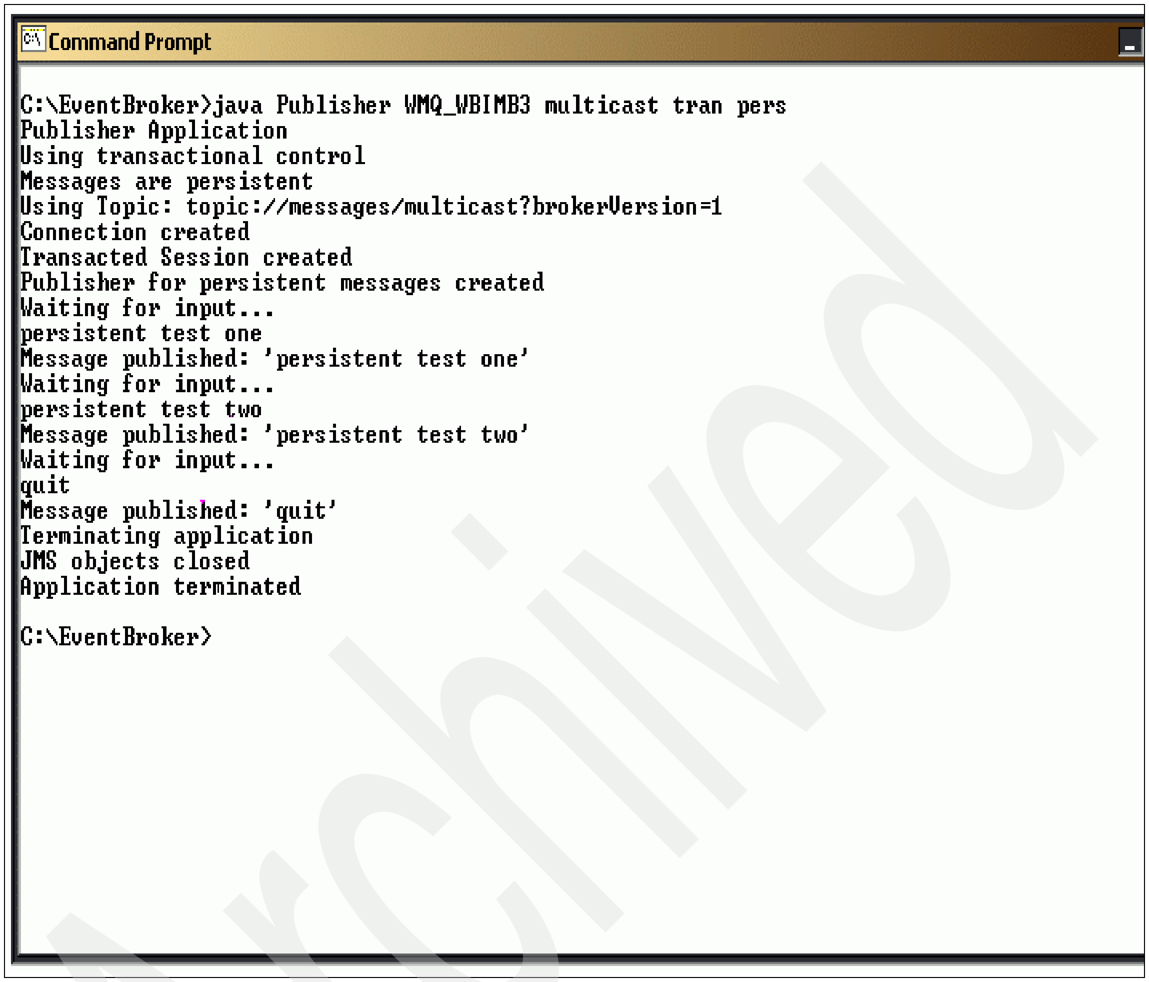
```
def tcf(WMQ_WBIMB3) brokerver(v2) qmanager(WBIMB3) brokerqmgr(WBIMB3)
clientid(clientid01)

def tcf(IP_2007) brokerver(v2) transport(direct) hostname(localhost) port(2007)

def tcf(IP_2008) brokerver(v2) transport(direct) hostname(localhost) port(2008)

def tcf(Multicast_2008) brokerver(v2) transport(direct) hostname(localhost)
port(2008) multicast(reliable)
```

The publishing and subscribing applications use these objects to determine how to connect to the broker. The settings above are detailed in “JNDI configuration” on page 170 and “JNDI configuration” on page 191.

13.4 Publishing and subscribing

Two publishers and three subscribers are required to show that the scenario is correctly functioning. All of the applications are using the multicast-enabled topic, but only the multicast subscriber is using the multicast transport. This scenario demonstrates that no matter which transport is being used, the matching of messages to subscriptions is done at a general level. This also shows that the required qualities of service for different parts of the architecture can be set without constraining all components to the slowest settings.

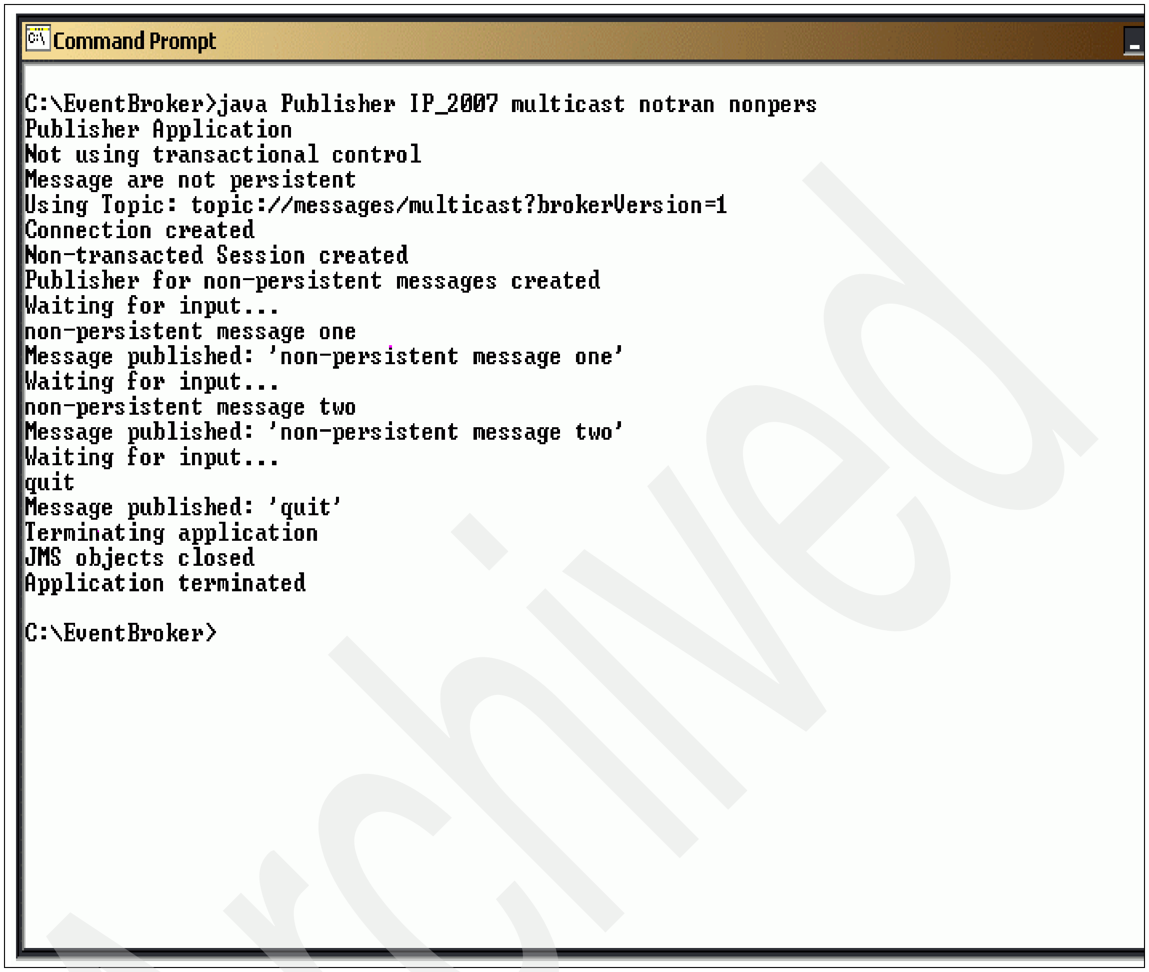
The first publisher connects to the broker using the WebSphere MQ queue-based transport and sends two publications. At least one subscriber must be started before the publications are sent (if not, there are no matching subscriptions for the publications and the broker therefore discards them). This is shown in Figure 13-1 on page 202 (as is a final *quit* message, but this was sent after all other applications had terminated).



```
C:\EventBroker>java Publisher WMQ_WBIMB3 multicast tran pers
Publisher Application
Using transactional control
Messages are persistent
Using Topic: topic://messages/multicast?brokerVersion=1
Connection created
Transacted Session created
Publisher for persistent messages created
Waiting for input...
persistent test one
Message published: 'persistent test one'
Waiting for input...
persistent test two
Message published: 'persistent test two'
Waiting for input...
quit
Message published: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 13-1 The publisher using WebSphere MQ queues

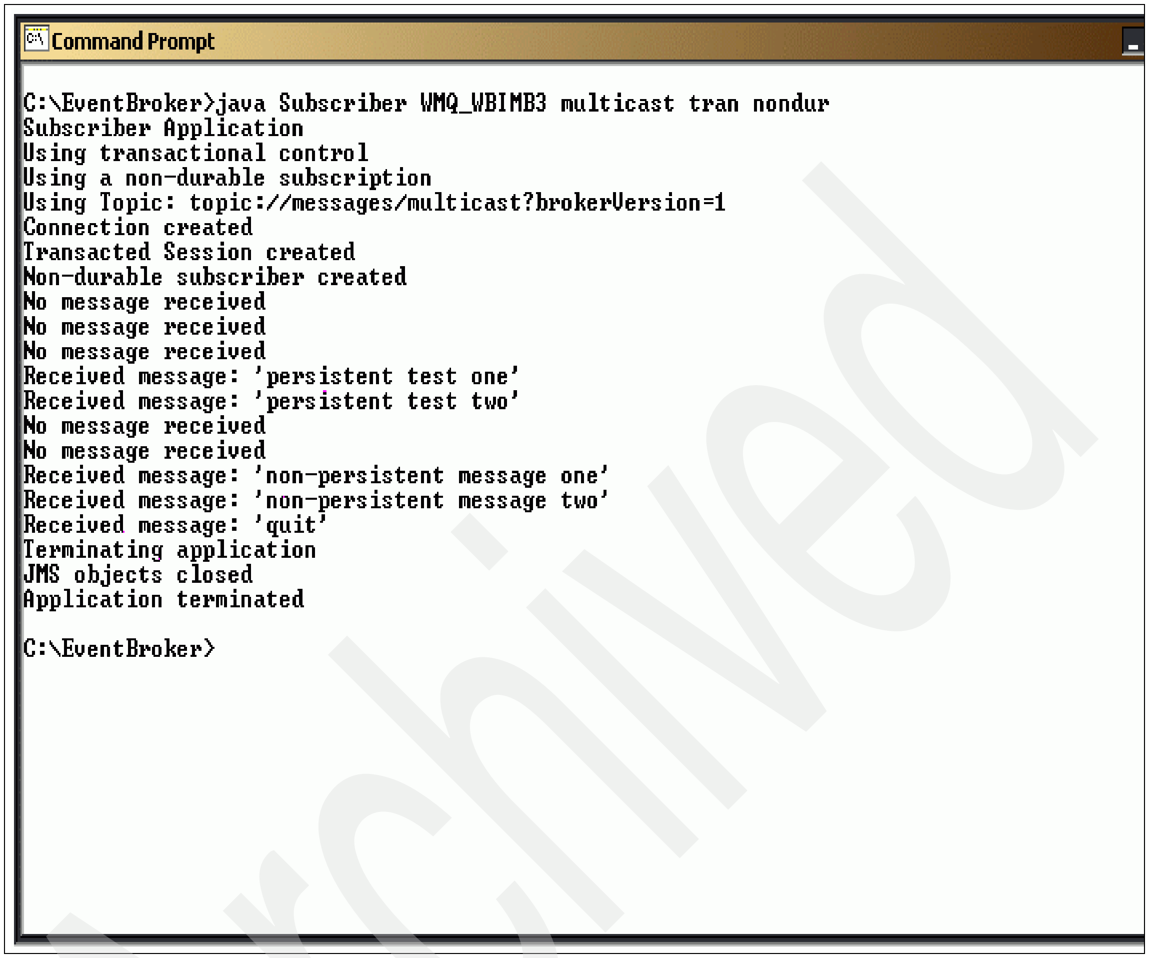
The second publisher connects to the broker using the WebSphere MQ real-time transport and sends two publications, before sending another publication instructing the subscribers to terminate. The same comments about subscriber registration above apply in this case as well. This application is shown in Figure 13-2 on page 203.



```
C:\EventBroker>java Publisher IP_2007 multicast notran nonpers
Publisher Application
Not using transactional control
Message are not persistent
Using Topic: topic://messages/multicast?brokerVersion=1
Connection created
Non-transacted Session created
Publisher for non-persistent messages created
Waiting for input...
non-persistent message one
Message published: 'non-persistent message one'
Waiting for input...
non-persistent message two
Message published: 'non-persistent message two'
Waiting for input...
quit
Message published: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 13-2 The publisher using WebSphere MQ real-time transport

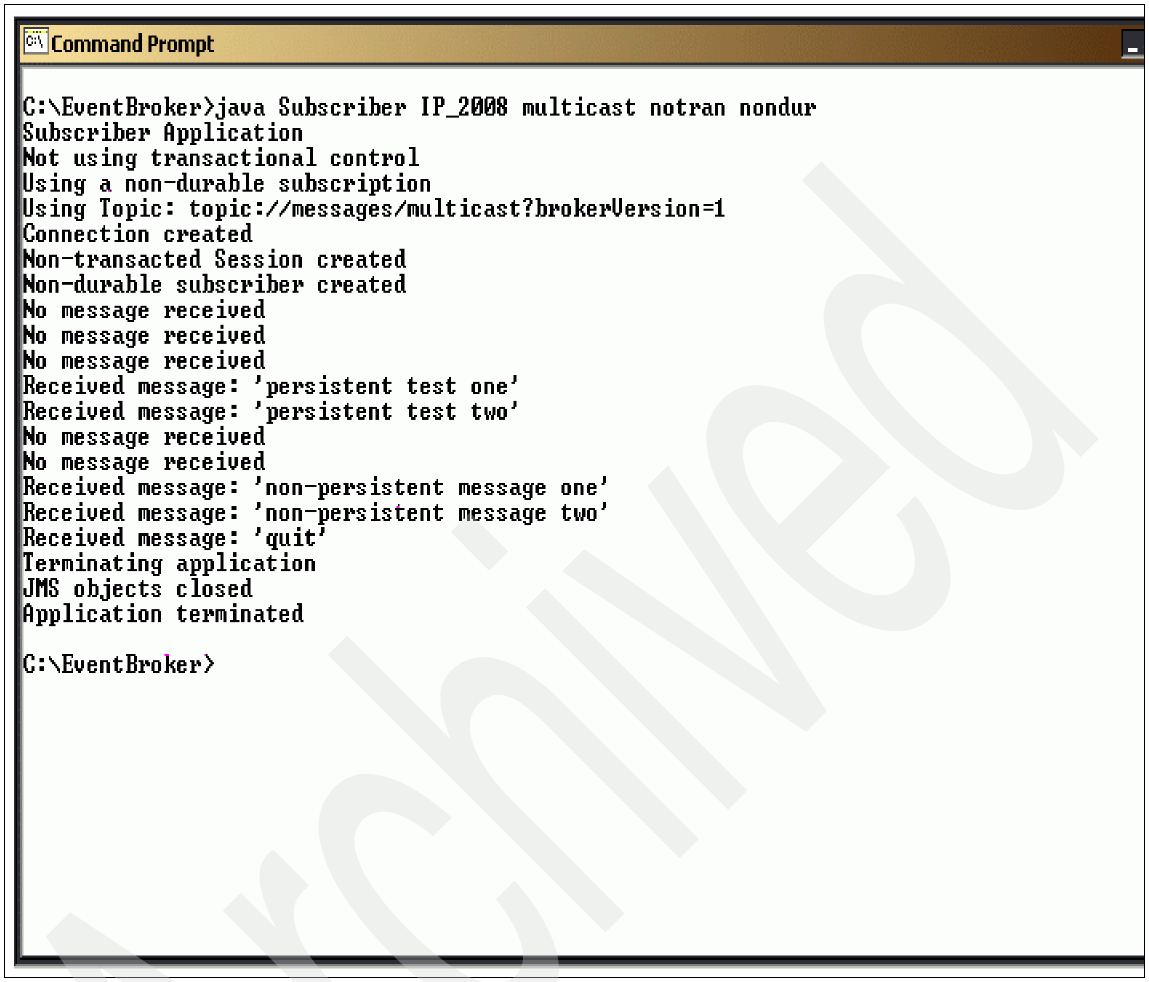
The first subscriber uses the WebSphere MQ queue-based transport (without a durable subscription this time). It receives all four messages from both publishers before terminating, as shown in Figure 13-3 on page 204.



```
C:\EventBroker>java Subscriber WMQ_WBIMB3 multicast tran nondur
Subscriber Application
Using transactional control
Using a non-durable subscription
Using Topic: topic://messages/multicast?brokerVersion=1
Connection created
Transacted Session created
Non-durable subscriber created
No message received
No message received
No message received
Received message: 'persistent test one'
Received message: 'persistent test two'
No message received
No message received
Received message: 'non-persistent message one'
Received message: 'non-persistent message two'
Received message: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 13-3 The subscriber using WebSphere MQ queues

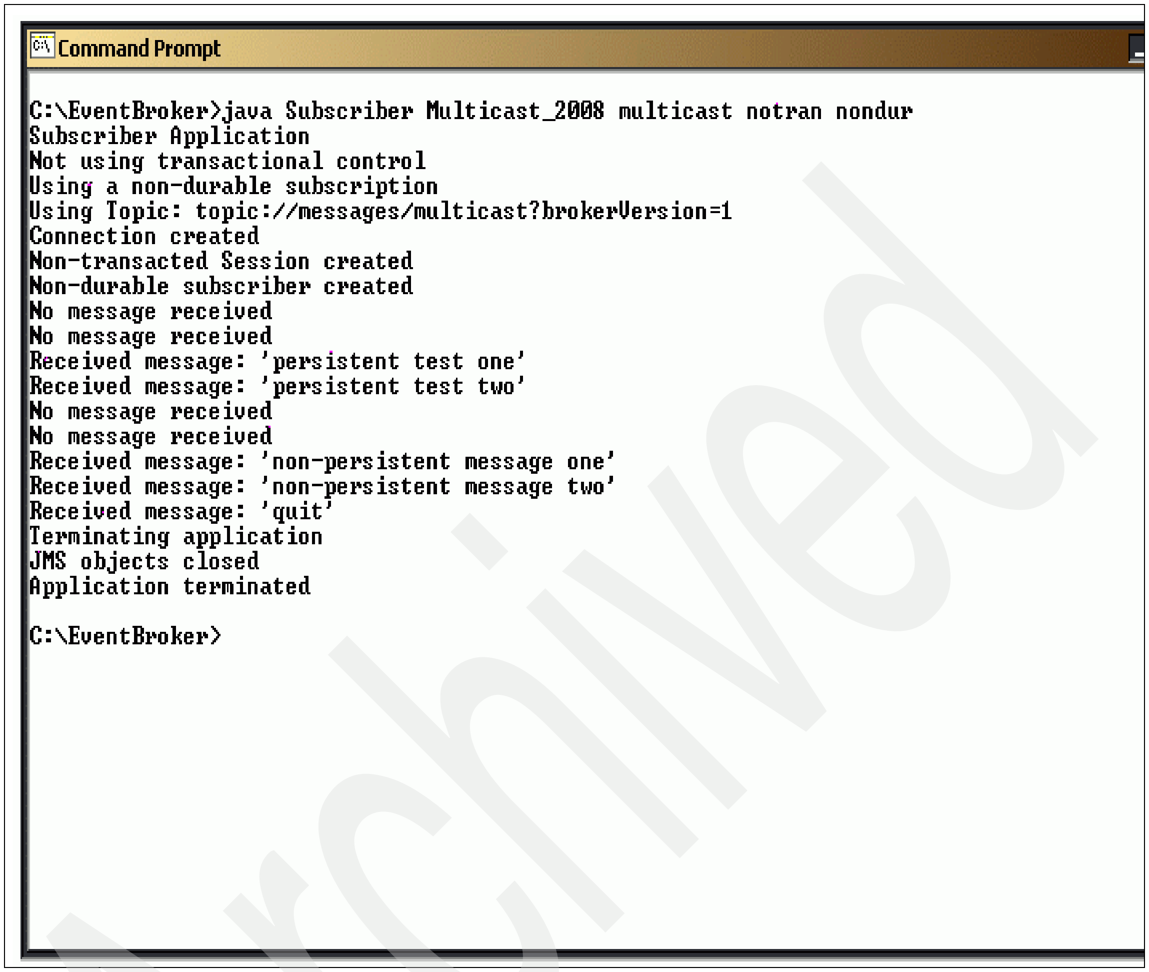
The second subscriber uses the WebSphere MQ real-time transport. It receives all four messages from both publishers before terminating, as shown in Figure 13-4 on page 205 below.



```
C:\EventBroker>java Subscriber IP_2008 multicast notran nondur
Subscriber Application
Not using transactional control
Using a non-durable subscription
Using Topic: topic://messages/multicast?brokerVersion=1
Connection created
Non-transacted Session created
Non-durable subscriber created
No message received
No message received
No message received
Received message: 'persistent test one'
Received message: 'persistent test two'
No message received
No message received
Received message: 'non-persistent message one'
Received message: 'non-persistent message two'
Received message: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 13-4 The subscriber using WebSphere MQ real-time transport

The third subscriber uses the WebSphere MQ multicast transport. It receives all four messages from both publishers before terminating, as shown in Figure 13-5 on page 206 below.



```
C:\EventBroker>java Subscriber Multicast_2008 multicast notran nondur
Subscriber Application
Not using transactional control
Using a non-durable subscription
Using Topic: topic://messages/multicast?brokerVersion=1
Connection created
Non-transacted Session created
Non-durable subscriber created
No message received
No message received
Received message: 'persistent test one'
Received message: 'persistent test two'
No message received
No message received
Received message: 'non-persistent message one'
Received message: 'non-persistent message two'
Received message: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 13-5 The subscriber using WebSphere MQ multicast transport

All the subscribers use different TopicConnectionFactory objects, different transports and different message flows. The WebSphere MQ queue-based applications use transactional control, whereas the other applications do not. Despite this complete mismatching of application resources and settings, WebSphere Business Integration Event Broker has successfully stream-crossed and ensured that all subscribers receive any messages published to the topic they have subscribed to.

14

Cloned brokers for high availability

This scenario shows how to configure a cloned broker system designed to mitigate broker failures and thus provide a greater degree of system reliability.

14.1 Preparation

The preparations necessary for this demonstration are contained in Chapter 7, “Configuration and administration of WBI Event Broker” on page 103, and Chapter 8, “Overview of basic scenario” on page 159. These are as follows:

- Create a queue manager for the broker and Configuration Manager to share.
- Create databases for the Configuration Manager and broker.
- Create a Configuration Manager and a broker.
- Create the topics required for the demonstrations in the Broker Toolkit.
- Create a JNDI context and store entries for the topics.

14.2 Broker configuration

First, another broker must be created in order to join with the first broker in a cloned system. This needs a separate queue manager and database created before the new broker can be created (as shown in Chapter 7, “Configuration and administration of WBI Event Broker” on page 103). The two queue managers need to be connected in the same way as described in “Broker configuration” on page 180.

14.2.1 Broker topology

Add the two brokers to the broker topology as described in “Broker topology” on page 181, but without adding the collective. Save and deploy the configuration, and then clone the brokers from the command line, using the following commands:

```
mqsichangeproperties BRK3 -e default -o DynamicSubscriptionEngine -n
clonedPubSubBrokerList -v \"BRK4,WBIMB4\"

mqsichangeproperties BRK4 -e default -o DynamicSubscriptionEngine -n
clonedPubSubBrokerList -v \"BRK3,WBIMB3\"
```

Both brokers need the name of the other broker they are to clone with, and the name of its underlying queue manager. The brokers need to be restarted to detect this change.

14.2.2 Message flows

The message flow used in this scenario is the same as that specified in “Message flows” on page 188. Exactly the same flow must be deployed to both brokers (that is, use the same .bar file). There is no conflict in this case because each broker's queue manager has a queue called SYSTEM.BROKER.DEFAULT.STREAM (unlike ports, which are scoped at a machine level, so brokers on the same machine must use different ports to avoid conflicts).

14.3 JNDI configuration

TopicConnectionFactory references are needed to match the message flows deployed to the two brokers above. Access the JNDI context as shown in Chapter 8, “Overview of basic scenario” on page 159, and execute the following commands:

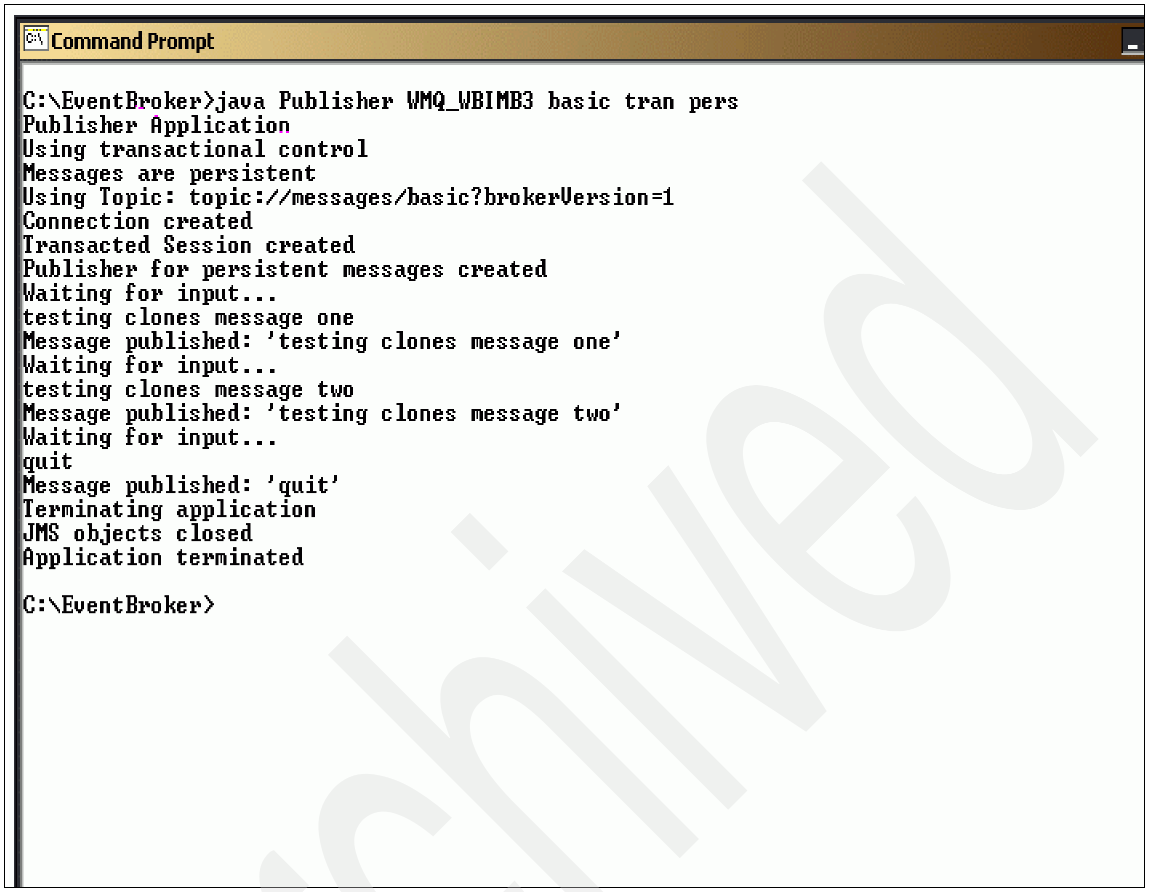
```
def tcf(WMQ_WBIMB3) brokerver(v2) qmanager(WBIMB3) brokerqmgr(WBIMB3)
clientid(clientid01) clonesupp(enabled)

def tcf(WMQ_WBIMB4) brokerver(v2) qmanager(WBIMB4) brokerqmgr(WBIMB4)
clientid(clientid02) clonesupp(enabled)
```

The publishing and subscribing applications use these objects to determine how to connect to the broker. Most of the settings above are detailed in “JNDI configuration” on page 191, and *clonesupp* is set to *enabled* to instruct the JMS to support cloned brokers.

14.4 Publishing and subscribing

Two publishers and one subscriber are required to show that the scenario is correctly functioning. This scenario demonstrates that no matter from which broker a publication originates, underlying WebSphere MQ distributed queuing allows the message to reach the subscribing application, without having to pass through the broker it directly subscribed to.

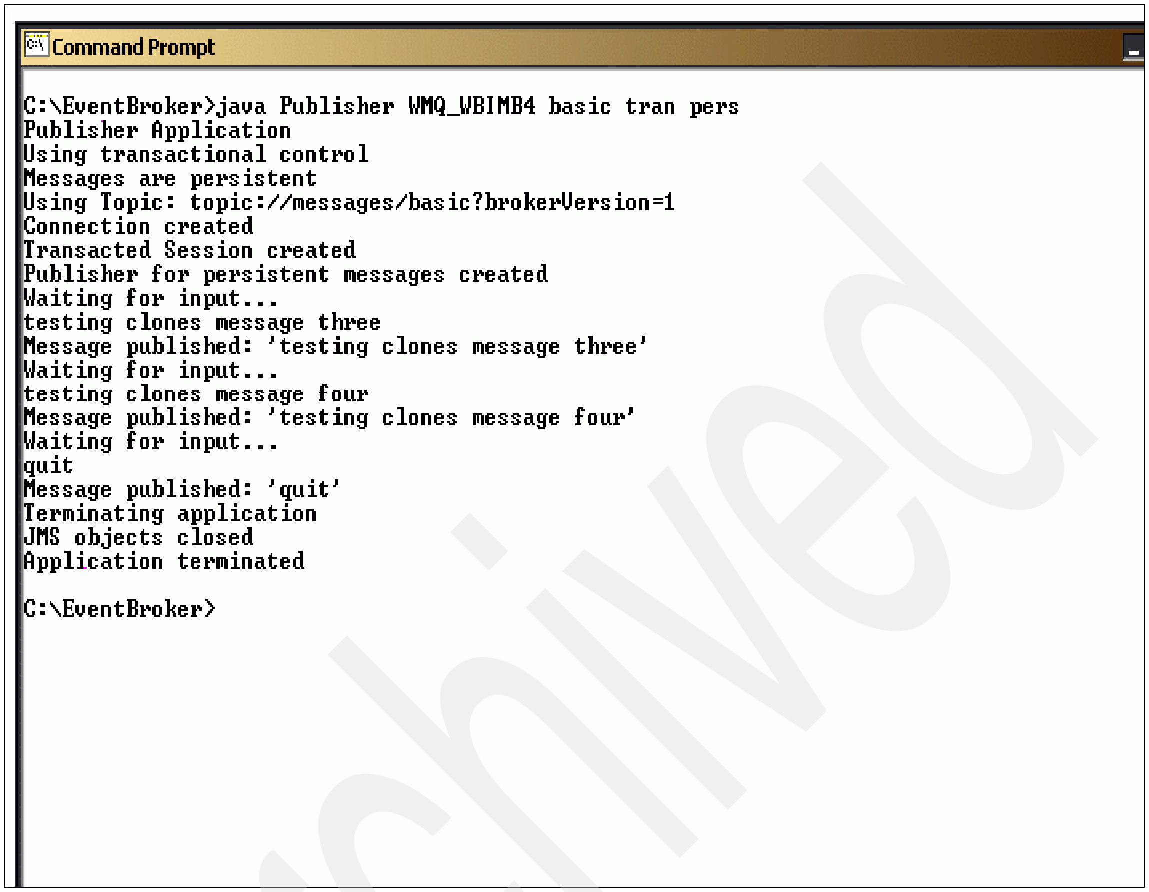
The first publisher connects to the first broker and sends two publications. The subscriber must be started before the publications are sent (if not, there are no matching subscriptions for the publications and the broker therefore discards them). This is shown in Figure 14-1 on page 210 below (as is a final *quit* message, but this is sent after all other applications have terminated).



```
C:\EventBroker>java Publisher WMQ_WBIMB3 basic tran pers
Publisher Application
Using transactional control
Messages are persistent
Using Topic: topic://messages/basic?brokerVersion=1
Connection created
Transacted Session created
Publisher for persistent messages created
Waiting for input...
testing clones message one
Message published: 'testing clones message one'
Waiting for input...
testing clones message two
Message published: 'testing clones message two'
Waiting for input...
quit
Message published: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 14-1 The publisher on broker 1

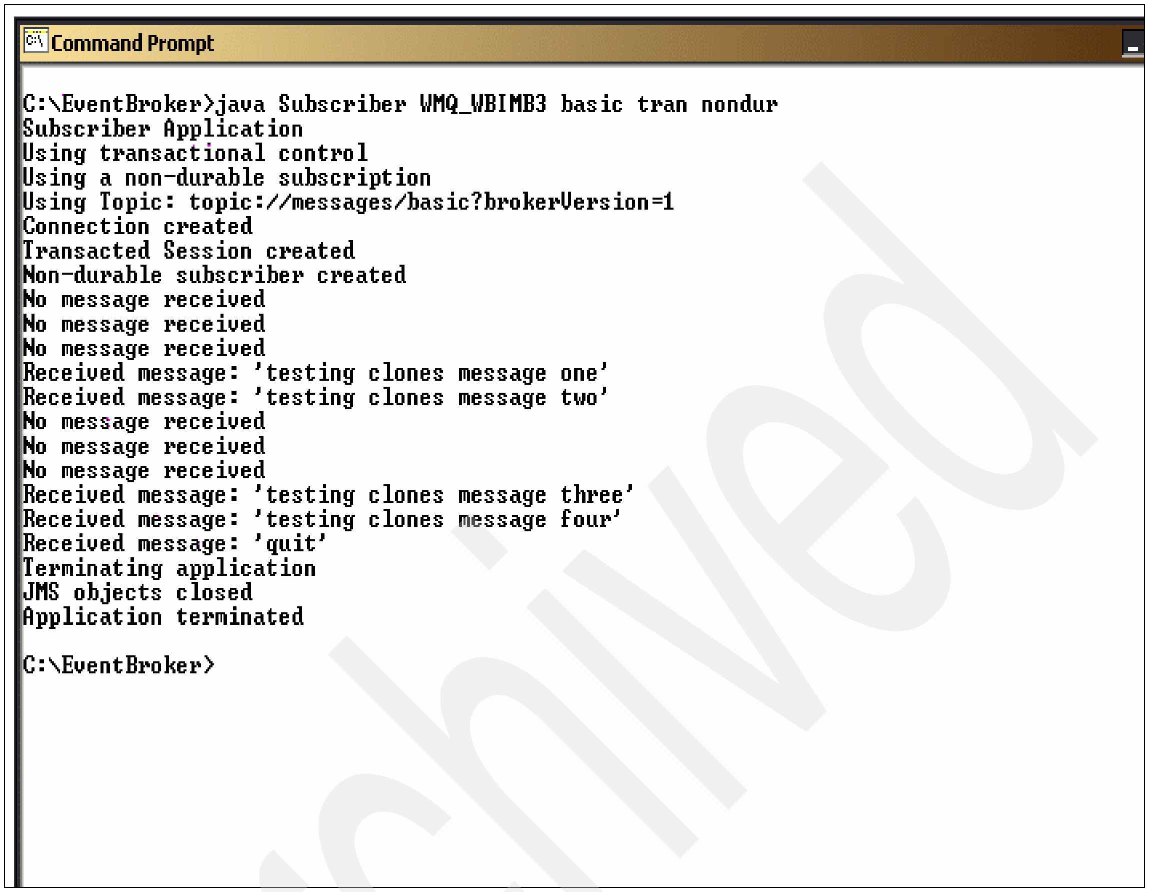
The second publisher connects to the second broker and sends two publications, before sending another publication instructing the subscribers to terminate. The same comments about subscriber registration above apply in this case as well. This application is shown in Figure 14-2 on page 211 below.



```
C:\EventBroker>java Publisher WMQ_WBIMB4 basic tran pers
Publisher Application
Using transactional control
Messages are persistent
Using Topic: topic://messages/basic?brokerVersion=1
Connection created
Transacted Session created
Publisher for persistent messages created
Waiting for input...
testing clones message three
Message published: 'testing clones message three'
Waiting for input...
testing clones message four
Message published: 'testing clones message four'
Waiting for input...
quit
Message published: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 14-2 The publisher on broker 2

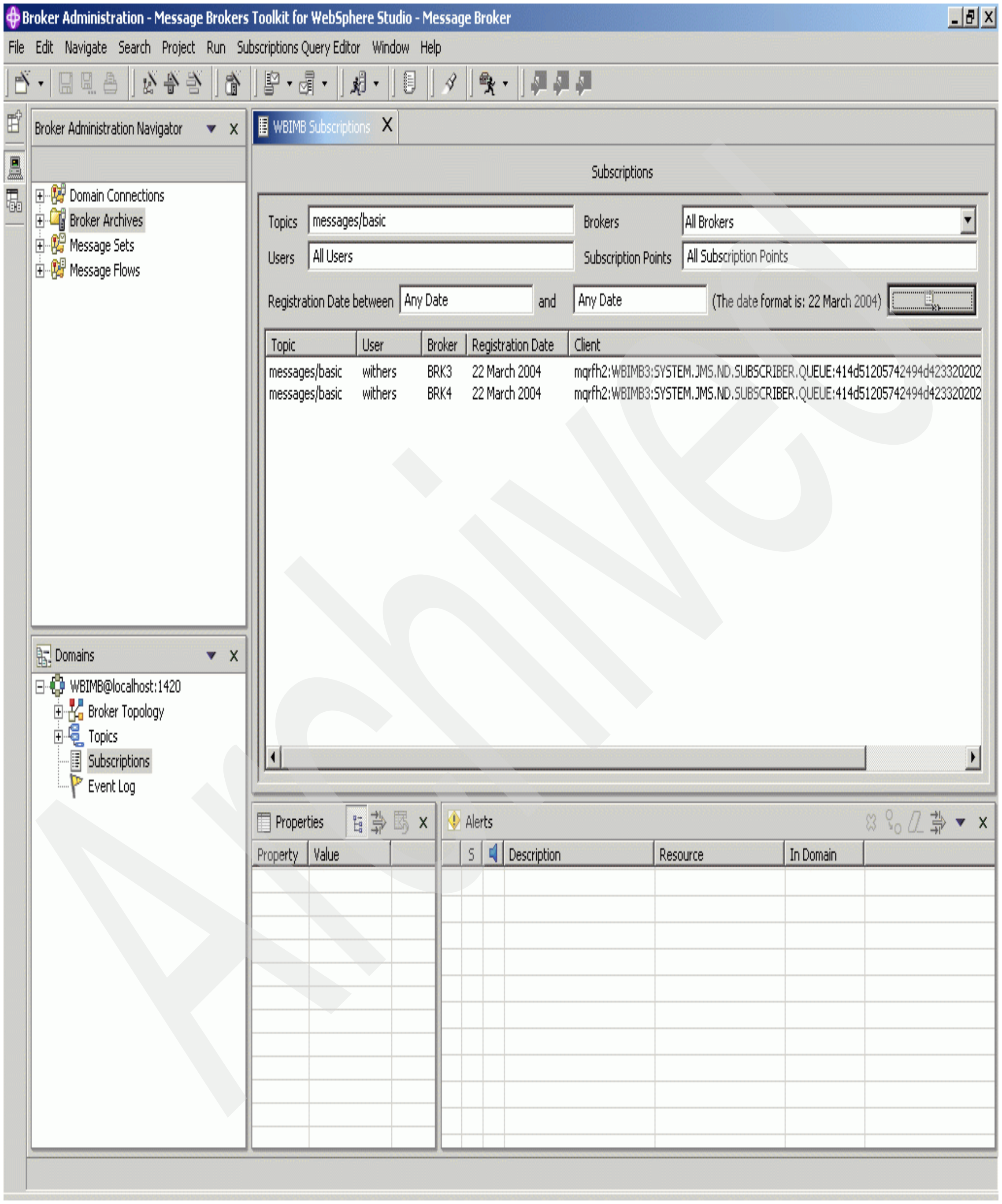
The subscriber uses the WebSphere MQ queue-based transport (without a durable subscription this time). It receives all four messages from both publishers before terminating, as shown in Figure 14-3 on page 212.



```
C:\EventBroker>java Subscriber WMQ_WBIMB3 basic tran nondur
Subscriber Application
Using transactional control
Using a non-durable subscription
Using Topic: topic://messages/basic?brokerVersion=1
Connection created
Transacted Session created
Non-durable subscriber created
No message received
No message received
No message received
Received message: 'testing clones message one'
Received message: 'testing clones message two'
No message received
No message received
No message received
Received message: 'testing clones message three'
Received message: 'testing clones message four'
Received message: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 14-3 The subscriber

The subscriber has a single input queue owned by the first broker's queue manager. This is where published messages are put by *both* brokers. This is accomplished by the cloning mechanism, where the brokers each maintain a subscription registration for the one subscriber. The registration information is the same on each broker to allow for the remote queue manager where the subscription queue is actually held. This is shown in Figure 14-4 on page 213 below, which shows the subscriptions to the topic in question on all brokers in the topology.



Subscriptions

Topics: messages/basic
Brokers: All Brokers
Users: All Users
Subscription Points: All Subscription Points
Registration Date between: Any Date and Any Date (The date format is: 22 March 2004)

Topic	User	Broker	Registration Date	Client
messages/basic	withers	BRK3	22 March 2004	mqrfh2:WBIMB3:SYSTEM.JMS.ND.SUBSCRIBER.QUEUE:414d51205742494d423320202
messages/basic	withers	BRK4	22 March 2004	mqrfh2:WBIMB3:SYSTEM.JMS.ND.SUBSCRIBER.QUEUE:414d51205742494d423320202

Figure 14-4 Cloned subscriptions

The subscription data (in the Client column) can be seen to be queue manager WBIMB3 for both brokers BRK3 and BRK4.

15

Using multicast in a broker collective

This scenario shows how to configure a broker collective to use the WebSphere MQ Reliable Multicast Messaging (RMM) transport within the boundaries of the product design.

15.1 Preparation

The preparations necessary for this demonstration are contained in Chapter 7, “Configuration and administration of WBI Event Broker” on page 103, and Chapter 8, “Overview of basic scenario” on page 159. These are as follows:

- Create a queue manager for the broker and Configuration Manager to share.
- Create databases for the Configuration Manager and broker.
- Create a Configuration Manager and a broker.
- Create the topics required for the demonstrations in the Broker Toolkit.
- Create a JNDI context and store entries for the topics.

15.2 Broker configuration

Complete the broker collective configuration as detailed in “Broker configuration” on page 180, “Broker topology” on page 181 and “Message flows” on page 183. Once this has been completed and deployed successfully, the brokers need to be configured for multicast. The purpose of the steps below is to ensure that the brokers do not clash with each other in the allocation of multicast group addresses.

Open the broker topology, and right-click the first broker. Select **Properties** and then select the **Multicast** panel. First, check the **Multicast Enabled** box. The three important multicast settings for a multiple broker configuration are Min Address, Max Address and Data Port. For the first broker, the only one that needs changing is Max Address, which needs to be reduced to 231.255.255.255. This reduces the range of multicast group addresses available to the first broker down to approximately half the multicast range.

Repeat the process for the second broker, but this time the parameters that require alteration are Min Address and Data Port. These must be changed to 232.0.0.0 and 34344, respectively, ensuring that the multicast group address space is completely partitioned across the two brokers, and they do not clash for control of one port. If the brokers are on separate machines, the port can be left to the default of 34343.

Save and deploy the new configuration, restart the brokers and manually repeat the deployment of the Topics Hierarchy (right-click **Topics**, select **Deploy Topics Configuration -> Complete**). This step is necessary to ensure that the multicast-enabled topic is deployed to the brokers when they are configured for multicast.

15.3 JNDI configuration

TopicConnectionFactory references are needed to match the message flows used above. Access the JNDI context as shown in Chapter 8, “Overview of basic scenario” on page 159, and execute the following commands:

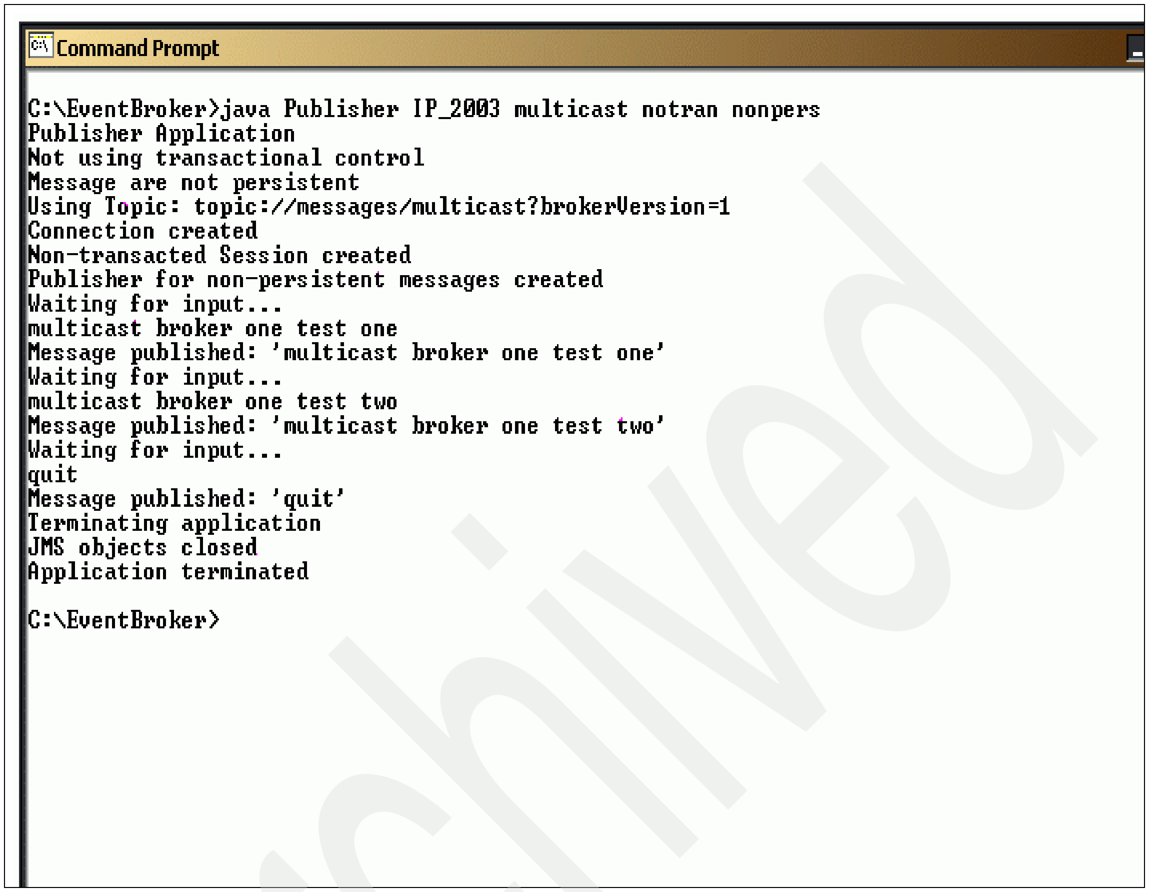
```
def tcf(IP_2003) brokerver(v2) transport(direct) hostname(localhost) port(2003)

def tcf(IP_2005) brokerver(v2) transport(direct) hostname(localhost) port(2005)

def tcf(IP_2006) brokerver(v2) transport(direct) hostname(localhost) port(2006)

def tcf(Multicast_2006) brokerver(v2) transport(direct) hostname(localhost)
port(2006) multicast(reliable)
```

The publishing and subscribing applications use these objects to determine how to connect to the broker. The settings above are detailed in “JNDI configuration” on page 170 and “JNDI configuration” on page 175.

15.4 Publishing and subscribing

Two publishers and two subscribers are sufficient to show that the scenario is correctly functioning. This scenario demonstrates that even though publications are shared across a collective, they are only delivered to multicast subscribers if they originate on the same broker. This is because, despite the possibility of partitioning the multicast group space, the topic configuration cannot be partitioned across different brokers in the same broker domain. This limitation, when combined with the operation of a broker collective, means that subscribers may receive multiple copies of the same message.

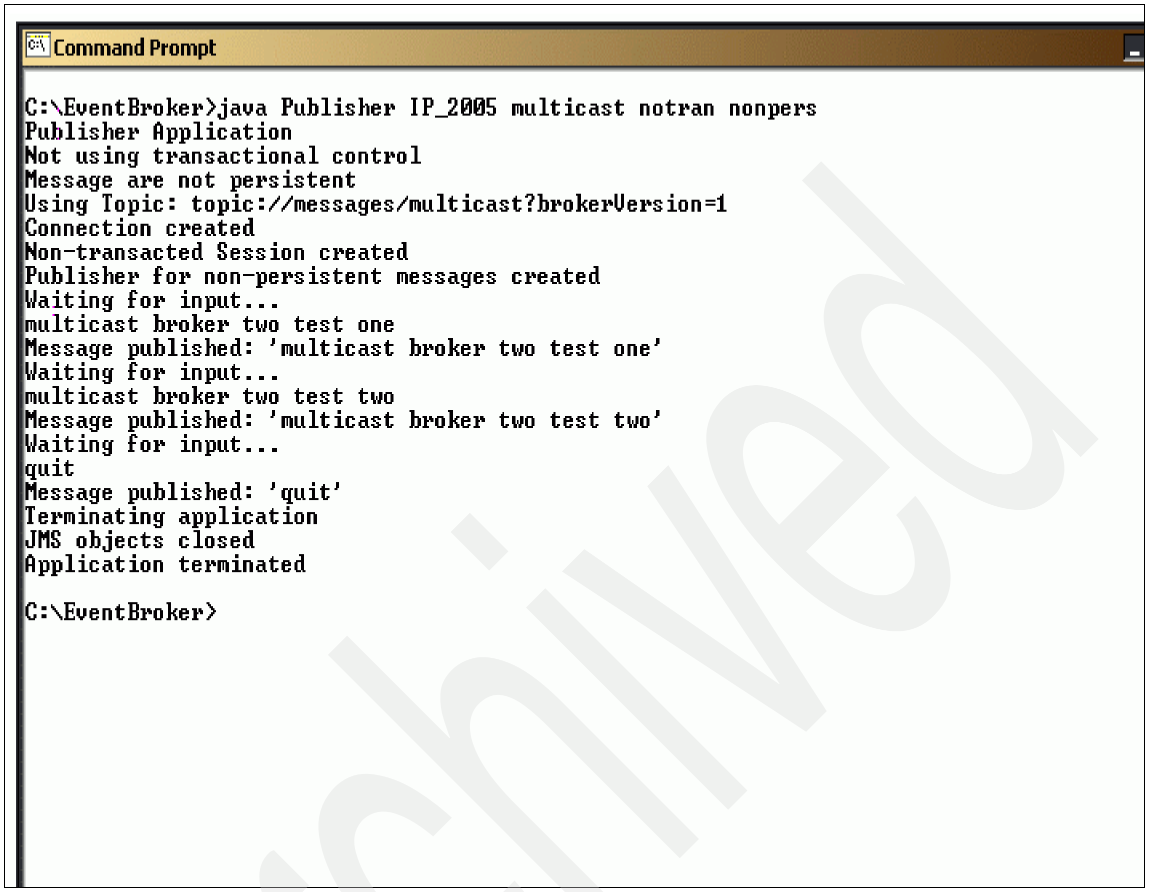
The first publisher connects to the first broker in the collective and sends two publications. The subscribers must be started before the publications are sent (if not, there are no matching subscriptions for the publications and the broker therefore discards them). This is shown in Figure 15-1 on page 218 below (as is a final quit message, but this was sent after all other applications had terminated).



```
C:\EventBroker>java Publisher IP_2003 multicast notran nonpers
Publisher Application
Not using transactional control
Message are not persistent
Using Topic: topic://messages/multicast?brokerVersion=1
Connection created
Non-transacted Session created
Publisher for non-persistent messages created
Waiting for input...
multicast broker one test one
Message published: 'multicast broker one test one'
Waiting for input...
multicast broker one test two
Message published: 'multicast broker one test two'
Waiting for input...
quit
Message published: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 15-1 The publisher on broker 1

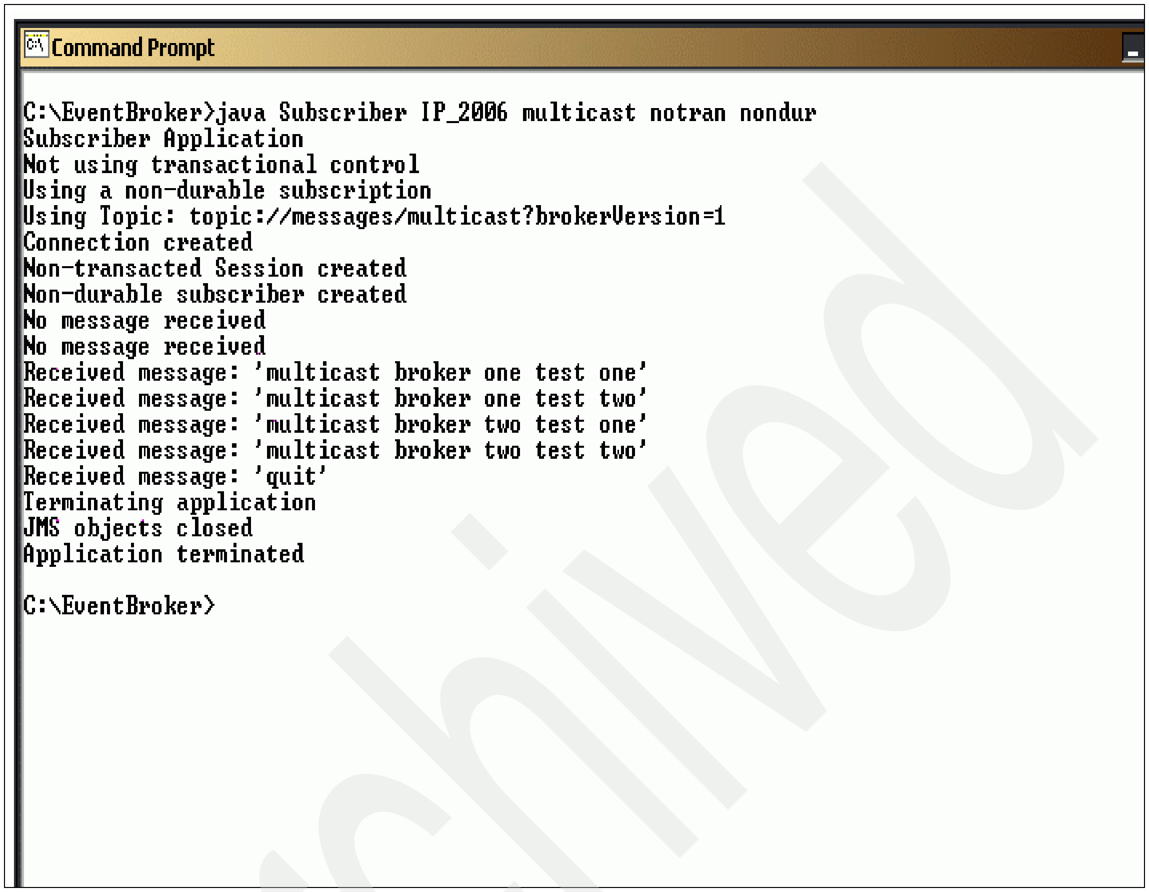
The second publisher connects to the second broker in the collective and sends two publications, before sending another publication instructing the subscribers to terminate. The same comments about subscriber registration above apply in this case as well. This application is shown in Figure 15-2 on page 219.



```
C:\EventBroker>java Publisher IP_2005 multicast notran nonpers
Publisher Application
Not using transactional control
Message are not persistent
Using Topic: topic://messages/multicast?brokerVersion=1
Connection created
Non-transacted Session created
Publisher for non-persistent messages created
Waiting for input...
multicast broker two test one
Message published: 'multicast broker two test one'
Waiting for input...
multicast broker two test two
Message published: 'multicast broker two test two'
Waiting for input...
quit
Message published: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 15-2 The publisher on broker 2

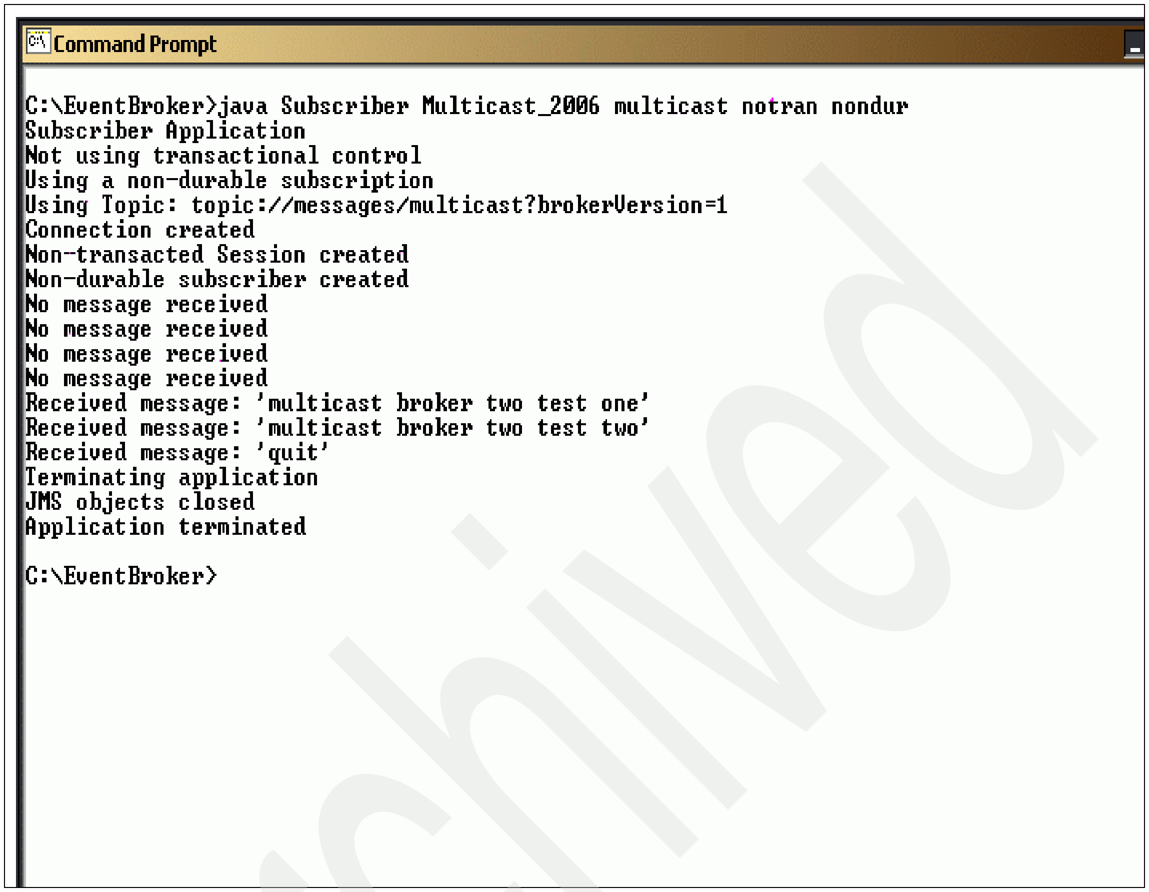
The first subscriber connects to the second broker in the collective using the WebSphere MQ real-time transport in unicast mode. It receives all four messages published by both publishers, showing that the collective is operating correctly. This is shown in Figure 15-3 on page 220.



```
C:\EventBroker>java Subscriber IP_2006 multicast notran nondur
Subscriber Application
Not using transactional control
Using a non-durable subscription
Using Topic: topic://messages/multicast?brokerVersion=1
Connection created
Non-transacted Session created
Non-durable subscriber created
No message received
No message received
Received message: 'multicast broker one test one'
Received message: 'multicast broker one test two'
Received message: 'multicast broker two test one'
Received message: 'multicast broker two test two'
Received message: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 15-3 The unicast subscriber

The second subscriber also connects to the second broker in the collective, but this time uses the real-time transport in multicast mode. It receives only the two messages published by the publisher also connected to the second broker, as shown in Figure 15-4 on page 221.



```
C:\EventBroker>java Subscriber Multicast_2006 multicast notran nondur
Subscriber Application
Not using transactional control
Using a non-durable subscription
Using Topic: topic://messages/multicast?brokerVersion=1
Connection created
Non-transacted Session created
Non-durable subscriber created
No message received
No message received
No message received
No message received
Received message: 'multicast broker two test one'
Received message: 'multicast broker two test two'
Received message: 'quit'
Terminating application
JMS objects closed
Application terminated

C:\EventBroker>
```

Figure 15-4 The multicast subscriber

The messages that come into the second broker from across the collective are matched to the unicast subscriber but not the multicast subscriber. This is correct behavior and is the result of a design decision in the product.

The best way of thinking about unicast (either WebSphere MQ queues or real-time transport) and multicast in a broker collective is by considering how the brokers act (or appear to act) to their subscribers.

- For unicast subscribers, the brokers act collectively by sharing publications.
- For multicast subscribers, the brokers appear to act individually, as multicast subscribers only receive messages published to the same broker they are connected to.

Despite this behavior in a collective, the configuration of multiple brokers for multicast shown above is important. If separate brokers exist on the same subnet, they can still interfere with each other's multicast group allocations, even if not in a collective or connected directly in any other way. Multiple brokers on the same network that use multicast must *always* be carefully configured to use separate partitions of the multicast group space.

A

Code used in the business case scenario

This chapter provides JMS source code for various scenarios demonstrated in Chapter 9, “Real-time transport for single broker performance” on page 165, through Chapter 15, “Using multicast in a broker collective” on page 215.

Code used to demonstrate various scenarios

The softcopy is available on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246088

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbook form number, SG246088.

Publisher application code

```
import javax.jms.*;
import java.io.*;
import java.util.*;
import javax.naming.*;
import javax.naming.directory.*;
// A publisher class
// Takes in a TCF and topic and looks them up in JNDI
// Creates JMS objects in order to publish messages
// Waits for command line input for message data
// Loops until it receives "quit"
// Closes JMS objects and terminates
public class Publisher
{
    public static void main(String args[])
    {
        // Strings for default ICF and URL for JNDI connection
        String icf = "com.sun.jndi.fscontext.RefFSContextFactory";
        String url = "file://C:/jndi";

        // Variables to track modes
        boolean trans = true;
        boolean pers = true;

        // Required JMS object declarations
        Context ctx = null;
        TopicConnectionFactory tcf = null;
        Topic topic = null;
        TopicConnection conn = null;
        TopicSession session = null;
        TopicPublisher pub = null;
        BufferedReader in = null;
```

```
output("Publisher Application");

// Check our input parameters are correct
if(args.length != 4)
{
    usage();
    System.exit(0);
}

// Verify our transactionality from input parameter
if(args[2].equals("tran")) trans = true;
else if(args[2].equals("notran")) trans = false;
else
{
    usage();
    System.exit(0);
}

// Verify our message persistence from input parameter
if(args[3].equals("pers")) pers = true;
else if(args[3].equals("nonpers")) pers = false;
else
{
    usage();
    System.exit(0);
}

if(trans) output("Using transactional control");
else output("Not using transactional control");

if(pers) output("Messages are persistent");
else output("Message are not persistent");

try
{
    // Create a Hashtable for JNDI required parameters
    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY, icf);
    env.put(Context.PROVIDER_URL, url);
    env.put(Context.REFERRAL, "throw");

    // Initialise the JNDI Context
    ctx = new InitialDirContext(env);
}
catch(Exception e)
{
    output("Unable to create JNDI context");
    e.printStackTrace();
    System.exit(0);
```

```
}

// Obtain the TCF from JNDI context
try
{
    tcf = (TopicConnectionFactory)ctx.lookup(args[0]);
}
catch(Exception e)
{
    usage();
    e.printStackTrace();
    try{ctx.close();}catch(Exception ce){}
    System.exit(0);
}

// Obtain the Topic from JNDI context
try
{
    topic = (Topic)ctx.lookup(args[1]);
    output("Using Topic: " + topic.toString());
}
catch(Exception e)
{
    usage();
    e.printStackTrace();
    try{ctx.close();}catch(Exception ce){}
    System.exit(0);
}

// Close our JNDI context
try{ctx.close();}catch(Exception ce){}

// Create a reader from stdin to take text for published messages
try
{
    in = new BufferedReader(new InputStreamReader(System.in));
}
catch(Exception e)
{
    output("Unable to create BufferedReader from System.in");
    e.printStackTrace();
    System.exit(0);
}

// Create our JMS objects
try
{
    // Create TopicConnection
    conn = tcf.createTopicConnection();
```

```
        output("Connection created");

        // Create TopicSession with required transactionality
        session = conn.createTopicSession(trans,
                                          Session.AUTO_ACKNOWLEDGE);
        if(trans) output("Transacted Session created");
        else output("Non-transacted Session created");

        // Create TopicPublisher
        pub = session.createPublisher(topic);

        // Set correct delivery mode (persistence) on publisher
        if(pers) pub.setDeliveryMode(DeliveryMode.PERSISTENT);
        else pub.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
        if(pers) output("Publisher for persistent messages created");
        else output("Publisher for non-persistent messages created");
    }
    catch(JMSException e)
    {
        output("Unable to create JMS objects");
        e.printStackTrace();
        Exception le = e.getLinkedException();
        if(le != null) le.printStackTrace();

        // Close the reader
        if(in != null)
            try{in.close();}catch(Exception ie){}

        // Close our JMS objects
        if(pub != null)
            try{pub.close();}catch(JMSException je){}
        if(session != null)
            try{session.close();}catch(JMSException je){}
        if(conn != null)
            try{conn.close();}catch(JMSException je){}

        System.exit(0);
    }

    // Go into a publishing loop
    String input = "";
    boolean run = true;
    while((!(input.equals("quit"))) && run)
    {
        output("Waiting for input...");
        try
        {
            // Wait for input from command line
            input = in.readLine();
            // Received input, create and publish message from it
            TextMessage msg = session.createTextMessage();
            msg.setText(input);
            pub.publish(msg);
            output("Message published: '"+input+"'");

            // Commit the send if we are transacted
            if(trans) session.commit();

            // NOTE even "quit" messages are published
            // This stops the subscribers too
        }
        catch(JMSException e)
        {
            output("Error while publishing messages");
            e.printStackTrace();
            Exception le = e.getLinkedException();
            if(le != null) le.printStackTrace();
            run = false;
        }
        catch(IOException ie)
        {
            output("Error while publishing messages");
            ie.printStackTrace();
            run = false;
        }
    }

    output("Terminating application");
    // Close the reader
    if(in != null)
        try{in.close();}catch(Exception e){}

    // Close our JMS objects
    if(pub != null)
        try{pub.close();}catch(JMSException je){}
    if(session != null)
        try{session.close();}catch(JMSException je){}
    if(conn != null)
        try{conn.close();}catch(JMSException je){}
    output("JMS objects closed");
    output("Application terminated");
}

private static void usage()
{
    output("java Publisher TCF Topic tran|notran pers|nonpers");
}
```

```
    private static void output(String s)
    {
        System.out.println(s);
    }
}
```

Subscriber application code

```
import javax.jms.*;
import java.io.*;
import java.util.*;
import javax.naming.*;
import javax.naming.directory.*;

// A subscriber class
// Takes in a TCF and topic and looks them up in JNDI
// Creates JMS objects in order to receive messages over Pub/Sub
// Loops receiving messages until it receives "quit"
// Closes JMS objects and terminates

public class Subscriber
{
    public static void main(String args[])
    {
        // Strings for default ICF and URL for JNDI connection
        String icf = "com.sun.jndi.fscontext.RefFSContextFactory";
        String url = "file://C:/jndi";

        // Variables to track modes
        String subid = "";
        boolean trans = true;
        boolean dur = true;

        // Required JMS object declarations
        Context ctx = null;
        TopicConnectionFactory tcf = null;
        Topic topic = null;
        TopicConnection conn = null;
        TopicSession session = null;
        TopicSubscriber sub = null;
        BufferedReader in = null;

        output("Subscriber Application");

        // Check our input parameters are correct
```

```
if((args.length != 4) && (args.length != 5))
{
    usage();
    System.exit(0);
}

// Verify our transactionality from input parameter
if(args[2].equals("tran")) trans = true;
else if(args[2].equals("notran")) trans = false;
else
{
    usage();
    System.exit(0);
}

// Verify our subscriber durability from input parameter
if(args[3].equals("dur")) dur = true;
else if(args[3].equals("nondur")) dur = false;
else
{
    usage();
    System.exit(0);
}

// Detect and use subscriber ID if we are durable
if(dur) subid = args[4];

if(trans) output("Using transactional control");
else output("Not using transactional control");

if(dur) output("Using a durable subscription, subscriber ID: " +
               subid);
else output("Using a non-durable subscription");

try
{
    // Create a Hashtable for JNDI required parameters
    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY, icf);
    env.put(Context.PROVIDER_URL, url);
    env.put(Context.REFERRAL, "throw");

    // Initialise the JNDI Context
    ctx = new InitialDirContext(env);
}
catch(Exception e)
{
    output("Unable to create JNDI context");
    e.printStackTrace();
```

```
        System.exit(0);
    }

    // Obtain the TCF from JNDI context
    try
    {
        tcf = (TopicConnectionFactory)ctx.lookup(args[0]);
    }
    catch(Exception e)
    {
        usage();
        e.printStackTrace();
        try{ctx.close();}catch(Exception ce){}
        System.exit(0);
    }

    // Obtain the Topic from JNDI context
    try
    {
        topic = (Topic)ctx.lookup(args[1]);
        output("Using Topic: " + topic.toString());
    }
    catch(Exception e)
    {
        usage();
        e.printStackTrace();
        try{ctx.close();}catch(Exception ce){}
        System.exit(0);
    }

    // Close our JNDI context
    try{ctx.close();}catch(Exception ce){}

    // Create our JMS objects
    try
    {
        // Create TopicConnection
        conn = tcf.createTopicConnection();
        output("Connection created");

        // Create TopicSession with required transactionality
        session = conn.createTopicSession(trans,
                                          Session.AUTO_ACKNOWLEDGE);
        if(trans) output("Transacted Session created");
        else output("Non-transacted Session created");

        // Create TopicSubscriber
        if(dur)
        {
            // Create durable subscription
            sub = session.createDurableSubscriber(topic, subid);
            output("Durable subscriber created");
        }
        else
        {
            // Create ordinary subscription
            sub = session.createSubscriber(topic);
            output("Non-durable subscriber created");
        }
        conn.start();
    }
    catch(JMSException e)
    {
        output("Unable to create JMS objects");
        e.printStackTrace();
        Exception le = e.getLinkedException();
        if(le != null) le.printStackTrace();

        // Close our JMS objects
        if(sub != null)
            try{sub.close();}catch(JMSException je){}
        if(session != null)
            try{session.close();}catch(JMSException je){}
        if(conn != null)
            try{conn.close();}catch(JMSException je){}

        System.exit(0);
    }

    // Go into a subscribing loop
    boolean run = true;
    while(run)
    {
        try
        {
            // Wait for 5 seconds to receive a message
            TextMessage msg = (TextMessage)sub.receive(5000);

            if(msg != null)
            {
                // We got a message so process it
                String data = msg.getText();
                output("Received message: '"+data+"'");

                // If "quit" then stop looping
                if(data.equals("quit")) run = false;

                // Commit the receive if we are transacted
                if(trans) session.commit();
            }
            else
            {
                // We didn't get a message, so loop round again
                output("No message received");
            }
        }
        catch(JMSException e)
        {
            output("Error while receiving messages");
            e.printStackTrace();
            Exception le = e.getLinkedException();
            if(le != null) le.printStackTrace();
            run = false;
        }
    }

    output("Terminating application");

    // Close our JMS objects
    if(sub != null)
        try{sub.close();}catch(JMSException je){}

    // Check to see if we are to delete the durable subscription
    if(dur)
    {
        try
        {
            // Create a reader from stdin
            in = new BufferedReader(new InputStreamReader(System.in));
            System.out.print("'y' deletes the durable subscription: ");
            String input = in.readLine();
            if(input.equals("y"))
            {
                // Unsubscribe the durable subscription
                session.unsubscribe(subid);
                output("Durable subscription deleted");
            }
        }
        catch(JMSException e)
        {
            output("Error unsubscribing durable subscription");
            e.printStackTrace();
            Exception le = e.getLinkedException();
            if(le != null) le.printStackTrace();
        }
        catch(Exception e)
        {
                output("Error unsubscribing durable subscription");
                e.printStackTrace();
            }

        }

        // Close the rest of the JMS objects
        if(session != null)
            try{session.close();}catch(JMSException je){}
        if(conn != null)
            try{conn.close();}catch(JMSException je){}
        output("JMS objects closed");
        output("Application terminated");
    }

    private static void usage()
    {
        output("java Subscriber TCF Topic tran|notran dur|nondur [subID]");
    }

    private static void output(String s)
    {
        System.out.println(s);
    }
}
```

B

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246088

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG246088.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name	*Description*
code.zip	All Zipped Code samples used in the business case scenario

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	0.55 KB minimum
Operating System:	Windows 2000 or Windows XP (Professional or Server)
Processor:	Pentium® III 500MHz or better
Memory:	256 MB or higher is recommended

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Abbreviations and acronyms

ACL	Access Control Lists
AMI	Application Messaging Interface
API	Application Programming Interface
DLQ	Dead Letter Queue
GUI	Graphical User Interface
HA	High Availability
IBM	International Business Machines Corporation
ITSO	International Technical Support Organization
JDBC	Java Database Connection
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JNI	Java Native Interface
LDAP	Lightweight Data Access Protocol
MOM	Message Oriented Middleware
MQI	Message Queue Interface
OAM	Object Authority Manager
ODBC	Open Database Connectivity
SCADA	Supervisory, Control And Data Acquisition
SOAP	Simple Object Access Protocol
SPOF	Single Point of Failure
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TAG	Tag Delimited String
TTL	Time To Live
UoW	Unit of Work
URL	Universal Resource Locator
XML	Extensible Markup Language

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see "How to get IBM Redbooks" on page 239. Note that some of the documents referenced here may be available in softcopy only.

- *MQSeries Publish/Subscribe Applications,* SG24-6282

Other publications

These publications are also relevant as further information sources:

- *WebSphere MQ Security,* SC34-6079

Online resources

These Web sites and URLs are also relevant as further information sources:

- SupportPac, MA88

  `http://www-4.ibm.com/software/integration/support/supportpacs/`

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

- IBM Support and downloads

  ibm.com/support

- IBM Global Services

  ibm.com/services

Index

IBM

Redbooks

WebSphere Business Integration Pub/Sub Solutions

WebSphere Business Integration Pub/Sub Solutions

JMS scenarios showing high message rate and high availability

Brokers using persistent and real-time transports

Pub/Sub architectures using WBI Event Broker

This IBM Redbook provides both technical background and implementation best practise recommendations for WebSphere Business Integration Pub/Sub solutions, with emphasis on the use of JMS API. It shows the differences between tuning for performance (high message rate) and robustness (high availability), and how to balance these to provide the correct solution for a set of business requirements.

The first part of this publication provides an overview of the technology behind the Pub/Sub messaging paradigm, leading to how best to use the WebSphere Business Integration product family to provide a Pub/Sub solution.

The second part introduces a sample set of applications and shows how to develop the Pub/Sub infrastructure to meet different sets of business requirements. The scenarios cover configurations involving multicast, cloned brokers, persistence and stream-crossing.

SG24-6088-00 ISBN 073849769X