IBM

WebSphere Message Broker Basics

Introduces WebSphere Message Broker V6

Describes basic installation, configuration, and development tasks

Explores the Message Brokers Toolkit

Saida Davies
Laura Cowen
Cerys Giddings
Hannah Parker

Redbooks

IBM

International Technical Support Organization

WebSphere Message Broker Basics

December 2005

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

First Edition (December 2005)

This edition applies to:

Version	Release	Modification	Product name	Product Number	Platform
6	0	0	WebSphere Message Broker	Windows	5724-J05
6	0	0	WebSphere MQ	Windows	5724-H72
8	2	0	DB2 UDB Enterprise Server Edition	Windows	5765-F41
6	0	1	IBM Rational Agent Controller	Windows	N/A

Contents

Figures

ix

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

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

Redbooks (logo) ™	DB2 Universal Database™	Parallel Sysplex®
developerWorks®	DB2®	Rational®
z/OS®	IBM®	Redbooks™
Cloudscape™	MQSeries®	WebSphere®

The following terms are trademarks of other companies:

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

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

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Preface

This IBM® Redbook is a guide to using WebSphere® Message Broker V6.0.

This book updates the popular redbook SG24-7090, which contained basic material for WebSphere Business Integration Message Broker V5.0. This book covers many of the significant updates and new features that have been introduced in WebSphere Message Broker V6.0.

This book starts with instructions for installation, basic configuration, and checking that the product is configured correctly.

Guidance is provided for developing basic message flows and message sets. Message transformation is demonstrated using the main range of techniques available in the Message Brokers Toolkit: ESQL, Java™, and mapping.

Common administration tasks are also described for WebSphere Message Broker, including how to configure broker domains and how to deploy message flow applications to the broker.

The sample message flows and message sets that are demonstrated in this book are available for download from the Web.

This book includes details about where to look for diagnostic information in the product and help with solving problems. This includes how to use the Flow Debugger.

Assistance is provided on where to find more information about the product, including product documentation updates and sample applications.

In addition, the application development chapter from the WebSphere Business Integration Message Broker Basics book has been extended to three chapters to include more details about how to perform message application development using ESQL, Java, and mappings.

The team that wrote this redbook

This book was produced by a team of specialists from the home of Business Integration and Messaging Middleware product development, IBM Hursley.

From left, back row: Saida, Cerys
Front row: Hannah, Laura

Photograph by: James P Hodgson
WebSphere MQ & ESB Delivery - Test
IBM Hursley

Saida Davies is a Project Leader for the International Technical Support
Organization (ITSO) and has seventeen years of experience in IT. She has
published several IBM Redbooks™ on various business integration scenarios.
Saida has experience in the architecture and design of WebSphere MQ
solutions, 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 as a Senior IT Specialist with IBM
Global Services, her role included the development of services for z/OS and

WebSphere MQ within the z/OS and Windows® platform. 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 received Bravo Awards for her project contribution. She has a degree in Computer Studies and her background includes z/OS systems programming. Saida supports Women in Technology activities, and contributes and participates in the their meetings.

Laura Cowen is a Technical Writer who has worked on the WebSphere MQ family of products since joining the IBM Hursley Software Labs in 2002. Her fields of expertise are usability, documentation, and Eclipse help systems. She was involved in the development of the out-of-box experience of WebSphere Message Broker and the Eclipse-based WebSphere MQ Explorer. She is co-author of the redbook WebSphere Business Integration Message Broker Basics.
Laura actively participates in the British and international human-computer interaction (HCI) community and has several publications in the field, including a conference paper about usability evaluation methodologies. For three years, she has been Editor of Interfaces, the quarterly magazine of the HCI specialist group in the British Computer Society. She is also involved in promoting the use of Open Source Software and Linux® in desktop computing, and encouraging teenage girls to consider careers in IT.
Prior to joining IBM, Laura was Lead Usability Researcher for an information design consultancy. She holds a first class honours degree in Psychology and a Masters in human-computer interaction from Lancaster University, UK.

Cerys Giddings is a Usability Practitioner for WebSphere Message Broker, working on many of the usability enhancements for the product over the last two versions. She has worked on the WebSphere MQ family of products since joining IBM Hursley in 2000. A former Team Leader from the WebSphere MQ Test organization, she has participated in producing the IBM Certified System Administrator - WebSphere Business Integration Message Broker V5 certification tests. Cerys is co-author of the WebSphere Business Integration Message Broker Basics, WebSphere InterChange Server Migration Scenarios, and WebSphere Message Broker V6 Migration redbooks. She has over 10 years of experience in providing IT education and support, and holds a Masters from the University of Wales, as well as the BCS Professional Examinations at Certificate and Diploma levels.

Hannah Parker is a Software Engineer and has worked in IBM for three years since graduating from the University of Exeter with a Bachelors of Science degree in Cognitive Science. She joined the Hursley Development Laboratory and has worked both as a technical information developer and a software tester for many of the IBM products including WebSphere Application Server, WebSphere Platform Messaging, WebSphere Message Brokers, WebSphere

Voice Response, and WebSphere Voice Application Access. While working in a technical environment, she has focused her interests on the end-user experience and product consumability. As such, she has developed her expertise to be able to explain complicated concepts in simple and comprehensible ways. Hannah has co-authored a developerWorks® tutorial for the CCXML language. In addition to her daily job responsibilities, Hannah is an active member of the IBM Hursley community. In 2004 she co-led the Graduate Induction team, running a week-long event to introduce and integrate new graduates to IBM Hursley. Her enthusiasm for science, engineering, and IT enables her to actively mentor girls in local schools through mentorplace scheme, and also be part of the organizing team for the week-long Blue Fusion, Hursley's contribution to National Science Week.

The team would like to thank the following people for their support and contributions to this project:

Darren Stuart, Application developer/consultant
IBM Global Services, IBM Business Consulting Services
IBM Australia

Julie Czubik
International Technical Support Organization, Poughkeepsie Center

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 email to:

    redbook@us.ibm.com

► Mail your comments to:

    IBM Corporation, International Technical Support Organization
    Dept. 8IB IBM United Kingdom Ltd
    12195
    Research Triangle Park, Dept. HZ8 Building 662, NC

1

Introduction

This chapter provides an overview of the scope of this book and includes a brief introduction to the contents of the book.

The following topics are discussed:

- ▶ The scope of the redbook
- ▶ Intended audience
- ▶ Overview of chapters
- ▶ What is not covered in this book
- ▶ Assumptions

1.1 The scope of this book

The aim of this book is to introduce new users to the concepts and basic functionality of WebSphere Message Broker V6.0.

The following products are discussed:

- ► WebSphere Message Broker V6.0
- ► WebSphere Event Broker V6.0

WebSphere Event Broker contains a subset of the features that are available in WebSphere Message Broker. Discussion in this redbook focusses on the features that are available in WebSphere Message Broker and highlights only when a feature is not available in WebSphere Event Broker (for example, message sets).

This book provides basic information that is designed to enable users to be able to install and configure the product, to develop and deploy simple applications, and to complete common administration tasks, including how to configure message broker domains.

This book provides step-by-step instructions on creating WebSphere Message Broker components and developing a set of simple message flows and message sets to demonstrate how to use the key capabilities in the product.

In addition to information about the basic functionality of WebSphere Message Broker, this book gives details of where to look for diagnostic information in the product and offers troubleshooting and problem determination advice. Assistance is also provided on where to get help with solving problems, and where to find more information about the product, including product documentation updates and sample applications.

The sample message flows and message sets used for demonstration purposes in this book can be downloaded from the Web. See Appendix B, "Code" on page 319, for instructions on how to download these samples.

The Message Brokers Toolkit is available on Windows and Linux (x86 platform), but this book focuses on Windows for simplicity. The instructions provided for development of resources, administration, and problem determination for the Message Brokers Toolkit also apply to Linux. The majority of the concepts and other steps introduced in the book are also valid for other platforms, although operating system differences apply to some areas, such as accessing the system log and the locations of files.

1.1.1 Intended audience

This book is targeted at new users, providing information to enable users to start using WebSphere Message Broker V6.0 as quickly as possible. It describes the most direct way to install the product, provides information for developing applications, and offers basic information for administering and problem solving.

Although this book is aimed at new users, it is also helpful for users of any previous versions of WebSphere Message Broker who are not familiar with the Message Brokers Toolkit.

Instructions are provided in this book for some features that are new to this version of the product, such as message mapping, runtime versionning, and the JavaCompute node. These will provide a useful introduction to these new features for experienced users. For descriptions of the all the major new features for WebSphere Message Broker V6.0 refer to *Migrating to WebSphere Message Broker V6.0*, SG247198.

1.1.2 Overview of the topics covered

This section gives a short overview of the topics that are covered in each of the chapters in this book.

► Chapter 2, "Product overview" on page 7

This chapter gives a high-level introduction to WebSphere Message Broker. It describes the basic components of the product, and discusses some of the product's capabilities. This chapter also provides an overview of application integration, including the role of WebSphere Message Broker in application integration.

► Chapter 3, "Installation" on page 17

This chapter describes the software that is required to install both the Express Installation and Advanced Installation of WebSphere Message Broker V6.0, including the optional software. The function and relevance of the software is explained. Instructions for installing WebSphere Message Broker,and configuring and verifying the installation are provided, along with a discussion of relevant security considerations and system requirement information.

► Chapter 4, "Developing applications with ESQL" on page 47

This chapter describes how to create message flow applications in which the logic of the message flows is defined by using ESQL. The chapter provides step-by-step instructions on how to create and deploy two message flow applications: A simple message flow and a more complex message flow application. The complex message flow application contains two message flows that access and update a DB2® database table and create new messages. All the resources and ESQL code that you need to create the

message flow applications in this chapter are available to download from the Internet.

► Chapter 5, "Developing applications with Java" on page 97

This chapter describes how to create message flow applications in which the logic of the message flows is defined using Java. The chapter provides step-by-step instructions to create and deploy two message flow applications: A simple message flow and a more complex message flow application. The complex message flow application contains two message flows that access and update a DB2 database table and create new messages. All the resources and Java code that you need to create the message flow applications in this chapter are available for download from the Internet.

► Chapter 6, "Developing applications with mappings" on page 135

This chapter describes how to create message flow applications in which the logic of the message flows is defined using the graphical mapping tools in the Message Brokers Toolkit. The chapter provides step-by-step instructions to create and deploy two message flow applications: A simple message flow and a more complex message flow application. The complex message flow application contains two message flows that access and update a DB2 database table, use an external message set and message mappings to create a new message. All the resources that you need to create the message flow applications in this chapter are available to download from the Internet.

► Chapter 7, "Administration" on page 205

This chapter provides an overview of the administration of the runtime environment for WebSphere Message Broker. Instructions are provided for manually creating a simple broker domain and adding remote brokers. Information is also provided on deploying message flow applications to the broker and adding versioning information to the resources. The chapter also introduces the concepts and use of publish/subscribe.

► **Chapter 8, "Troubleshooting and problem determination" on page 241**

This chapter provides assistance with determining the cause and resolution of problems when using WebSphere Message Broker. These include locating error information, using trace, and troubleshooting common problems. Instructions are also provided for using the Flow Debugger to debug message flows including debugging ESQL, Java, and mappings.

1.1.3 What is not covered

Information about migrating from previous versions of WebSphere Message Broker is not discussed in this book. For detailed information about product migration (including migrating from V2.1, V5.0, and V5.1) and information about

the new features in WebSphere Message Broker V6.0, refer to *Migrating to WebSphere Message Broker V6.0*, SG247198.

As a basic introduction to WebSphere Message Broker, this book does not contain advanced information about messaging architectures or about the development and maintenance of production systems.

The code examples contained in this book are not designed to be high performance code and, therefore, should be used in test and evaluation environments only. The code examples used are to demonstrate the capability of the product and how to perform basic tasks. They do not demonstrate best practices with the product for production environments. When designing message flow applications it is important to refer to the product documentation for advice on selecting the most appropriate nodes for the task, and for advice on coding for best practices, standards, and for best performance.

Platform-specific information is restricted to Windows, although much of the information can be applied to other platforms.

1.1.4 Assumptions

This book makes a number of assumptions in order to simplify the information here and make it useful to as many customers as possible.

The following assumptions are made:

► Customers have administrator privileges for the system on which WebSphere Message Broker is installed.

► Customers are familiar with the Microsoft® Windows operating system.

► Customers have access to the Internet.

This book assumes that the user is working with WebSphere Message Broker V6.0. It is recommended that, when made available, users apply the latest fix packs. For information on the latest available fix pack and instructions on how to install fix packs, refer to "Installing product fix packs" on page 43.

Product overview

This chapter provides a high-level introduction to WebSphere Message Broker describing its role in application integration. The main capabilities and components of the product are described.

This chapter discusses the following topics:

► WebSphere Message Broker in application integration
► Versions of WebSphere Message Broker
► WebSphere Message Broker capabilities and components

7

2.1 Application integration

Application integration is a big challenge for enterprises, and IBM provides a number of software solutions and offerings to assist companies with integrating their applications. WebSphere Message Broker is an important part of the IBM offerings, and the way in which WebSphere Message Broker benefits application integration is described in the following sections.

2.1.1 Application integration and WebSphere Message Broker

Application integration, at a high level, refers to solutions that are implemented to integrate software applications within and between organizations. Historically, application integration has been concerned with the integration of legacy software applications, such as between different departments and divisions within companies, or new acquisitions. Within an organization, these applications often vary considerably across departments, exist on different platforms, are written in different programming languages, and use different data formats. Integrating the applications is a more practical and cost effective solution than the alternative of re-writing the existing applications.

WebSphere Message Broker is used in the implementation of an application integration architecture because it provides a mechanism for connecting, routing, and transforming business data from a variety of transports without any need to change the underlying applications generating the data.

WebSphere Message Broker enhances the flow and distribution of information by enabling the transformation and intelligent routing of messages without the need to change either the applications that are generating the messages or the applications that are consuming them. In WebSphere Message Broker, connectivity is provided by applications that communicate by sending and receiving messages.

WebSphere Message Broker also has the following key capabilities that make it a valuable solution for business integration:

► Distributes any type of information across and between multiple diverse systems and applications, providing delivery of the correct information in the correct format and at the correct time

► Reduces the number of point-to-point interconnections and simplifies application programming by removing integration logic from the applications

► Routes information in real time based on topic and content to any endpoint using a powerful publish/subscribe messaging engine

- Validates and transforms messages in-flight between any combination of different message formats, including Web Services, and other XML and non-XML formats
- Routes messages based on (evaluated) business rules to match information content and business processes
- Improves business agility by dynamically reconfiguring information distribution patterns without reprogramming end-point applications
- Access control to securely deliver personalized information to the right place at the right time

2.2 WebSphere Message Broker

WebSphere Message Broker is a powerful information broker that allows both business data and information, in the form of messages, to flow between disparate applications and across multiple hardware and software platforms. Business rules can be applied to the data that is flowing through the message broker in order to route, store, retrieve, and transform the information.

2.2.1 Editions of WebSphere Message Broker

There are three editions of the WebSphere Message Broker product as described in the sections below.

WebSphere Event Broker

WebSphere Event Broker is used for the distribution and routing of messages from disparate applications. It can distribute information and data, which is generated by business events in real time, to people, applications, and devices throughout an enterprise. WebSphere Event Broker has support for multiple transport protocols and extends the flow of information in an organization beyond point to point, utilizing flexible distribution mechanisms such as publish/subscribe and multicast.

WebSphere Message Broker

WebSphere Message Broker contains all the functionality of WebSphere Event Broker and extends it to include additional capabilities to enable storage, transformation, and enrichment of data flowing through the broker. Detailed capabilities of the product are described in the following sections and are based upon the functional capabilities of the WebSphere Message Broker specifically.

Rules and Formatter Extension

WebSphere Message Broker with Rules and Formatter Extension includes the Rules and Formatter Extension from New Era Of Networks which provides Rules and Formatter nodes and associated runtime elements. These support the functionality supplied with earlier releases of WebSphere MQ Integrator. The functionality provided by the Rules and Formatter Extension is not discussed any further in this book.

2.2.2 Capabilities of WebSphere Message Broker

The primary capabilities of WebSphere Message Broker are message routing, message transformation, message enrichment, and publish/subscribe. Together these capabilities make WebSphere Message Broker a powerful tool for business integration.

Message routing

WebSphere Message Broker provides connectivity for both standards based and non-standards based applications and services. The routing can be simple point-to-point routing or it can be based on matching the content of the message to business rules defined to the broker.

WebSphere Message Broker contains a choice of transports that enable secure business to be conducted at any time and any place, providing powerful integration using mobile, telemetry, and Internet technologies. WebSphere Message Broker is built upon WebSphere MQ and therefore supports the same transports. However, it also extends the capabilities of WebSphere MQ by adding support for other protocols, including real-time Internet, intranet, and multicast endpoints.

WebSphere Message Broker supports the following transports:

- ► WebSphere MQ Enterprise Transport
- ► WebSphere MQ Web Services Transport
- ► WebSphere MQ Real-time Transport
- ► WebSphere MQ Multicast Transport
- ► WebSphere MQ Mobile Transport
- ► WebSphere MQ Telemetry Transport
- ► WebSphere Broker JMS Transport

In addition to the supplied transports, the facility exists for users to write their own input nodes to accept messages from other transports and formats as defined by the user.

Message transformation and enrichment

Transformation and enrichment of in-flight messages is an important capability of WebSphere Message Broker, and allows for business integration without the need for any additional logic in the applications themselves.

Messages can be transformed between applications to use different formats, for example, transforming from a custom format in a legacy system to XML messages that can be used with a Web service. This provides a powerful mechanism to unify organizations because business information can now be distributed to applications that handle completely different message formats without a need to reprogram or add to the applications themselves.

Messages can also be transformed and enriched by integration with multiple sources of data such as databases, applications, and files. This allows for any type of data manipulation including logging, updating, and merging. For the messages that flow through the broker, business information can be stored in databases or can be extracted from databases and files and added to the message for processing in the target applications.

Complex manipulation of message data can be performed using the facilities provided in the Message Brokers Toolkit, such as ESQL and Java.

In WebSphere Message Broker, message transformation and enrichment is dependant upon a broker understanding the structure and content of the incoming message. Self-defining messages, such as XML messages, contain information about their own structure and format. However, before other messages, such as custom format messages, can be transformed or enhanced, a message definition of their structure must exist. The Message Brokers Toolkit contains facilities for defining messages to the message broker. These facilities are discussed in more detail below.

Publish/subscribe

The simplest way of routing messages is to use point-to-point messaging to send messages directly from one application to another. Publish/subscribe provides an alternative style of messaging in which messages are sent to all applications that have subscribed to a particular topic.

The broker handles the distribution of messages between publishing applications and subscribing applications. As well as applications publishing on or subscribing to many topics, more sophisticated filtering mechanisms can be applied.

An improved flow of information around the business is achieved through the use of publish/subscribe and the related technology of multicast. These flexible distribution mechanisms move away from hard-coded point-to-point links.

2.2.3 Components of WebSphere Message Broker

WebSphere Message Broker is comprised of two principle parts, a *development environment* for the creation of message flows, message sets, and other message flow application resources; and a *runtime environment*, which contains the components for running those message flow applications that are created in the development environment.

Development environment

The development environment is where the message flow applications that provide the logic to the broker are developed. The broker uses the logic in the message flow applications to process messages from business applications at run time. In the Message Brokers Toolkit, you can develop both message flows and message sets for message flow applications.

Message flows

Message flows are programs that provide the logic that the broker uses to process messages from business applications. Message flows are created by connecting nodes together, with each node providing some basic logic. A selection of built-in nodes is provided with WebSphere Message Broker for performing particular tasks. These tasks can be combined to perform complex manipulations and transformations of messages.

A choice of methods is available for defining the logic in the message flows to transform data. Depending on the different types of data or the skills of the message flow application developer, the following options are available:

► Extended Structured Query Language (ESQL)
► Java
► eXtensible Style sheet Language for Transformations (XSLT)
► Drag-and-drop mappings

The nodes in the message flows define the source and the target transports of the message, any transformations and manipulations based on the business data, and any interactions with other systems such as databases and files.

For an example message flow, see the simple ESQL message flow shown in Figure 4-5 on page 55.

Message sets

A message set is a definition of the structure of the messages that are processed by the message flows in the broker. As mentioned previously, in order for a message flow to be able to manipulate or transform a message, the broker must know the structure of the message. The definition of a message can be used to

verify message structure, and to assist with the construction of message flows and mappings in the Message Brokers Toolkit.

Message sets are compiled for deployment to a broker as a message dictionary, which provides a reference for the message flows to verify the structure of messages as they flow through the broker.

Broker Application Development perspective

The Broker Application Development perspective is the part of the Message Brokers Toolkit that is used to design and develop message flows and message sets. It contains editors that create message flows, create transformation code such as ESQL, and define messages.

Runtime environment

The runtime environment is the set of components that are required to deploy and run message flow applications in the broker.

Broker

The broker is a set of application processes that host and run message flows. When a message arrives at the broker from a business application, the broker processes the message before passing it on to one or more other business applications. The broker routes, transforms, and manipulates messages according to the logic that is defined in their message flow applications.

A broker uses WebSphere MQ as the transport mechanism both to communicate with the Configuration Manager, from which it receives configuration information, and to communicate with any other brokers to which it is associated.

Each broker has a database in which it stores the information that it needs to process messages at run time.

Execution groups

Execution groups enable message flows within the broker to be grouped together. Each broker contains a default execution group. Additional execution groups can be created as long as they are given unique names within the broker.

Each execution group is a separate operating system process and, therefore, the contents of an execution group remain separate from the contents of other execution groups within the same broker. This can be useful for isolating pieces of information for security because the message flows execute in separate address spaces or as unique processes.

Message flow applications are deployed to a specific execution group. To enhance performance, the same message flows and message sets can be running in different execution groups.

Configuration Manager

The Configuration Manager is the interface between the Message Brokers Toolkit and the brokers in the broker domain. The Configuration Manager stores configuration details for the broker domain in an internal repository, providing a central store for resources in the broker domain.

The Configuration Manager is responsible for deploying message flow applications to the brokers. The Configuration Manager also reports back on the progress of the deployment and on the status of the broker. When the Message Brokers Toolkit connects to the Configuration Manager, the status of the brokers in the domain is derived from the configuration information stored in the Configuration Manager's internal repository.

Broker domain

Brokers are grouped together in broker domains. The brokers in a single broker domain share a common configuration that is defined in the Configuration Manager. A broker domain contains one or more brokers and a single Configuration Manager. It can also contain a User Name Server. The components in a broker domain can exist on multiple machines and operating systems, and are connected together with WebSphere MQ channels.

A broker belongs to only one broker domain.

For information about creating a broker domain, and administration tasks that can be performed over the domain, see Chapter 7, "Administration" on page 205.

User Name Server

A User Name Server is an optional component that is required only when publish/subscribe message flow applications are running, and where extra security is required for applications to be able to publish or subscribe to topics. The User Name Server provides authentication for topic-level security for users and groups that are performing publish/subscribe operations.

Broker Administration perspective

The Broker Administration perspective is the part of the Message Brokers Toolkit that is used for the administration of any broker domains that are defined to the Message Brokers Toolkit. This perspective is also used for the deployment of message flows and message sets to brokers in the defined broker domains.

The Broker Administration perspective also contains tools for creating message broker archive (bar) files. Bar files are used to deploy message flow application resources such as message flows and message sets.

The Broker Administration perspective also contains tools to help test message flows. These tools include Enqueue and Dequeue for putting and getting messages from WebSphere MQ queues.

3

Installation

This chapter details the software and system considerations for installing
WebSphere Message Broker. Instructions for performing an Express Installation
of the product and verifying the installation are provided.

The following tasks are discussed:

- ► Planning for installation
- ► Security issues
- ► Installing the required software
- ► Post installation tasks
- ► Verifying installation
- ► Installing product fix packs

3.1 Planning for installation

This section provides information about the required and optional software for installing WebSphere Message Broker V6.0 on Windows. The system requirements for these products, such as disk space and system memory, are also discussed.

For information about installing on other operating systems, operating system patches, and Java runtime environments, refer to the installation guide supplied in the product package or in the WebSphere Message Broker V6.0 product documentation: **Installing** → **Installation Guide**. See "Useful links" on page 317 for a link to the product documentation online.

3.1.1 Required software

The information provided in this section assumes that you want to perform a full WebSphere Message Broker installation on a Windows computer, including all the runtime components and the Message Brokers Toolkit. In order to perform a full installation of WebSphere Message Broker you must install all the required software, as listed in this section. All the required software listed in this section is automatically installed as a part of the Express Installation (see "Express Installation" on page 26 for more information).

It is possible to install only parts of WebSphere Message Broker on a machine, for example, installing just the broker component for the run time, in which case not all of the software mentioned in this section is needed.

For the latest information about supported versions of the required software, refer to the latest WebSphere Message Broker V6.0 readme file (see "Useful links" on page 317).

WebSphere MQ

WebSphere MQ provides the mechanism of communication between the Configuration Manager, the brokers, and the business applications, and must be installed on a machine on which any of the WebSphere Message Broker runtime components is installed. A separate WebSphere MQ queue manager is required for each broker that is created on a system. The Configuration Manager and User Name Server also require queue managers.

The Message Brokers Toolkit can communicate with the Configuration Manager without WebSphere MQ being installed, but you cannot create a broker domain or the Default Configuration on a machine without WebSphere MQ.

WebSphere MQ V6.0 is supplied in the WebSphere Message Broker V6.0 package but WebSphere MQ Version 5.3 Fix Pack 1 and later is also supported.

WebSphere MQ Explorer

WebSphere MQ Explorer it is a graphical administration tool for creating, modifying, and deleting WebSphere MQ resources and for monitoring the WebSphere MQ network. WebSphere MQ Explorer is an optional component of WebSphere MQ, although it is installed by default when you run the Typical installation of WebSphere MQ.

WebSphere MQ Explorer is available on Windows and Linux (x86 platform) only, which are the same operating systems on which the Message Brokers Toolkit is available.

WebSphere Eclipse Platform

The WebSphere Eclipse Platform V3.0.1 is a prerequisite software product for WebSphere MQ Explorer. If you are installing WebSphere MQ manually, install WebSphere Eclipse Platform before installing WebSphere MQ.

ODBC Drivers for Cloudscape

The Configuration Manager uses an embedded Derby database for its internal configuration repository. On Windows the Open Database Connectivity (ODBC) Drivers for Cloudscape™ enable this embedded Derby database to be used as a broker database.

This configuration is supported for test and evaluation purposes only. Only the embedded Derby database associated with the Configuration Manager can be used with WebSphere Message Broker. For production environments, an alternative supported broker database must be used.

WebSphere Message Broker run time

The WebSphere Message Broker run time is the product code that enables the creation of resources that exist at run time, including the Configuration Manager, brokers, and User Name Server. It also includes the message transformation services for WebSphere Message Broker, which are not available in the WebSphere Event Broker run time.

Message Brokers Toolkit

The Message Brokers Toolkit is a graphical user interface built on the Rational® Application Developer platform. The Message Brokers Toolkit is used for development and administrative purposes.

WebSphere Message Broker developers use the Message Brokers Toolkit to create message flow applications to process business logic. These development resources are then deployed to the runtime components in the broker domain using the Message Brokers Toolkit. The Message Brokers Toolkit is also used to administer the runtime components and manage the configuration of broker domains.

The Message Brokers Toolkit can be installed without the runtime component, for development of message flow applications and the administration of remote broker domains.

The Message Brokers Toolkit is available on Windows and Linux (x86 platform). No prerequisite products are required for the Message Brokers Toolkit. If other products that use the Rational Application Developer platform are installed on a machine, the Message Brokers Toolkit uses the same installation of Rational Application Developer, as long as it is a compatible version. Refer to the production documentation and latest readme for information about the Rational Application Developer level and updates.

3.1.2 Optional software

In addition to the required software, several optional products can be installed. These products are described in this section.

Rational Agent Controller

Rational Agent Controller (RAC) Version 6.0.0.1 is used by the Message Brokers Toolkit during message flow debugging. If you want to use the Flow Debugger, you must install Rational Agent Controller on every machine on which a broker is running and where debugging might be required.

When you install Rational Agent Controller on the machine on which a broker is installed, you are installing a server that communicates with the Rational Agent Controller client that is embedded in the Message Brokers Toolkit.

Because the base level of Rational Application Developer changes over the life cycle of the product, the level of Rational Agent Controller required to work with WebSphere Message Broker is likely to change. Refer to the production documentation and the latest readme for information about the Rational Agent Controller level and updates.

DB2 Universal Database Enterprise Server

DB2 Universal Database is supported as a repository for broker configuration data. DB2 Universal Database Enterprise Server Version 8.2 is supplied in the

WebSphere Message Broker V6.0 package. Other DB2 Universal Database V8.2 editions are also supported in production environments.

On Windows, it is not necessary to install DB2 Universal Database in a WebSphere Message Broker test or evaluation environment because the embedded Derby database used by the Configuration Manager can be used instead. However, DB2 Universal Database or another supported database must be used for a production environment.

When installing DB2 Universal Database Enterprise Server, three options are available: Compact installation, Typical installation, and Custom installation. The Compact installation is adequate for all the functionality that is required by WebSphere Message Broker V6.0. However, the installation instructions in this chapter describe the Typical installation because the Typical installation includes the graphical administration tools, such as the Control Center, which is used in Chapters 4–6.

> **Important:** To work with the exercises contained in Chapter 4, "Developing applications with ESQL" on page 47; Chapter 5, "Developing applications with Java" on page 97; and Chapter 6, "Developing applications with mappings" on page 135, you must have DB2 Universal Database installed on your system.

Other supported databases

Oracle, Sybase Adaptive Server Enterprise (ASE), and Microsoft SQL Server are all supported as broker databases in a production environment for WebSphere Message Broker V6.0. These products are not supplied with WebSphere Message Broker, so you must obtain your own copy if you want to use any of these for the broker database.

Any of these supported databases can also be used to contain user data that can be accessed at run time by message flows.

For information on supported levels of these databases, refer to the WebSphere Message Broker Installation Guide and system requirements documentation.

3.1.3 Software requirements

This section contains information on the software requirements for WebSphere Message Broker V6.0.

Disk space

Before installing WebSphere Message Broker, consider the amount of disk space that is required. This includes not only the amount of disk space that WebSphere Message Broker takes up after installation, but also space for

temporary files during the installation, space for the installation of required software, and space for post-installation configuration and development.

Disk space for WebSphere Message Broker

The amount of disk space that is required depends on the components that are to be installed and the working space that is required by those components.

The total space required for the installed product is likely to increase by the order of tens to hundreds of megabytes as resources are developed, logs are generated, and fix packs are applied. This increase in space requirement over time should be considered, but there are mechanisms to control the space requirements, such as cleaning up logs and archiving resources. When planning an installation of WebSphere Message Broker, ensure that you take these disk space requirements into account.

One of the main sources of additional space usage is the application of fix packs when the Message Brokers Toolkit is installed. The reason for this is that new levels of plug-ins for the Message Brokers Toolkit might be installed with a fix pack so several levels of the same plug-in might be present on a machine to enable compatibility with other products and updates in the Rational Application Developer environment.

Some additional space might be required for temporary files during the installation. Any temporary files are deleted when the installation is complete.

> **Important:** If disk space runs low during the Message Brokers Toolkit installation, problems can sometimes occur. The symptom of this with WebSphere Message Broker is missing parts of the Message Brokers Toolkit that are specific to WebSphere Message Broker, such as message sets and transformation nodes. If this symptom occurs, uninstall the Message Brokers Toolkit and clear extra space on the hard drive before reinstalling.

During installation, the wizard displays the actual requirements for the components that are being installed. Table 3-1 shows typical figures for components that are installed on Windows and Linux.

Table 3-1   Disk space that is required for components installed on Windows and Linux

Component	Space required on Windows	Space required on Linux
Broker, Configuration Manager, and User Name Server	315 MB plus 300 MB temporary space	280 MB plus 300 MB temporary space

Component	Space required on Windows	Space required on Linux
Transformation Services (optional broker extension)	25 MB	20 MB
WebSphere Message Broker Toolkit	4.2 GB plus 1.5 GB temporary space	4.2 GB plus 1.5 GB temporary space

Installing only one or two of the runtime components does not significantly reduce the storage requirements over installing the entire WebSphere Message Broker run time.

Disk space for WebSphere MQ

An installation of WebSphere MQ, excluding WebSphere MQ Explorer, can take up to 25.2 MB of disk space. This figure can increase depending on the WebSphere MQ components that are created.

WebSphere MQ Explorer can require up to 40.5 MB of disk space.

Disk space for WebSphere Eclipse Platform

An installation of WebSphere Eclipse Platform requires approximately 1.5 GB for code and data.

Database disk space

The database products that are supplied with WebSphere Message Broker V6.0 require the following additional disk space:

► DB2 Universal Database Enterprise Server requires approximately 300 MB of disk space for a compact installation. A typical installation requires approximately 350-560 MB of disk space, and a custom installation requires approximately 260-600 MB of disk space.

► ODBC drivers for Cloudscape require approximately 105 MB of disk space on Windows.

If an additional database is installed on the system (for example, one of the other supported databases), significantly more disk space is required.

In addition to the installation disk space for databases, extra disk storage is needed for the configuration and running of databases for the broker repository and user databases. For the broker, the minimum disk space that is required for each database is 10 MB.

System memory

The minimum amount of system memory that is required for the WebSphere Message Broker V6.0 run time is 512 MB of RAM.

For running the Message Brokers Toolkit, a minimum of 512 MB of RAM is required on both Windows and Linux (x86 platform). For improved performance of the Message Brokers Toolkit, provide a minimum of 1 GB of RAM.

The system memory for the prerequisite software must also be considered. For production environments in particular, the provision of extra memory will help to improve performance of the WebSphere Message Broker components and the Message Brokers Toolkit.

3.2 Security issues

Security requirements must be considered before installing WebSphere Message Broker. Different forms of security control are available to cover different aspects of the product use. This section briefly discusses some of these aspects for Windows. For more detailed information, and information for other platforms, refer to the security information in the product documentation: **Security → Planning for security**.

3.2.1 User ID

There are a number of aspects to consider when deciding on a suitable user ID under which WebSphere Message Broker V6.0 is to be installed:

1. The permitted number of characters for the user ID varies according to operating system:
   – On Windows, the maximum password length is 12 characters.
   – On Linux and UNIX®, the maximum password length is 8 characters.

   Databases in the WebSphere Message Broker configuration might also contain additional restrictions, for example, DB2 Universal Database.

2. The case of the user ID should be consistent; use all lowercase or all uppercase letters.

3. On Windows, the Administrator user ID must not be used. Any attempt to use the Administrator user ID with the WebSphere Message Broker run time produces authorization errors.

4. On Windows, the user ID must be a member of the Administrators security group.

It is recommended that user ID and password combinations are valid on all of the operating systems that are involved in your WebSphere Message Broker configuration.

3.2.2 Other security issues

Security must be considered for the machines on which WebSphere Message Broker is installed and used. This section briefly discusses some of the other security issues for WebSphere Message Broker. For further information about users and privileges, refer to the WebSphere Message Broker documentation on security.

Database security

In a test environment, the same user ID and password combination can be used to access the database and to create WebSphere Message Broker components. This leads to simplicity in a test environment and in situations where a single user may be administering the database as well as the WebSphere Message Broker environment. As an example, the Default Configuration wizard, which sets up a basic broker domain environment for test and evaluation purposes, uses the user ID and password of the current user to create components and access the database for the broker.

In a more complex environment or a production environment, the security access for a database is important, particularly for access to databases storing business data. In these environments it is typical for a database administrator to determine permissions for the database. Often different user IDs are used for different tasks and have associated levels of access. Sufficient access is required to create and alter database entries for the broker database, and to perform operations in a user database if the message flow applications are required to do this.

For improved database security, ensure that any passwords are changed if the default user ID for the database product is used. For example, when using DB2, leaving the db2admin password as db2admin presents a potential security risk.

WebSphere MQ security

WebSphere Message Broker users must be members of the *mqm* WebSphere MQ security group. This security group enables the user to create WebSphere MQ components and to access queues and other WebSphere MQ resources needed to enable communication of the WebSphere Message Broker components.

SSL can be used to secure the connection between the Message Brokers Toolkit and Configuration Manager, as well as the channels connecting remote brokers.

3.3 Installing the required software

The information that is contained in this section assumes that the Launchpad on Windows is being used to install the required software. For instructions on the alternative methods of installing WebSphere Message Broker V6.0, refer to the Installation Guide.

3.3.1 The Launchpad

When you insert the product media (DVD or CD), the Launchpad starts automatically if autorun is enabled. If the Launchpad does not start automatically, or if you are using a downloaded copy of the product, double-click the **mqsilaunchpad.exe** file. This file is usually located in the root directory.

The Launchpad contains links to the following information:

► The Installation Guide, which contains a link to the most recent version of the Installation Guide that is available on the Internet

► The product readme file, which contains last minute, undocumented changes to the product, and a link to the most recent version of the readme file available on the Internet

► The Quick Tour, which is a graphical overview of the product

The Launchpad offers two types of installation:

► Express Installation
► Advanced Installation

Express Installation

Use the Express Installation if you want a full installation of WebSphere Message Broker V6.0 including all required prerequisite software. The Express Installation requires minimal input. During installation, default values are used for the prerequisite software to make the installation as simple and as fast as possible. Refer to "Installing with the Express Installation" on page 28 for instructions on how to complete an Express Installation.

The Express Installation installs all of the software that is required for a minimum installation of WebSphere Message Broker:

► WebSphere Eclipse Platform V3.0.1
► WebSphere MQ V6.0
► ODBC Drivers for Cloudscape
► WebSphere Message Broker V6.0
► WebSphere Message Brokers Toolkit V6.0

The Launchpad detects whether any of the required software has already been installed on your system. If none of the required software exists on your system, the state for each product is shown as *Pending* and all check boxes are selected. If you clear any of the check boxes, the state of the corresponding software changes to *Required*.

If any of the products that have the state Required are not installed, you cannot complete the default configuration that is described later in this chapter.

It is possible that some of the software already exists on your system, but that the state of the software is shown as `Incorrect level`. In this situation, refer to the product documentation of the software for information about how to obtain the correct level. In this situation, you can use the Express Installation to install all of other required products, and then, if necessary, use the Advanced Installation to install the software that is at the wrong level.

To see more information about each of the required products, click the icon next to the product name. This expands the details about the selected product.

Advanced Installation

The Advanced Installation section of the Launchpad enables each required and optional software product to be installed manually. This is useful if you want to select the install location or install specific components for any of the products. It is also useful for resolving difficulties if required software installed on a system is at an unsupported level.

Advanced Installation allows you to install the software that is required for a minimum installation of WebSphere Message Broker V6.0, and also some optional products. The following products can be installed with the Advanced Installation:

Required products:

► WebSphere Eclipse Platform V3.0.1
► WebSphere MQ V6.0
► ODBC Drivers for Cloudscape
► WebSphere Message Broker V6.0
► WebSphere Message Brokers Toolkit V6.0

Optional products:

► Rational Agent Controller
► DB2 Universal Database Enterprise Server Edition V8.2
► Oracle
► Sybase
► Microsoft SQL Server 2000

The Launchpad detects whether any of the required software already exists on your system. If none of the required software has been installed on your system, the state for each product is shown as Required.

The Launchpad also displays the state of the optional products, initially shown as `Not Installed`. To install one of the optional products, click the associated plus icon. This brings up details about the product and, for Rational Agent Controller and DB2 Universal Database, provides the option to start installing the product.

Although the Advanced Installation displays Oracle, Sybase, and Microsoft SQL Server, these three products cannot be installed through the Launchpad. They are not included with the WebSphere Message Broker V6.0 package, and need to be installed manually if they have been purchased separately.

The Advanced Installation is useful if you want to install only one of the required components. For example, if you have a machine on which you want to create a remote broker, you will need only the WebSphere Message Broker runtime components, and not the Message Brokers Toolkit.

3.3.2 Installing with the Express Installation

This section provides step-by-step instructions for installing WebSphere Message Broker and the related required software using the Express Installation. These instructions are basic because the Express Installation is designed to be simple to use with minimal intervention.

Before you start, ensure that you have read the software requirements and security issues that are discussed in this chapter.

To start the Express Installation:

1. In the Launchpad, navigate to the Express Installation page. Open the Express Installation page by clicking the button in the top left corner.

2. Ensure that all check boxes are selected. If none of the software exists on your system, the state for each product is *pending*.

3. Click **Launch Express Installation for WebSphere Message Broker**. This starts the installation of the required software.

While the required software is being installed, a few panels are displayed to prompt for some user input. Accept all the default values and click **Next** where appropriate. Selecting the default values is the simplest option, but you can choose custom values if, for example, you want to change the installation location of the Message Brokers Toolkit to a different drive instead of the default C drive.

Additional information

Here we provide some additional information:

1. When installing WebSphere Message Broker on some Windows machines, the dialog shown in Figure 3-1 might be displayed. Click **Open** to start the installer. This dialog might be displayed several times during the installation.
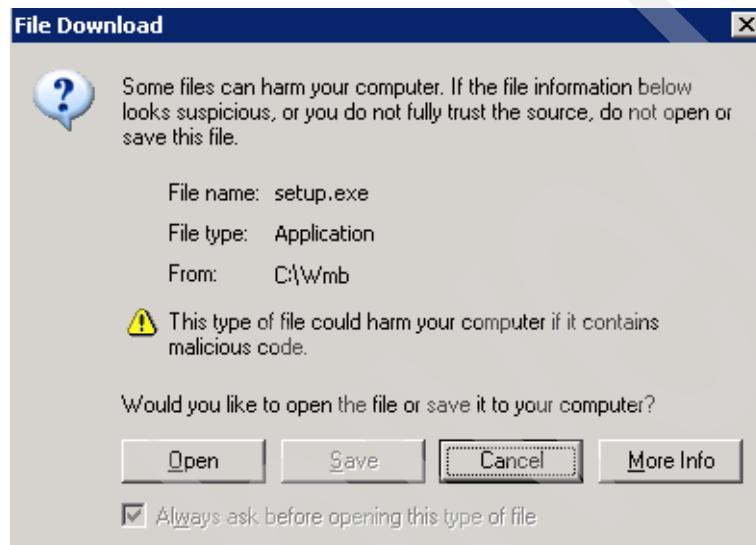
Figure 3-1   File download warning dialog

2. The progress of the installation can be tracked by referring to the product states on the Express Installation page of the Launchpad. See Figure 3-2 on page 30.

> **Tip:** The Launchpad can change the state of a product from Pending to Installed before the installer for that product has completely finished. Wait for each installer to finish before attempting to use either the Launchpad or the newly installed product.
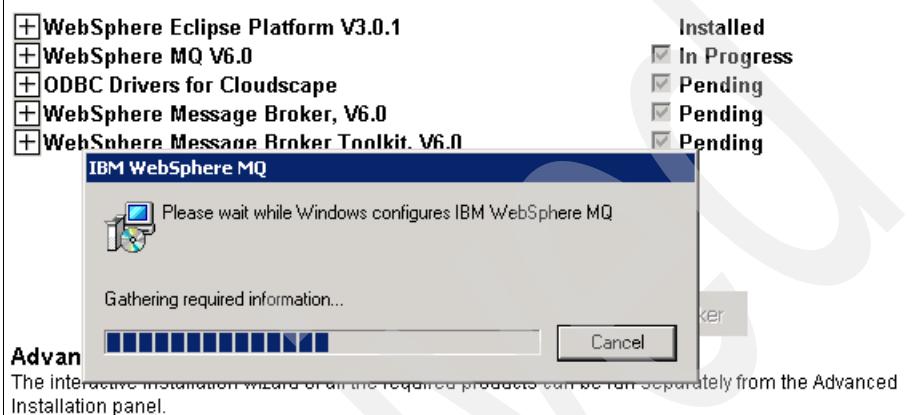
Figure 3-2   The Launchpad during an Express Installation

Tip: There is a known problem with the Message Brokers Toolkit installation where the Launchpad reverts to saying Pending part way through the installation. The Launchpad must be reopened to check that the Message Brokers Toolkit has been installed.

Do not click the Launchpad while an installation is taking place.

If errors occur during installation, see "Locating error information" on page 242 for help information.

3.3.3 Installing with the Advanced Installation

This section provides brief step-by-step instructions for installing WebSphere Message Broker and the related software using the Advanced Installation. With the Advanced Installation the product installations are manual and require intervention. Options need to be selected based upon your own configuration requirements. If you prefer to have a simple configuration with minimal intervention refer to the previous section for the Express installation.

To install DB2 Universal Database for use with the examples provided in this book, you will need to install DB2 Universal Database manually.

Before you start, ensure that you have read the software requirements and security issues that are discussed earlier in this chapter.

To install the products using the Advanced Installation:

1. In the Launchpad, open the Advanced Installation page by clicking the button in the top left corner.
2. Expand the product that you want to install. This brings up details about the product and, for RAC and DB2, provides the option to start installing the product.
3. Click the button to install the product.

Oracle, Sybase, and Microsoft SQL Server are not included with the WebSphere Message Broker package; you must obtain your own copy before you can install any of these products. Although there is an option to install these products through the Launchpad, even if you have your own copy of these products, you cannot use the Launchpad to install them.

Installing WebSphere Eclipse Platform

WebSphere Eclipse Platform is required to run WebSphere MQ Explorer, the graphical administration tool for WebSphere MQ. If you want to use WebSphere MQ Explorer, you must install WebSphere Eclipse Platform before you install WebSphere MQ.

To install the WebSphere Eclipse Platform, follow the instructions for installing with the Advanced Installation and accept the default options (the only required user input is to specify a location for the installation).

Installing WebSphere MQ

WebSphere MQ V6.0 facilitates messaging between the Configuration Manager, the brokers, and business applications.

Before you start, if you want to use WebSphere MQ Explorer, ensure that WebSphere Eclipse Platform V3.0.1 is installed.

To install WebSphere MQ, follow the instructions for installing with the Advanced Installation and, when prompted, enter your responses. By using this Advanced Installation, you are able to complete an installation of WebSphere MQ that is tailored to match your requirements.

If you manually install WebSphere MQ using the Advanced Installation you can create the Default Configuration for WebSphere MQ. The WebSphere MQ

Default Configuration can be used to learn about WebSphere MQ and to verify the WebSphere MQ installation. When the first part of the installation is complete, the installer launches a new wizard to create the WebSphere MQ Default Configuration. Follow the instructions in the wizard and select the appropriate options for your network configuration.

For more information about WebSphere MQ, refer to the WebSphere MQ Quick Beginnings book for Windows. Also see "Useful links" on page 317 for a link to the WebSphere MQ documentation, including the Quick Beginnings books for all platforms.

Installing Rational Agent Controller

Rational Agent Controller (RAC) is required for message flow debugging in the Message Brokers Toolkit.

Before you start:

1. Ensure that any Eclipse-based products and tools on the machine are closed.

2. Ensure that you have installed all of the required software (see "Required software" on page 18).

3. Ensure that an existing JRE is available on the system on which you are installing Rational Agent Controller.

   On Windows, the following JREs are supported:

   – IA32 J2RE 1.4.1 IBM Windows 32 (build cn1411_20040301a)
   – J2RE 1.4.2 IBM Windows 32
   – Sun™ Java™ 2 Standard Edition (build 1.4.2_04b05)

   You can use the JRE from either the Message Brokers Toolkit or WebSphere MQ installations. The default location of the Message Brokers Toolkit JRE is C:\Program Files\IBM\MessageBrokersToolkit\6.0\eclipse\jre\bin\java.exe, as shown in Figure .
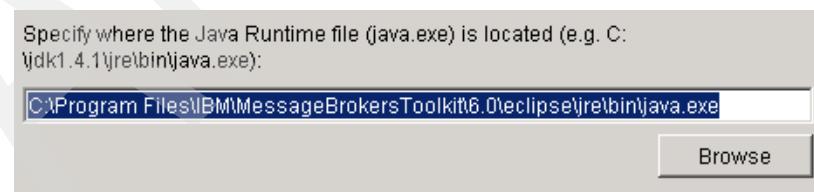
Figure 3-3   Location of the Message Brokers Toolkit java.exe file

To install Rational Agent Controller, follow the instructions for installing with the Advanced Installation and, when prompted, select the following options:

► When specifying the location of the Java runtime file, browse to a directory that contains the java.exe file. Ensure that you include the file name in the location (see Figure on page 32).

► Select to **Disable** security if appropriate. This is suggested for test and evaluation systems.

Installing DB2 Enterprise Server V8.2

DB2 Universal Database is a supported relational database manager for use with the broker component of WebSphere Message Broker V6.0.

> **Important:** To complete the exercises in Chapters 4, 5, and 6 you must install DB2 Universal Database.

Before you start, ensure that you have stopped all WebSphere Message Broker V6.0 components, such as brokers, the Configuration Manager, and the User Name Server.

To install DB2 Universal Database, follow the instructions for Installing with the Advanced Installation and, when prompted, select the following options:

► **Typical** installation is recommended to provide an installation that is suitable for use with WebSphere Message Broker V6.0.

► Provide a user ID and password. This can be the same user account as the user account under which WebSphere Message Broker V6.0 was installed. It is recommended that the user ID is no more than 8 characters to prevent any later problems relating to restrictions.

► On the "Specify a contact for health monitor notification" page, select **Defer the task until after installation is complete**.

3.4 Post installation tasks

After a full installation of WebSphere Message Broker V6.0, ensure that the following products are running using the instructions in this section:

► WebSphere MQ
► DB2 Universal Database

If Rational Agent Controller is installed, this must also be running in order to use the message flow debugger.

3.4.1 WebSphere MQ service

To verify that the WebSphere MQ service is running, perform any of the following tasks:

► Check the system tray icon to see whether WebSphere MQ is running. Figure 3-4 shows the system tray icon for WebSphere MQ in the started state (left icon) and the stopped state (right icon).

 If WebSphere MQ is not running, right-click the system tray icon, then click **Start WebSphere MQ**.

► Check the Services list for the IBM MQSeries® status. The status should be `Started`. To open Services, click **Start** → **Control Panel** → **Administrative Tools** → **Services**.

 If WebSphere MQ is not running (no status is displayed, as shown in Figure 3-6 on page 35) in the Services window, right-click **IBM MQSeries** then click **Start**.

Figure 3-4   System tray icons shows started and stopped state of WebSphere MQ

3.4.2 DB2 Universal Database

To verify that DB2 Universal Database is running, perform any of the following tasks:

► Check the system tray icon to see whether DB2 Universal Database is running. Figure 3-5 on page 35 shows the system tray icon for DB2 Universal Database in both the started state (left icon) and the stopped state (right icon).

 If DB2 Universal Database is not running, right-click the system tray icon, then click **Start (DB2)**.

► At a command prompt type `db2start`. If DB2 Universal Database is running, the message `The database manager is already active` is displayed.

 If DB2 Universal Database is not running, issuing the **db2start** command starts DB2 Universal Database. Note that if the user ID and password that you are logged on with does not have the appropriate authorities, the message `The DB2 service failed to log on` is displayed.

► Check Services for the status of the DB2 - DB2-0 service, which should be `Started`. To open Services, click **Start** → **Control Panel** → **Administrative Tools** → **Services**.

If DB2 is not running (no status is displayed), in the Services window, right-click **DB2 - DB2-0** then click **Start**.

Figure 3-5   System tray icons showing the started and stopped states of DB2

3.4.3 Rational Agent Controller

To verify that Rational Agent Controller (RAC) is running, check the Services list for the IBM Rational Agent Controller status, which should be `Started`. To open Services, click **Start** → **Control Panel** → **Administrative Tools** → **Services**.

If RAC is not running (no status is displayed, as shown in Figure 3-6), in the Services window, right-click **IBM Rational Agent Controller**, then click **Start**.

Figure 3-6   Part of Services window

3.5 Verifying the installation

This section describes how to verify that your installation of WebSphere Message Broker V6.0 was successful. To assist in rapidly verifying the installation two wizards are provided:

▶ Default Configuration wizard, for creating a simple broker domain.
▶ Prepare the Samples wizard, for setting up and deploying samples.

Use the instructions in the following sections to verify your WebSphere Message Broker installation.

3.5.1 Creating the default configuration

In the Message Brokers Toolkit, message sets and message flows can be developed without first creating any runtime components. However, you cannot test or run these applications, or run any of the provided samples, until you have configured the runtime components and created a broker domain.

The Default Configuration wizard creates the following components and resources on the system to provide a simple broker domain that can be used for test purposes:

► A Configuration Manager called WBRK6_DEFAULT_CONFIGURATION_MANAGER

► A broker called WBRK6_DEFAULT_BROKER

► A queue manager called WBRK6_DEFAULT_QUEUE_MANAGER, shared by the Configuration Manager and the broker

► A listener on the queue manager on port 2414

► A broker database called DEFBKDB6

Use the instructions in the following section to create a default configuration using the Default Configuration wizard. For instructions on how to manually create a simple broker domain see "Creating a broker domain" on page 206.

> **Important:** If you plan to complete the exercises in Chapter 4, "Developing applications with ESQL" on page 47; Chapter 5, "Developing applications with Java" on page 97; and Chapter 6, "Developing applications with mappings" on page 135, ensure that you have installed DB2 Universal Database prior to running the Default Configuration wizard. The Default Configuration wizard uses a DB2 Universal Database for the broker database if it is available, or the embedded Derby database if it is not. You might see problems if you use the Default Configuration created with Derby and then install DB2 Universal Database.

Running the Default Configuration wizard

Use the following instructions to create a default configuration:

1. Open the Message Brokers Toolkit. If the WebSphere Message Broker Welcome screen is not displayed, click **Help** → **Welcome**.

2. Click the Getting Started icon shown in Figure 3-7. The Getting Started page of the Welcome screen opens.

Figure 3-7   Getting Started icon on Message Brokers Toolkit Welcome page

3. Click the Create the Default Configuration icon (Figure 3-8). The Create the Default Configuration topic in the product documentation opens in the Message Brokers Toolkit help system.

4. Click the **Start the Default Configuration wizard** link to start the wizard.

Figure 3-8   The Default Configuration wizard icon in the Message Brokers Toolkit

> **Important:** In the Default Configuration wizard, you are asked to enter your password. Ensure that you enter the password carefully because the wizard does not verify the password. Any mistakes made when entering the password can lead to problems when creating the configuration, such as the broker failing to be created or started.

When the Default Configuration wizard has completed successfully, a simple broker domain is configured on the system. The connection to the broker domain is displayed in the Broker Administration perspective. This configuration can be used for running samples supplied with WebSphere Message Broker and for completing the examples provided in this book in Chapters 4, 5, and 6.

If any problems occur when running the Default Configuration wizard, refer to the Problem Determination section, 8.4.1, "Default Configuration wizard problems" on page 291.

> **Tip:** The Default Configuration wizard sets the broker and Configuration
> Manager Windows services to automatic. This means that when the system
> with the Default Configuration on it is restarted, the broker and Configuration
> Manager are started automatically. Problems can occur, however, if the
> components and software that they depend on are not started automatically,
> and errors such as the broker not being able to connect to its database or
> queue manager may be displayed in the Application log (see "Windows
> Application log" on page 253).
>
> To overcome this problem set the WebSphere MQ and DB2 Universal
> Database (if applicable) services to automatic start using Services:
>
> **Start → Control Panel → Administrative Tools → Services → IBM MQ
> Series → Properties**
>
> **Start → Control Panel → Administrative Tools → Services → DB2 -
> DB2-0 → Properties**
>
> Change the *startup type* to automatic.
>
> Ensure that the WBRK6_DEFAULT_QUEUE_MANAGER is set to automatic:
> 1. **Start → Programs → IBM WebSphere MQ → WebSphere MQ Explorer**
> 2. Select **WBRK6_DEFAULT_QUEUE_MANAGER**, then right-click and
>   select **Properties**.
> 3. Ensure Startup on the General tab is Automatic.
>
> Alternatively, set the Default broker and Configuration Manager to manual in
> the Services dialog.

3.5.2 Running the Getting Started samples

In the Message Brokers Toolkit a selection of samples is provided to
demonstrate different areas of functionality of the product and how to use them.
Included in the samples are the Getting Started samples, which are very basic
samples designed to verify that an installation of the product was successful. In
WebSphere Message Broker the Getting Started Samples are:

► Pager samples
► Scribble sample
► Soccer sample

The Pager samples are provided for you to use to verify your installation on
WebSphere Message Broker. The Soccer sample is the only sample supplied for

the WebSphere Event Broker and can be used to verify the WebSphere Event Broker installation.

To access the Getting Started samples, on the Getting Started page of the Welcome screen click the sample icon (Figure 3-9) to open the Message Brokers Toolkit Samples Gallery.

Figure 3-9   Sample icon from the Getting Started page

Follow the instructions in the Samples Gallery to set up and run the sample. The link provided to set up the sample launches the Prepare the Samples wizard. The Prepare the Samples wizard creates the required resources for the sample, such as WebSphere MQ queues, and deploys the sample to a specific execution group on the broker in the Default Configuration.

> **Important:** The samples can be imported and deployed using the Prepare the Samples wizard only if the Default Configuration has been created. The Default Configuration must have been created using the Default Configuration wizard. You cannot deploy the samples using the Prepare the Samples wizard if you have created the domain manually, even if you have used the same component names.

The Samples Gallery displays the selection of others samples available for WebSphere Message Broker and information about what they demonstrate and how to use them.

3.6 Next steps

After WebSphere Message Broker has been successfully installed and configured, you can begin to develop message flow applications and administer the system. Chapters 4, 5, and 6 provide instructions for developing and testing simple message flow applications:

► Chapter 4, "Developing applications with ESQL" on page 47
► Chapter 5, "Developing applications with Java" on page 97
► Chapter 6, "Developing applications with mappings" on page 135

Chapter 7, "Administration" on page 205, provides information about administering and extending a broker domain.

3.6.1 Navigating the Message Brokers Toolkit

The Message Brokers Toolkit is the graphical user interface for the WebSphere Message Broker V6.0 products that runs on Windows and Linux.

The Message Brokers Toolkit is the development environment for message flow applications and associated resources such as ESQL, Java, mappings, and message definitions. These message flow applications are deployed to the runtime components using the Message Brokers Toolkit.

The Message Brokers Toolkit is also used for administrative tasks such as configuring the properties and components in the broker domain, and publish/subscribe. Some administrative tasks such as creating a broker or stopping a Configuration Manager cannot be performed in the Message Brokers Toolkit and must be performed using the Command Console.

Perspectives

The Message Brokers Toolkit comprises a collection of perspectives. A perspective (the full workbench window) is a collection of views that can be moved and re-sized. Figure 3-10 on page 41 shows the Broker Application Development perspective of the Message Brokers Toolkit, which includes the Resource Navigator view (upper left), the Outline view (lower left), the Editor View (upper right, currently contains ESQL_Simple.msgflow), and the Problems view (lower right).
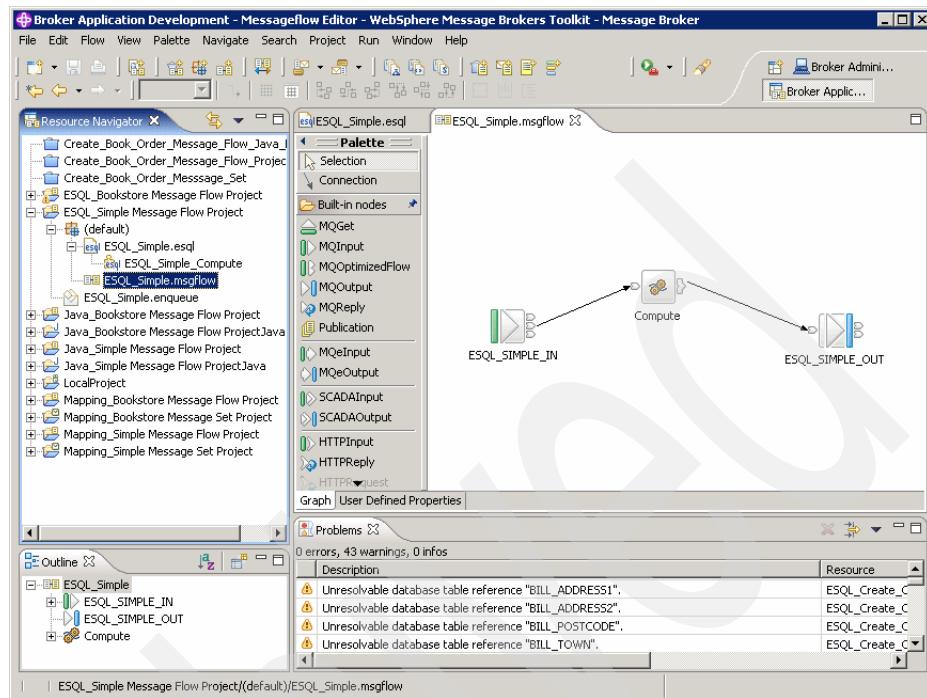
Figure 3-10   Broker Application Development perspective

Changing perspectives

The first time that the Message Brokers Toolkit is opened, the Broker Application Development perspective is displayed. To open a new perspective:

1. Click **Window** → **Open Perspective** → **Other...**.

2. From the list, click the name of the perspective that is to be used, then click **OK**.

When a new perspective opens, a button for the perspective is added to the toolbar on the top right of the Message Brokers Toolkit window. Figure 3-11 on page 42 shows the buttons for the Broker Application Development perspective and the Broker Administration perspective. In the figure, the Broker Application Development perspective is currently in use.

To change perspectives using the buttons:

▶   If you have previously opened the perspective, click the perspective button associated with that perspective, for example, Broker Administration perspective.

► If you have not previously opened the perspective, click the Open a perspective icon (with a cross in the top right corner). Then click **Other...** to bring up a list of available perspectives.

Figure 3-11   The perspectives buttons in the Message Brokers Toolkit

Tip: To make the perspectives buttons smaller and easier to view:

1. Right-click the toolbar near the perspectives button.
2. Click **Show Text**.

This hides the text associated with the buttons, allowing more perspective buttons to be displayed. The perspective buttons available depend upon the perspectives you have recently opened.

Table 3-2 shows which perspective is needed for each of the most common tasks that are performed in the Message Brokers Toolkit.

Table 3-2   Perspectives to use for common tasks

Task	Perspective
Developing message flows	Broker Application Development perspective
Developing message sets	Broker Application Development perspective
Deploying and testing message flow applications	Broker Administration perspective
Debugging message flow applications	Debug perspective
Managing broker domains	Broker Administration perspective
Configuring publish/subscribe applications	Broker Administration perspective
Connecting the Message Brokers Toolkit to a database	Data perspective

Task	Perspective
Developing a Java class for a JavaCompute node or user-defined node	Java perspective

Perspectives have a default layout that displays those tools most useful for the tasks that are regularly performed in that perspective. Perspectives can be customized to enable changes to the layout and content of each perspective. You can also reset a perspective to its default layout at any time. For further information on customizing the perspectives, see the WebSphere Message Broker product documentation: **Reference** → **Workbench** → **Perspectives**.

3.7 Installing product fix packs

The instructions in this section describe how to install a fix pack on top of an installed version of WebSphere Message Broker V6.0. When available, fix packs can be downloaded from the Web and include additional function and fixes to problems. These usually contain updates to both the WebSphere Message Broker run time and the Message Brokers Toolkit. See the Useful links section in Appendix A, "Getting help" on page 307.

3.7.1 Before you install a fix pack

Before installing a fix pack:

► Ensure that you are logged on with the same user ID that was used to install WebSphere Message Broker V6.0.

► Ensure that all WebSphere Message Broker components are stopped, including the Configuration Manager, the User Name Server (if applicable), and all brokers on the system.

► Ensure that all WebSphere Message Broker files, such as the product readme file, are closed.

► Close the Message Brokers Toolkit.

3.7.2 Installing a fix pack

To install a fix pack, follow the instructions that are provided with the fix pack, and read the contents of the memo.ptf file and any readme files.

3.8 Updates to the Message Brokers Toolkit

Many of the updates required to add new function or fix problems in the Message Brokers Toolkit can be accessed in the same way as performing a documentation update. Additionally, fix packs may be used as a method to update the toolkit.

> **Important:** A connection to the Internet is required to install updates to the Message Brokers Toolkit using the methods described in this section.

Updates are available using these options in Software Updates on the Help menu in the Message Brokers Toolkit:

- ▶ IBM Rational Product Updater
- ▶ Find and Install

To use the IBM Rational Product Updater use the instructions below:

1. On the Installed Products tab, select **WebSphere Message Brokers Toolkit**.

2. Select **Find Updates**. The IBM Rational Product Updater will search for WebSphere Message Broker updates.

3. If you are prompted, you must upgrade to the latest level of the IBM Rational Product Updater. Follow the on screen instructions to do this.

4. If fixes or updates are available (as shown in Figure 3-12 on page 45), then close the Message Brokers Toolkit.

5. Click the **Install Updates** button in the IBM Rational Product Updater.

6. Select **I agree to the terms in the license agreement** and then click **OK**. The selected features are then installed.

7. Restart the Message Brokers Toolkit.
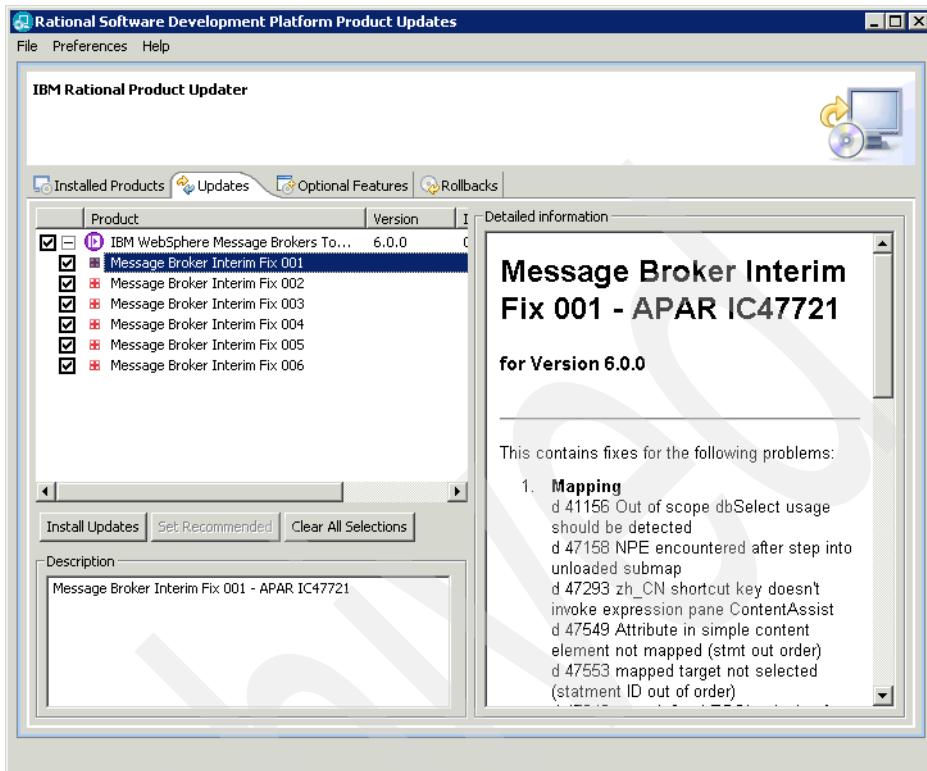
Figure 3-12   Installing Message Brokers Toolkit updates

To use the Find Updates method use the following instructions:

1. In the Message Brokers Toolkit, click **Help** → **Software Updates** → **Find and Install**. The Install/Update wizard opens.

2. In the Install/Update wizard (shown in Figure Ê on page 44), select **Search for updates of the currently installed features**, then click **Next**.

   The wizard searches for update information from the configured update sites, including the Message Brokers Toolkit Update Site.

3. Select available updates from those displayed in the wizard, then click **Next**.

4. Select **I agree to the terms in the license agreement** and then click **Next**.

5. Click **Finish**.

6. A warning message may be displayed; click **Install** to continue with the install of the updates.

7. Restart the Message Brokers Toolkit.
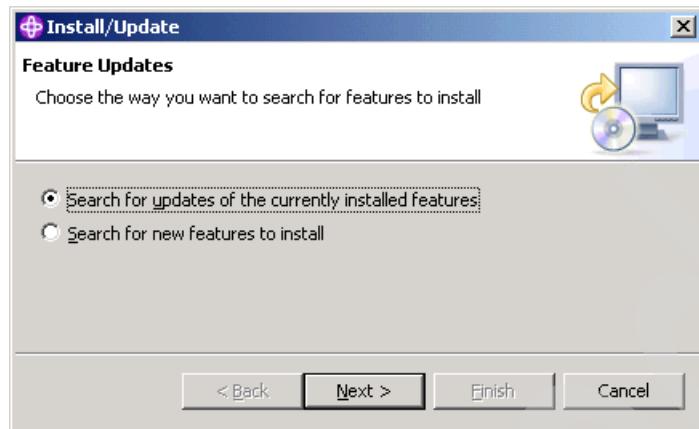
Figure 3-13 Find and Install

> **Tip:** Different updates may be seen between the IBM Rational Product
> Updater and the Find and Install wizard. This can be caused by problems with
> the Message Brokers Toolkit configuration. The Find and Install wizard
> provides an Error Details button for viewing errors that have been found and
> the IBM Rational Product Updater displays errors if they occur.

Further information about performing Message Brokers Toolkit and
documentation updates can be found in "Receiving automatic updates" on
page 315 and "Receiving manual updates" on page 315.

4

Developing applications with ESQL

This chapter describes how to develop message flow applications in the Message Brokers Toolkit using ESQL to define the logic of the message flows.

The following topics are discussed:

► Defining the logic of a message flow using ESQL
► ESQL and the ESQL editor in the Message Brokers Toolkit
► Inserting data into a database using a message flow
► Transforming a message from one XML structure to another

47

4.1 Developing message flow applications with ESQL

A message flow application is a program that processes messages in the broker. Message flow applications can transform messages between different formats, generate new messages based on other messages, and route messages according to the message's content or according to how the message flow is configured.

4.1.1 Messages in WebSphere Message Broker

When a message flow gets a message, the input node (for example, an MQInput node) parses the message into the message's logical tree structure. Part of this tree structure is the message tree, which contains the message properties, the message headers, and the message body.

The body of the message is a hierarchical tree of elements, or message fields. The message flow can interpret the hierarchy of elements in the message body only if the input node has been configured to use the correct parser. The messages in this book are all in XML format, so the input node must be configured to use the XML parser to interpret input messages. If the input node is not correctly configured, the message body is treated as a binary large object (BLOB). A BLOB is a single entity that cannot be navigated using Extended Structured Query Language (ESQL) that has been written to process XML messages.

For more information about the message tree and how it is populated by a message flow, see the product documentation: **Developing applications** → **Developing message flow applications** → **The message tree**.

4.1.2 The Message Flow editor

The graphical Message Flow editor in the Message Brokers Toolkit (Figure 4-1 on page 49) enables you to build message flows by clicking one of the supplied message flow nodes on the node palette (on the left of the Message Flow editor) and placing it on the *canvas* (the empty white area to the right of the node palette). By combining different nodes, connecting them together, and configuring their properties, you can quickly create a small program—a message flow.
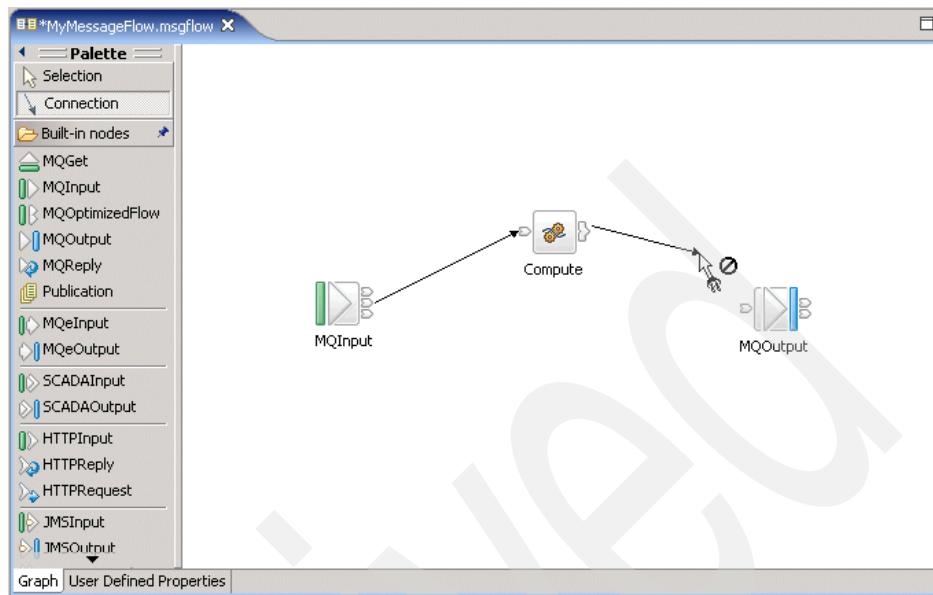
Figure 4-1   The Message Flow editor

> **Tip:** The first time that you open the Message Flow editor, the node palette is
> hidden. To show the node palette permanently, while the palette is hidden,
> click the small arrow at the top of the palette (Figure 4-2).

Figure 4-2   Opening the node palette

The built-in nodes that are supplied with WebSphere Message Broker can
perform a certain amount of processing and logic by themselves; with the
graphical Message Flow editor, you can use them to create complete message
flows that perform limited processing of messages.

However, to build useful message flows that suit the requirements of your business, you need to customize the message flows using ESQL, Java, or the graphical mapping tools in the Message Brokers Toolkit. The method you use depends the requirements of the message flow (for example, a Mapping node requires an external message definition, while ESQL is good for interacting with databases), and on your skills and programming experience. This chapter describes how to develop message flow applications in the Message Brokers Toolkit using ESQL to define the logic of the message flows.

4.1.3 ESQL and the ESQL editor

ESQL is based on Structured Query Language (SQL), which is commonly used to query relational databases like DB2 Universal Database. You can define the logic of message flows using ESQL by inserting ESQL code into built-in nodes that are supplied with WebSphere Message Broker, such as the Compute node, Database node, and Filter node. The ESQL is stored in a separate file, which you edit in the ESQL editor. The ESQL editor validates your ESQL and, while you are editing, you can get assistance by pressing Ctrl+Spacebar (or selecting **Content Assist** from the Edit menu) to open the code assist window, as shown in Figure 4-3.
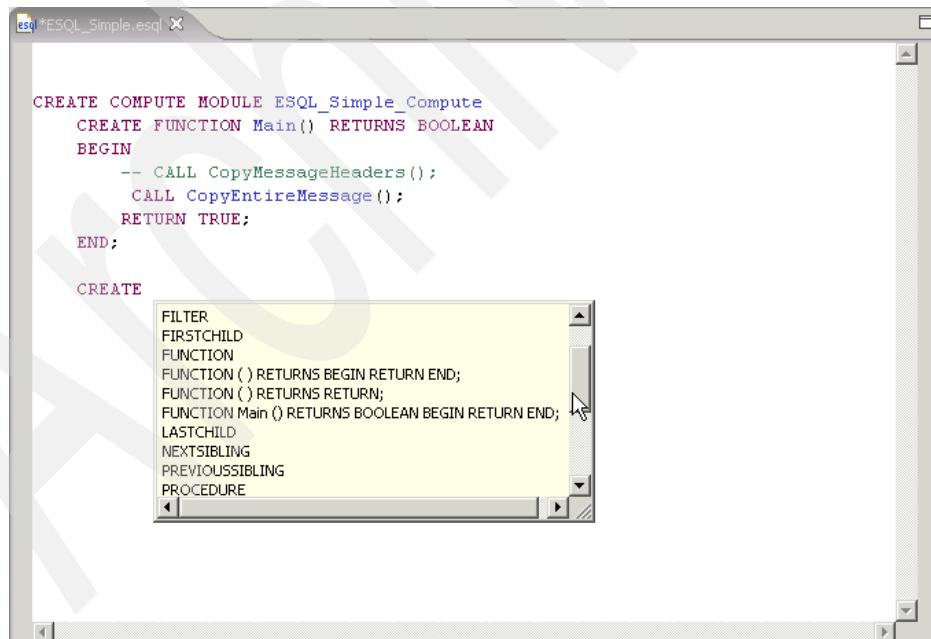
Figure 4-3   The ESQL editor with code assist

The message flow applications described in this chapter use ESQL in Compute nodes and Database nodes. The processing done by these nodes can be defined using ESQL to perform a range of tasks, including manipulating messages, accessing and updating database tables, and creating new messages. It is possible to reduce the amount and complexity of ESQL that you code by adding other nodes from the node palette that specialize in performing some tasks; for example, a Filter node specializes in routing messages according to their content, a DataInsert node specializes in inserting data into rows in database tables, and the RouteToLabel and Label nodes specialize in dynamically routing messages based on their content.

The message flows in this chapter demonstrate how to use ESQL so only the Compute and Database nodes are used.

4.1.4 Scenarios demonstrated in this chapter

This chapter focuses on how to define the logic of message flows with ESQL. We provide step-by-step instructions to create, deploy, and test two message flow applications:

▶ Simple message flow application

The Simple message flow application demonstrates how to build a very basic message flow. The ESQL_Simple message flow takes an XML input message from a WebSphere MQ queue, uses ESQL in a Compute node to build an XML output message that has the same contents as the input message, then puts the output message on another WebSphere MQ queue.

▶ Bookstore message flow application

The Bookstore message flow application is based around the scenario of an online bookstore. The first message flow, ESQL_Create_Customer_Account, uses ESQL in a Database node to create accounts in a DB2 database table for new customers who have registered their details with the bookstore, for example, their contact details and delivery address. The second message flow, ESQL_Book_Order, uses ESQL in a Compute node to process an order that has been submitted by an online customer and create a response message to confirm the order with a unique order number.

You do not need skills or experience in coding ESQL to be able to create the message flow applications in this chapter because all the code is available to download from the Internet. For more information see Appendix B, "Code" on page 319.

4.1.5 Before you start

The instructions in this chapter assume that you have run the Default Configuration wizard to create the default configuration. However, you can create your own broker domain and substitute the component names when following the instructions.

For more information about the Default Configuration wizard see 3.5, "Verifying the installation" on page 35. For more information about administering components see "Starting the components" on page 213.

Ensure that the broker and the Configuration Manager are running.

Starting the broker and the Configuration Manager

You cannot start components from the Message Brokers Toolkit; you must start them from the command line. Enter all commands in a WebSphere Message Broker Command Console, which is a command window with additional WebSphere Message Broker Environment settings.

To start the Command Console click **Start** → **Programs** → **IBM WebSphere Message Brokers 6.0** → **Command Console**.

To start the broker, enter the following command in the Command Console, where WBRK6_DEFAULT_BROKER is the name of the broker in the default configuration:

`mqsistart WBRK6_DEFAULT_BROKER`

To start the Configuration Manager, enter the following command in the Command Console, where WBRK6_DEFAULT_CONFIGURATION_MANAGER is the name of the Configuration Manager in the default configuration:

`mqsistart WBRK6_DEFAULT_CONFIGURATION_MANAGER`

Open the Windows Event Viewer to check that the components have started without any problems. See 8.1.5, "Windows Event Viewer" on page 253, for information about how to access and view entries in the Windows Event Viewer.

4.2 Developing the Simple message flow application

Each message flow is stored in a message flow file with the extension .msgflow. The message flow file is, in turn, stored in a Message Flow project, along with

any associated ESQL files (.esql). Projects are containers that store files while you are working on them in the Message Brokers Toolkit.

When you have created the files that contain the message flow, add, connect, and configure the message flow nodes in the Message Flow editor. Deploy the message flow to the broker so that you can test it.

4.2.1 Creating the ESQL_Simple message flow

To create the files in which the message flow is stored:

1. Ensure that you are working in the Broker Application Development perspective. If not, switch to the Broker Application Development perspective: Click **Window** → **Open Perspective...** → **Broker Application Development perspective**.

2. In the Broker Application Development perspective, create a Message Flow project called ESQL_Simple Message Flow Project:

   a. Click **File** → **New** → **Message Flow Project**.

   b. In the Project Name field, type `ESQL_Simple Message Flow Project`, then click **Finish**.

      A new project called ESQL_Simple Message Flow Project is displayed in the Resource Navigator view at the top-left of the Message Brokers Toolkit window.

3. Create the ESQL_Simple message flow in the ESQL_Simple Message Flow Project:

   a. In the Resource Navigator view, click the **ESQL_Simple Message Flow Project** to highlight it.

   b. Click **File** → **New** → **Message Flow**. The New Message Flow wizard opens.

   c. Ensure that the value in the Project field is `ESQL_Simple Message Flow Project` (Figure 4-4 on page 54).
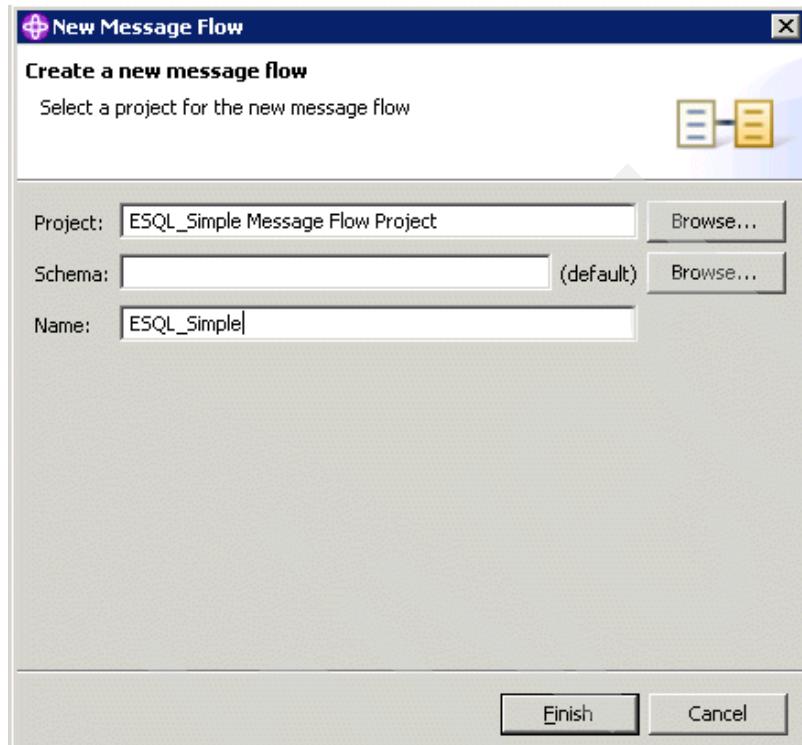
Figure 4-4   Creating the ESQL_Simple message flow

    d. Leave the Schema field empty so that the message flow is created in the default schema.

    e. In the Name field, type `ESQL_Simple`, then click **Finish**.

In the Resource Navigator view, a file called ESQL_Simple.msgflow is now displayed in the default schema of ESQL_Simple Message Flow Project. The ESQL_Simple.msgflow file opens automatically in the Message Flow editor.

For more information about schemas, see the product documentation: **Developing applications** → **Developing message flow application** → **Message flows overview** → **Broker schemas**.

Adding and connecting the ESQL_Simple nodes

Figure 4-5 on page 55 shows how the ESQL_Simple message flow looks in the Message Flow editor when you have added and renamed the nodes, and connected them together.
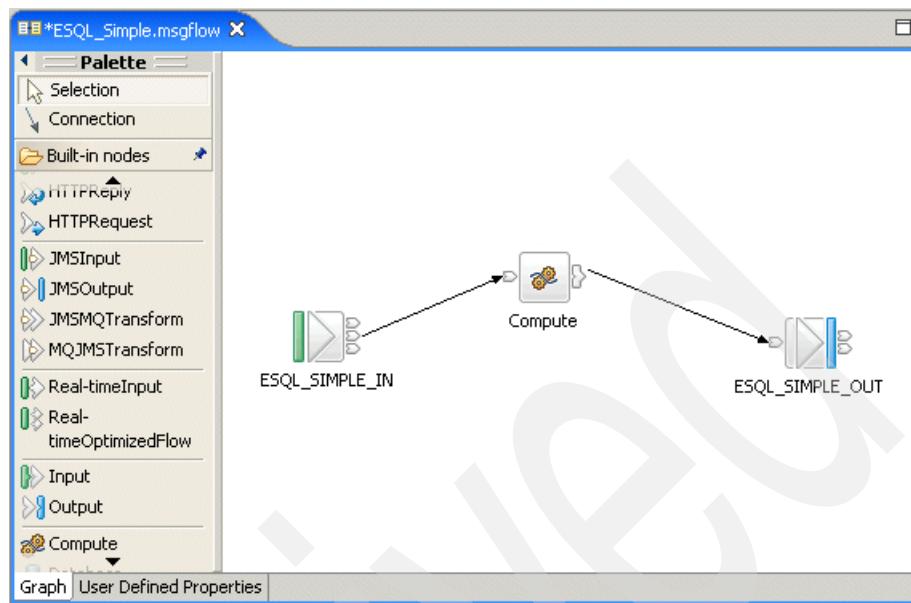
Figure 4-5   The ESQL_Simple message flow

To build the ESQL_Simple message flow:

1. Make sure that the Selection button (at the top of the node palette) is highlighted so that you can select nodes from the node palette.

2. Click the **MQInput** node to select it from the node palette, then click somewhere on the canvas (the white area to the right of the node palette) to start creating the message flow. The MQInput node is added to the canvas.

3. Add a Compute node and an MQOutput node to the message flow.

4. Rename each node as shown in Table 4-1 on page 56:

   a. Right-click the MQInput node, then click **Rename...**. The Rename Node dialog opens.

   b. Type `ESQL_SIMPLE_IN` then click **OK** (Figure 4-6 on page 56). The node on the canvas is renamed to ESQL_SIMPLE_IN.

   ESQL_SIMPLE_IN is also the name of the queue from which the MQInput node will get messages. Using the same name for both the queue and the node makes it easier for you to keep track of what queue to put the input message on when you are testing the message flow.

   c. Rename the Compute node and the MQOutput node.
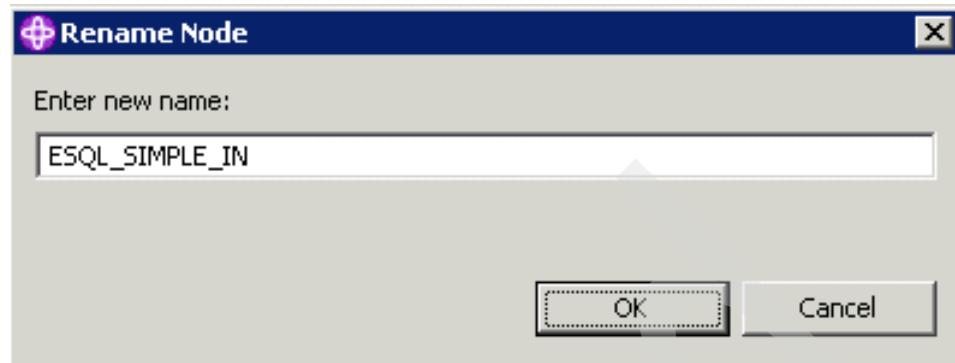
Figure 4-6   Renaming the MQInput node in the ESQL_Simple message flow

Table 4-1   The ESQL_Simple message flow nodes

Node type	Node name
MQInput	ESQL_SIMPLE_IN
Compute	Compute
MQOutput	ESQL_SIMPLE_OUT

5. Define the order in which the nodes process an input message by connecting them together as shown in Table 4-2 on page 57:

   a. Right-click the ESQL_SIMPLE_IN node, then click **Create Connection**. The Select Terminal dialog opens.

   b. In the Select Terminal dialog, click **Out**, then click **OK**. An arrow from the ESQL_SIMPLE_IN node follows the mouse pointer when you move the mouse because you have not specified which node to connect to.
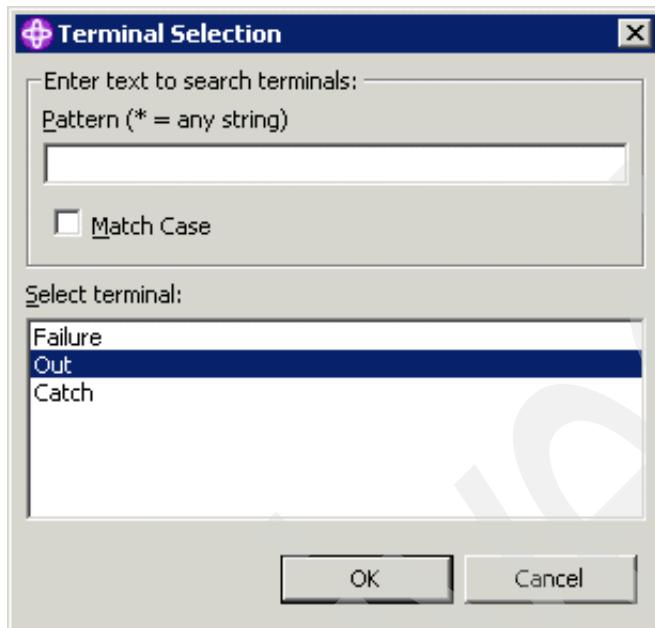
Figure 4-7   Selecting the Out terminal of the ESQL_SIMPLE_IN node

   c. Click the Compute node. The arrow connects the ESQL_SIMPLE_IN node to the Compute node.

   d. Connect the Out terminal of the Compute node to the ESQL_SIMPLE_OUT node. Ensure that you select the Out terminal on the Compute node and not, for example, Out1 or Out2.

Tip: Instead of using the Terminal Selection dialog, you can click the Connection button at the top of the palette to change to Connection mode. You can then directly click node terminals to create connections.

You must still use the Terminal Selection dialog to select the Out terminal of the Compute node because of the large number of output terminals available.

Remember to switch back to the Selection mode when you have finished making connections.

Table 4-2   Node connections in the ESQL_Simple message flow

Node name	Terminal	Connect to this node
ESQL_SIMPLE_IN	Out	Compute

Node name	Terminal	Connect to this node
Compute	Out	ESQL_SIMPLE_OUT

Saving and validating the ESQL_Simple message flow

To save the ESQL_Simple message flow, click **File → Save** or press Ctrl+S.

When you save a message flow file, the Message Flow editor validates the message flow. The ESQL_Simple message flow has two errors, as shown in Figure 4-8, which are indicated by a small white cross on a red background on the MQInput and Compute nodes. The ESQL_Simple message flow files and folders in the Resource Navigator view are also highlighted with crosses to show that the files contain errors.
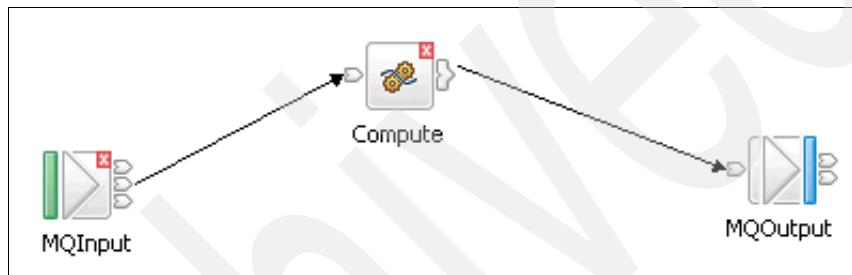
Figure 4-8   Validating the ESQL_Simple message flow

A brief description of each error is given in the Problems view below the Message Flow editor:

► The error in the MQInput node is because you have not entered the name of the WebSphere MQ input queue from which the MQInput node takes input messages.

► The error in the Compute node is because you have not created the ESQL module that defines how the Compute node should process input messages.

The following sections describe how to fix the errors by configuring the nodes.

4.2.2 Configuring the ESQL_Simple message flow

In the ESQL_Simple message flow, the MQInput node takes an input message from a WebSphere MQ local queue called ESQL_SIMPLE_IN. After the Compute node has processed the message, the MQOutput node puts the output message on a WebSphere MQ local queue called ESQL_SIMPLE_OUT.

To configure the message flow, create the two WebSphere MQ local queues, ESQI_SIMPLE_IN and ESQL_SIMPLE_OUT; set the properties on the message flow nodes; and create the ESQL module that processes the message in the Compute node.

Creating the WebSphere MQ local queues

WebSphere MQ objects are not administered from within the Message Brokers Toolkit, so use WebSphere MQ Explorer to create the WebSphere MQ local queues.

To create the ESQL_SIMPLE_IN and ESQL_SIMPLE_OUT queues:

1. Open WebSphere MQ Explorer: Click **Start** → **Programs** → **IBM WebSphere MQ** → **WebSphere MQ Explorer**.

2. In WebSphere MQ Explorer, in the Navigator view, expand **WBRK6_DEFAULT_QUEUE_MANAGER**, which is the name of the queue manager that hosts the broker.

3. Right-click the **Queues** folder under the WBRK6_DEFAULT_QUEUE_MANAGER queue manager, then click **New** → **Local Queue...** to open the New Local Queue wizard.

4. In the Name field of the wizard, type ESQL_SIMPLE_IN, then click **Next** (Figure 4-9 on page 60).
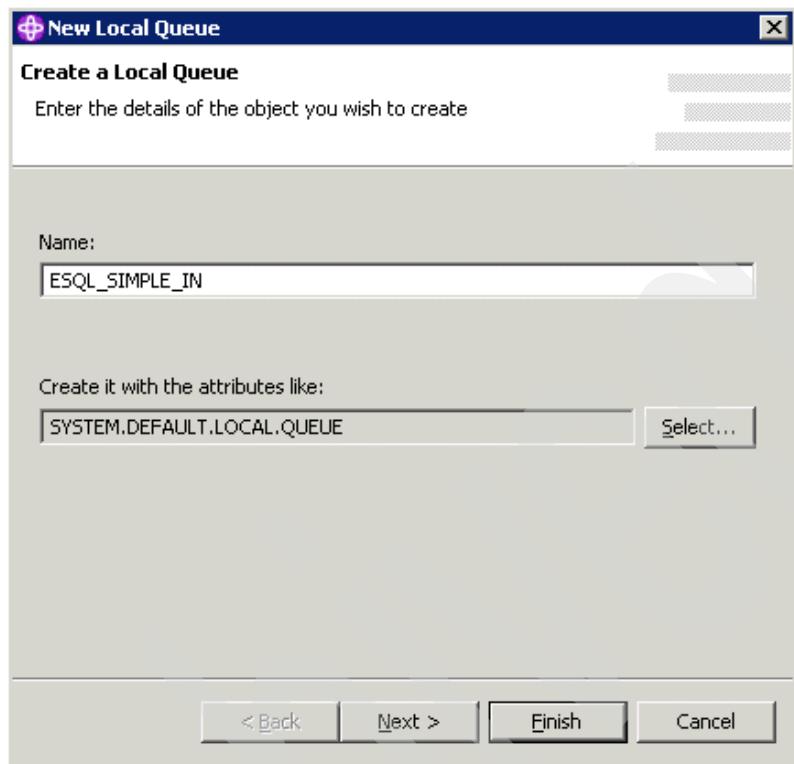
Figure 4-9   Creating a new WebSphere MQ queue

5. On the Storage page of the wizard, in the Backout requeue queue field, type
   `DLQ` (Figure 4-10 on page 61). The DLQ queue will be the backout requeue
   queue, or Dead-letter queue.

> **Tip:** The backout requeue queue, or Dead-letter queue, is where a message goes if the message flow cannot process it and rolls it back to the input queue. If you do not specify a backout requeue queue and there is a processing problem, the message is rolled back through the message flow and put back on the input queue, where it stays.
>
> The message flow gets messages from the input queue in the order in which the messages were put on the queue, so if the message is left on the input queue, it blocks any other input messages that you subsequently put on the queue.
>
> Another advantage to specifying a backout requeue queue is that any errors in the message processing are written to the Windows Event Viewer.
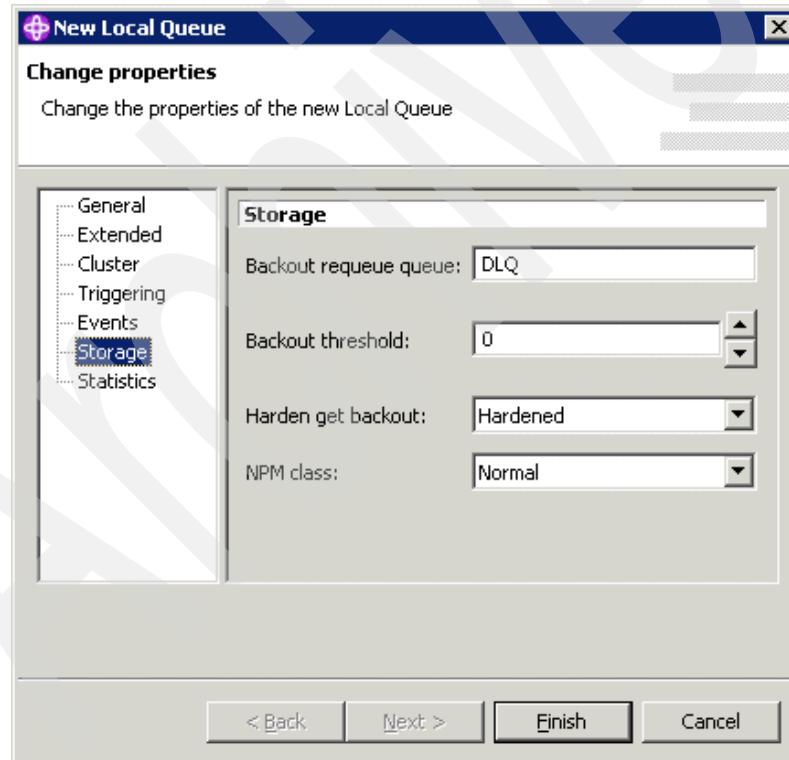
Figure 4-10   Specifying the backout requeue queue

6. Click **Finish**.

Click the **Queues** folder to display the queue, ESQL_SIMPLE_IN, in the Content view.

7. Create another local queue called ESQL_SIMPLE_OUT but do not specify a backout requeue queue. Only the input queue, ESQL_SIMPLE_IN, uses a backout requeue queue.

8. Create a third local queue called DLQ. This is the backout requeue queue, or Dead-letter queue.

   All three queues are displayed in the Content view (Figure 4-11).

You can use the DLQ queue as the backout requeue queue for any message flows that you create; you do not need to create a backout requeue queue for each message flow.
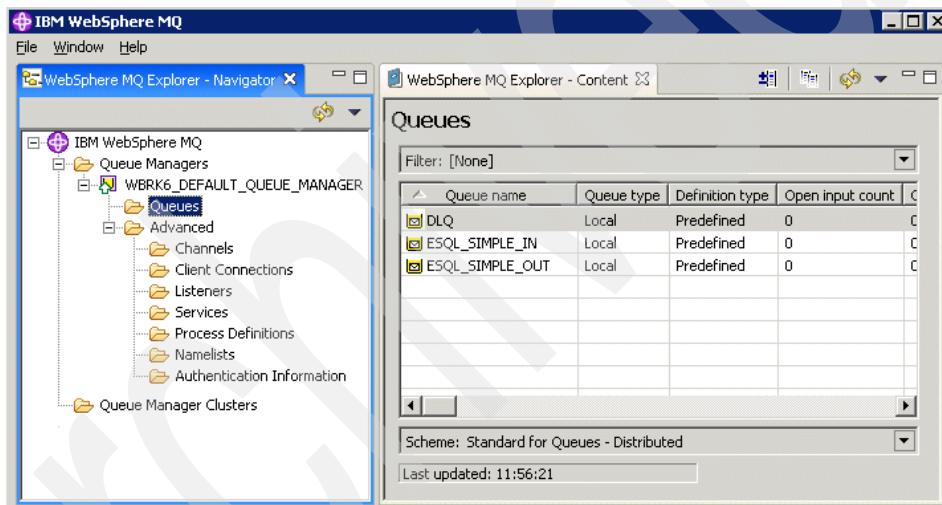
Figure 4-11   Displaying the queues in the WebSphere MQ Explorer Content view

The next section describes how to configure the nodes to connect to the correct queues.

Setting the properties of the nodes

The following instructions describe how to set the properties of the nodes in the ESQL_Simple message flow. When you have set the properties on the ESQL_SIMPLE_IN node and saved the message flow, the error label on the ESQL_SIMPLE_IN node disappears.

To set the properties of the nodes in the ESQL_Simple message flow:

1. Make sure that the **Selection** button at the top of the node palette is highlighted so that you can select nodes on the canvas.

2. Right-click the **ESQL_SIMPLE_IN** node, then click **Properties**. The node's Properties dialog opens.

3. On the Basic page of the Properties dialog, in the Queue Name field, type the name of the message flow's input queue: ESQL_SIMPLE_IN (Figure 4-12).
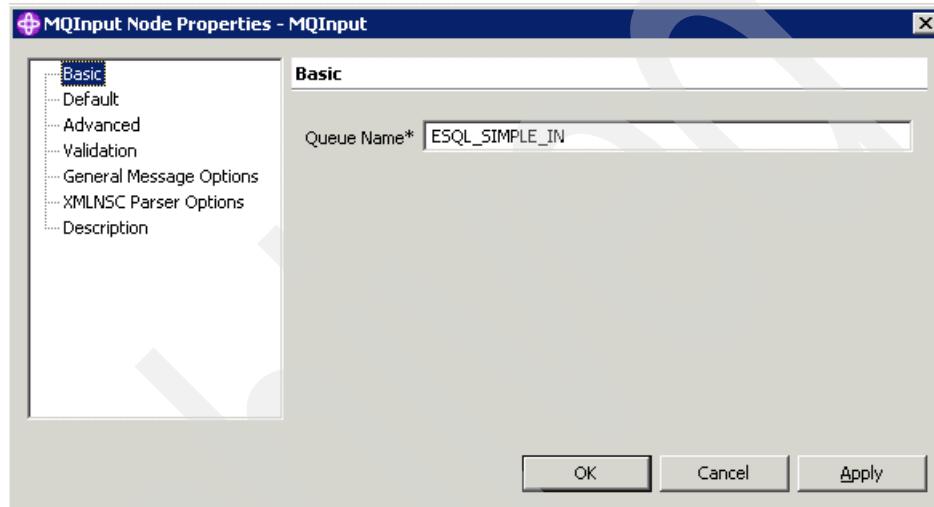
Figure 4-12   Setting the name of the input queue

4. On the Default page of the Properties dialog, in the Message Domain field, select **XML** from the list (Figure 4-13 on page 64).

   This tells the message flow to parse input messages as XML. If you do not set this property, the message flow cannot parse the input messages (see 4.1.1, "Messages in WebSphere Message Broker" on page 48).
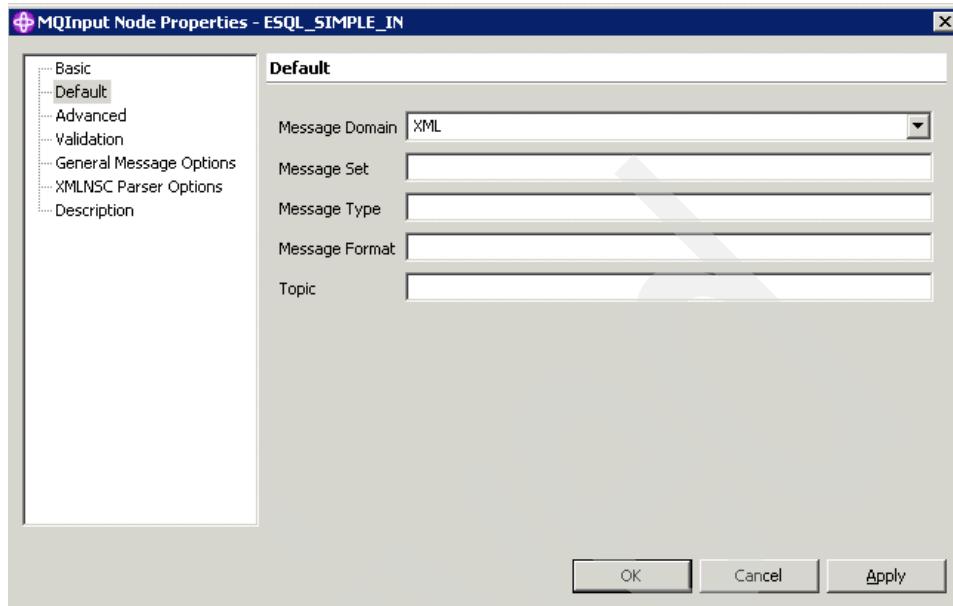
Figure 4-13   Specifying which parser to use to interpret input messages

Table 4-3 lists the properties to set for each of the nodes in the ESQL_Simple message flow.

5. Set the properties for the MQOutput node, ESQL_SIMPLE_OUT, as listed in Table 4-3.

    Do not enter a value in the Queue Manager Name field; if the field is empty, the message flow looks for the output queue on the same queue manager as the input queue.

Table 4-3   Node properties for the ESQL_Simple message flow

Node name	Page	Property	Value
ESQL_SIMPLE_IN	Basic	Queue name	ESQL_SIMPLE_IN
	Default	Message domain	XML
ESQL_SIMPLE_O UT	Basic	Queue name	ESQL_SIMPLE_O UT

6. Save ESQL_Simple.msgflow.

In the Message Flow editor, the error indicator on the MQInput node is no longer displayed.

> **Tip:** Double-click a node to open its Properties dialog quickly.

Do not edit any of the properties in the Compute node, but look in the Properties dialog to see what has caused the error that is still displayed in the Message Flow:

1. Open the Properties dialog for the Compute node.

2. On the Basic page of the Properties dialog, notice that the ESQL Module field contains `ESQL_Simple_Compute` (Figure 4-14). This value is entered automatically when the node is created because the Compute node must contain some ESQL. ESQL is held in a separate file called, in this case, ESQL_Simple.esql. However, an error is displayed in the Problems view because the module and the file ESQL_Simple.esql do not yet exist.
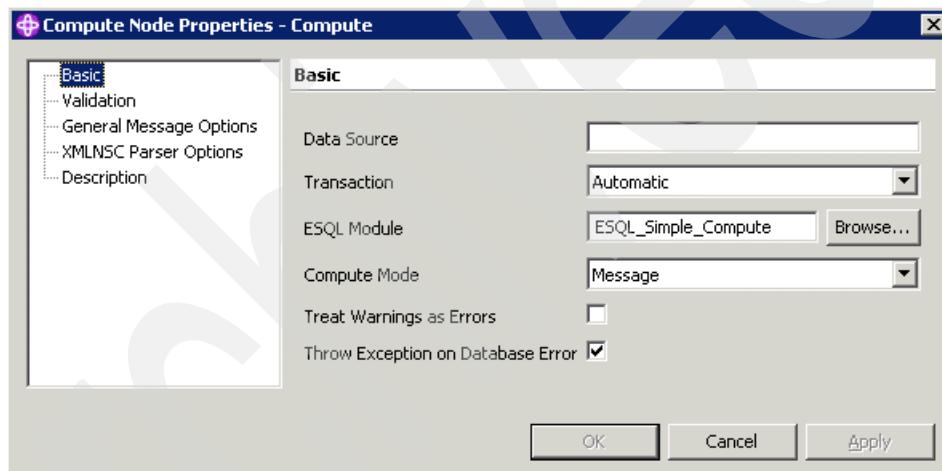
Figure 4-14   The Compute node properties

3. Click **Cancel** to close the Properties dialog box without saving any changes.

The next section describes how to create ESQL_Simple.esql and how to create some simple ESQL for the Compute node.

4.2.3 Writing ESQL for the Compute node

All of the ESQL that belongs to a message flow is stored, by default, in a single file. In this case, all of the ESQL for the ESQL_Simple message flow is stored in a file called ESQL_Simple.esql.

Creating the ESQL file

To create ESQL_Simple.esql:

1. In the Message Flow editor, right-click the **Compute** node, then click **Open ESQL**.

   ESQL_Simple.esql does not already exist, so the Message Brokers Toolkit creates the file in the ESQL_Simple Message Flow project. When the ESQL_Simple.esql file is created, it automatically opens in the ESQL editor and already contains the minimum ESQL that is needed for the Compute node to successfully validate.

2. Save ESQL_Simple.esql, then click the **ESQL_Simple.msgflow** tab to return to the Message Flow editor. The error on the Compute node is no longer displayed and no items relating to this task appear in the Problems view.

3. Click the **ESQL_Simple.esql** tab to display ESQL_Simple.esql in the ESQL editor again.

Writing the ESQL_Simple_Compute ESQL module

In the ESQL_Simple.esql file, there is a single module of ESQL called ESQL_Simple_Compute. This is the ESQL module that is referenced from the Compute node Properties dialog.

The ESQL that is generated automatically does not produce any output so you must edit the ESQL in ESQL_Simple.esql file:

1. Uncomment the fifth line (--CALL CopyEntireMessage();) of the ESQL_Simple_Compute module so that the Compute module can parse it. To uncomment the line, delete -- from the start of the line, as shown in Example 4-1.

Example 4-1   ESQL for the ESQL_Simple_Compute ESQL module

```
CREATE COMPUTE MODULE ESQL_Simple_Compute
   CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN
      -- CALL CopyMessageHeaders();
       CALL CopyEntireMessage();
      RETURN TRUE;
   END;

   CREATE PROCEDURE CopyMessageHeaders() BEGIN
      DECLARE I INTEGER;
      DECLARE J INTEGER;
      SET I = 1;
      SET J = CARDINALITY(InputRoot.*[]);
      WHILE I < J DO
         SET OutputRoot.*[I] = InputRoot.*[I];
```

```
        SET I = I + 1;
      END WHILE;
    END;

    CREATE PROCEDURE CopyEntireMessage() BEGIN
        SET OutputRoot = InputRoot;
    END;
END MODULE;
```

The Compute node can now parse the line that instructs the Compute node to copy the header and content of the input message (InputRoot) to the output message (OutputRoot). The message that the MQOutput node, ESQL_SIMPLE_OUT, puts on the ESQL_SIMPLE_OUT queue has the same content as the message that the MQInput node, ESQL_SIMPLE_IN, got from the ESQL_SIMPLE_IN queue.

2. Save ESQL_Simple.esql and ESQL_Simple.msgflow.

Attention: Ignore any warning messages about unresolvable database table references that are also displayed in the Problems view when you save the message flow. These messages are displayed because the Message Brokers Toolkit does not have access to the database fields that the ESQL in the message flow refers to. It is possible to connect to the database from the Message Brokers Toolkit, but it is unnecessary for the exercises in this chapter ("Creating the Create_Customer_Account message flow" on page 181 describes how to connect to a database from the Message Brokers Toolkit).

The next section describes how to deploy the ESQL_Simple message flow so that you can test it.

4.2.4 Deploying and testing the ESQL_Simple message flow

Before you can test the ESQL_Simple message flow, you must deploy it to the broker. Then, when you put the test input message on the ESQL_SIMPLE_IN queue, the broker processes the message using the deployed message flow.

Deploying the ESQL_Simple message flow
To deploy a message flow to the broker, package it in a message broker archive (bar) file. 7.4.2, "Deploying resources to a remote broker" on page 226 provides more information about deployment. This chapter just explains what you need to do to deploy the message flow applications that you create so that you can test them.

The following instructions about how to deploy the ESQL_Simple message flow assume that you have run the Default Configuration wizard so that the Default Configuration is available on the same computer as the Message Brokers Toolkit.

To deploy the ESQL_Simple message flow to the broker:

1. Switch to the Broker Administration perspective: From the Broker Application Development perspective, click **Window** → **Open Perspective** → **Broker Administration**.

2. Create a new bar file: Click **File** → **New** → **Message Broker Archive**. The New Message Broker Archive wizard opens.

3. In the wizard, click the **LocalProject** server project.

   This is the server project that was created by the Default Configuration wizard to contain the details of the connection between the Message Brokers Toolkit and the Configuration Manager. The instructions in this chapter assume that you are using the LocalProject server project (Figure 4-15 on page 69). If you have created your own domain, substitute the name of your server project in place of LocalProject.

4. In the File Name field, type the name of the bar file: `ESQL_Simple` (Figure 4-15 on page 69).

5. Click **Finish**.

   The ESQL_Simple.bar file is displayed in the Broker Administration Navigator view under Broker Archives → LocalProject. The ESQL_Simple.bar file is automatically opened in the Broker Archive editor.
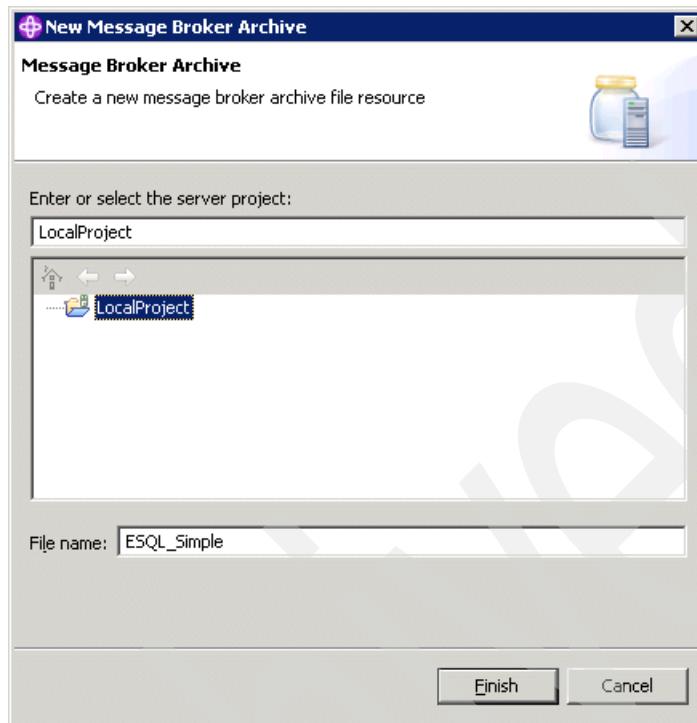
Figure 4-15   Creating a new message broker archive (bar) file

In the Broker Archive editor, there are two buttons. The first button adds files to the archive and the second button, with the red cross, deletes files from the archive (Figure 4-16).

Figure 4-16   The Add and Remove buttons in the Broker Archive editor

6. Click the **Add** button. The Add to Broker Archive dialog opens.

7. In the dialog, click **ESQL_Simple Message Flow Project** to highlight it, then select the **ESQL_Simple.msgflow** check box (Figure 4-17 on page 70).

8. Click **OK** to add the selected file to the bar file.
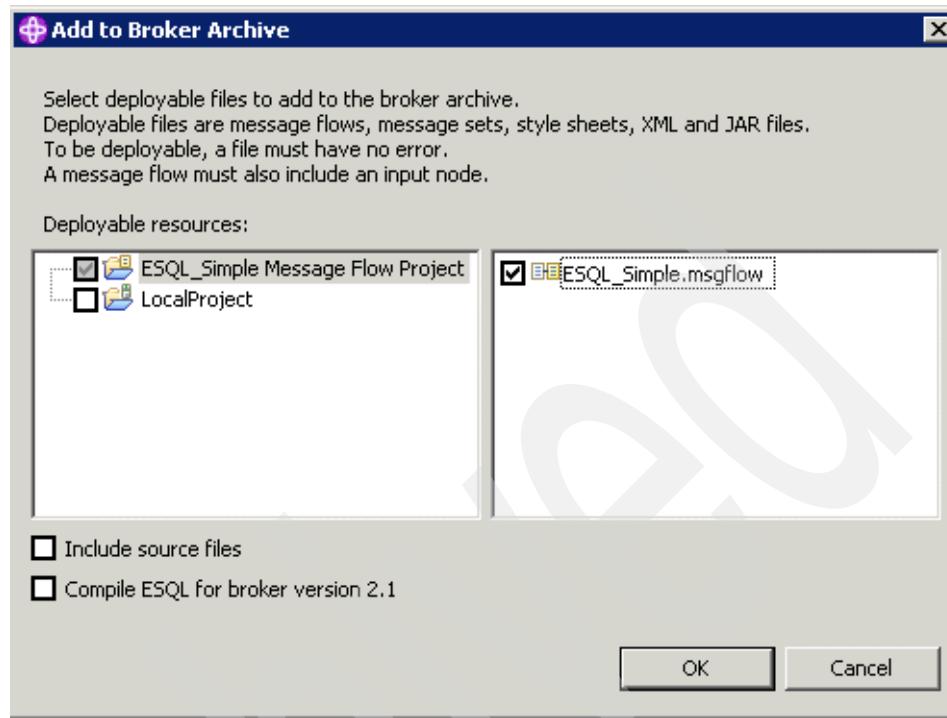
Figure 4-17   Adding the ESQL_Simple message flow to the ESQL_Simple.bar file

The compiled message flow, ESQL_Simple.cmf, is displayed in the ESQL_Simple.bar file in the Broker Archive editor (Figure 4-18 on page 71).

Figure 4-18   The compiled message flow in the bar file

9. Save the bar file: Click **File** → **Save** or press Ctrl+S.

10. Ensure that the Message Brokers Toolkit is connected to the Configuration Manager. If it is not connected, in the Domains view, right-click the connection, then click **Connect**. Wait while the connection is made. If you have problems connecting, see 8.4.3, "Problems connecting to the Configuration Manager" on page 296, for more information.

11. Create a new execution group on the WBRK6_DEFAULT_BROKER broker: Click **File** → **New** → **Execution Group**. The New Execution Group dialog opens.

12. In the dialog, expand the connection name, expand **Broker Topology**, then click **WBRK6_DEFAULT_BROKER** to highlight it.

13. In the Execution Group Name field, type ESQL_Simple (Figure 4-19 on page 72), then click **Finish**.
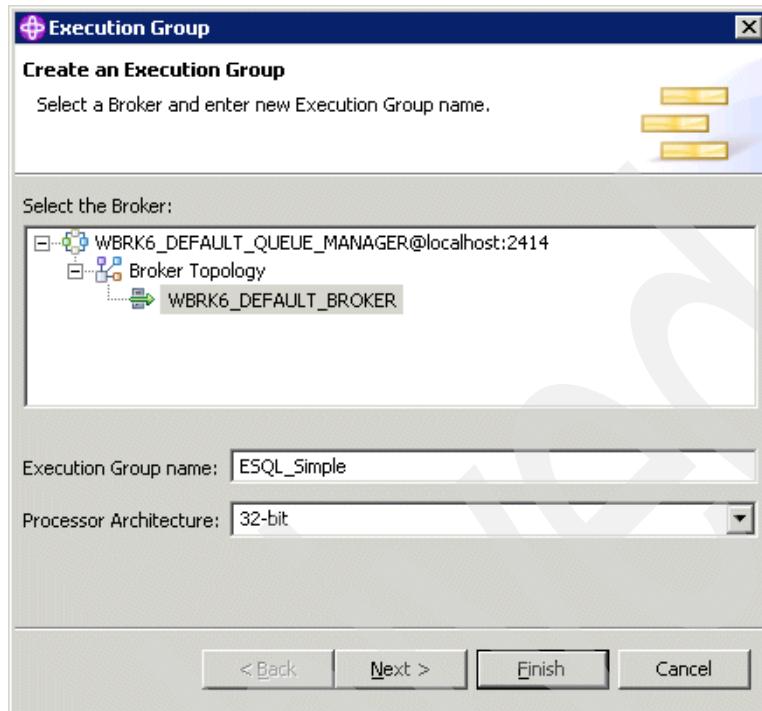
Figure 4-19   Creating a new execution group

The ESQL_Simple execution group is displayed in the Domains view under the WBRK6_DEFAULT_BROKER broker (Figure 4-20).
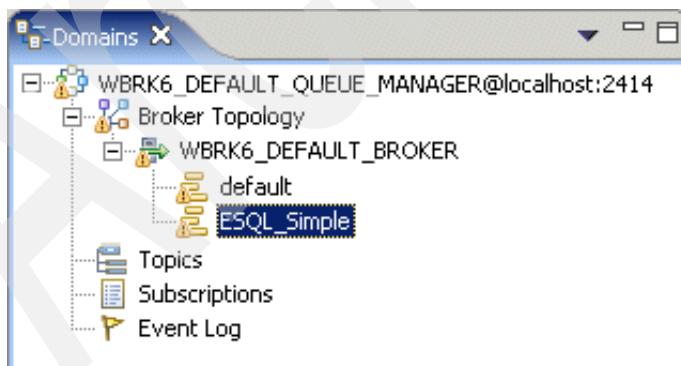
Figure 4-20   The ESQL_Simple execution group in the Domains view

14. In the Broker Administration Navigator view, right-click **ESQL_Simple.bar**, then click **Deploy File...**. The Deploy a BAR File dialog opens.

15. Click the name of the broker, **WBRK6_DEFAULT_BROKER**, to which you want to deploy the bar file, then click **OK** (Figure 4-21).
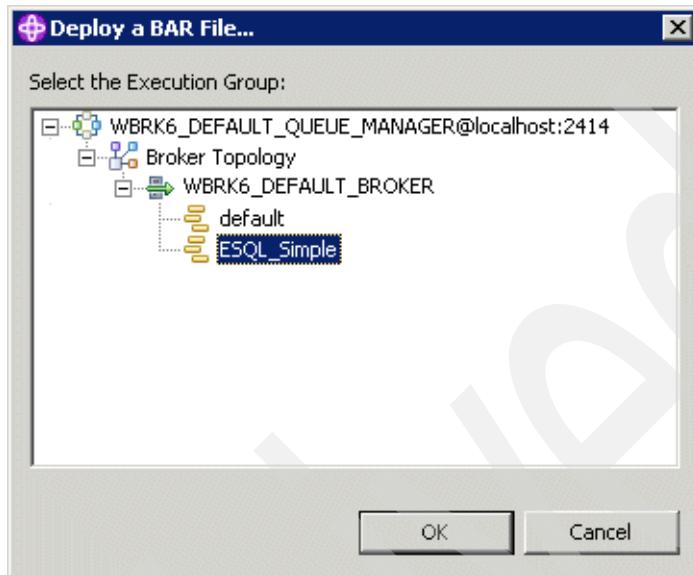
Figure 4-21   Deploying ESQL_Simple bar file to ESQL_Simple execution group

Wait until a message is displayed saying that the bar file was successfully deployed to the broker. This message does not mean that the deployment was completed successfully. When the bar file has been successfully deployed, the ESQL_Simple message flow is displayed in the Domains view under the ESQL_Simple execution group (Figure 4-22 on page 74).

If the bar file does not appear to be deployed successfully, look in the Windows Event Log for possible errors and see 8.4.4, "Problems with deployment" on page 299, for more information about problem determination.
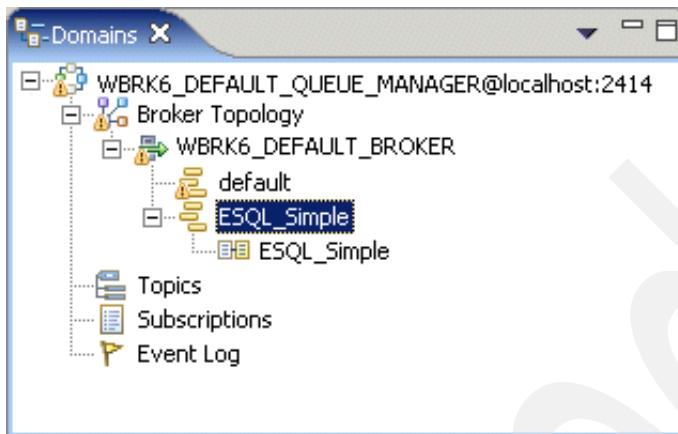
Figure 4-22   The ESQL_Simple message flow deployed

Testing the ESQL_Simple message flow

To test the ESQL_Simple message flow, use the Enqueue editor in the Message Brokers Toolkit to put an XML input message on the ESQL_SIMPLE_IN queue. The ESQL_Simple message flow gets the input message from the ESQL_SIMPLE_IN queue, processes the message, and then puts an output message on the ESQL_SIMPLE_OUT queue.

The Enqueue editor enables you to easily create a message without having to understand how to create message headers; the Enqueue editor automatically adds an MQMD header to the message. An MQMD header is required for messages used with WebSphere Message Broker. All you need to enter in the Enqueue editor is the details of the queue and queue manager on which to put the message, and the message data itself.

To create an enqueue file for the message content:

1. Click **File** → **New** → **Enqueue Message File**. The New Enqueue Message File wizard opens.

2. In the wizard, click **ESQL_Simple Message Flow Project** to highlight it, then in the File name field, type ESQL_Simple (Figure 4-23 on page 75). Click **Finish**.

   The new enqueue file, ESQL_Simple.enqueue, is added to ESQL_Simple Message Flow Project. Switch to the Broker Application Development perspective to see this; enqueue files are not displayed in the Broker Application Development perspective.

   The ESQL_Simple.enqueue file is automatically opened in the Enqueue editor.
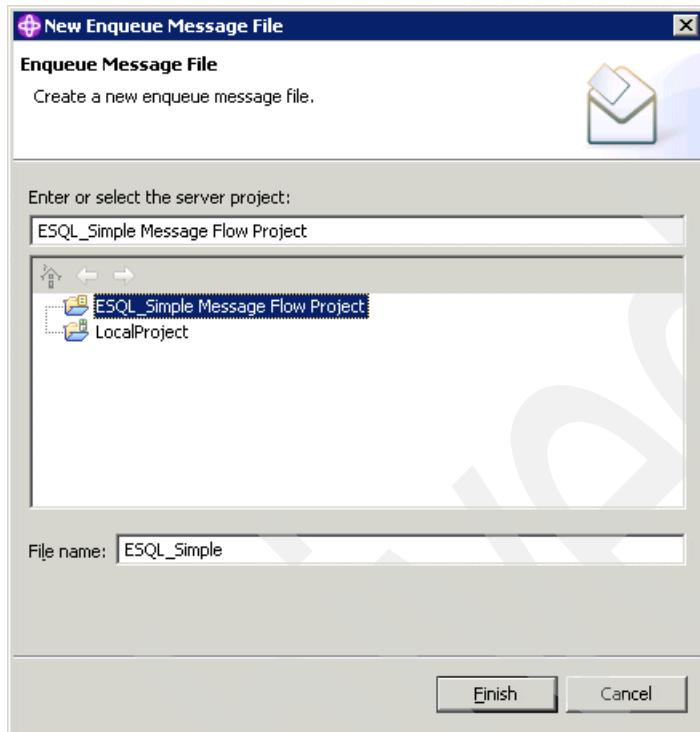
Figure 4-23   Creating a new enqueue message file

3. In the Queue manager name field, type the name of the queue manager, for example, WBRK6_DEFAULT_QUEUE_MANAGER. The Enqueue utility is case-sensitive.

4. In the Port field, type the port number that the queue manager listens on, for example, 2414.

5. In the Queue name field, type ESQL_SIMPLE_IN.

6. In the Message data field, type the XML message content shown in Example 4-2. You can also copy the message content from the Web material available to download (see Appendix B, "Code" on page 319).

Example 4-2   Input message content to test the ESQL_Simple message flow

```
<Message>
    <Body>
        Hello, world!
    </Body>
</Message>
```

Figure 4-24 on page 76 shows the completed ESQL_Simple.enqueue file.

7. Click the **General** tab to return to the main page.

8. Save the ESQL_Simple.enqueue file.

9. Click the **Write To queue** button to put the XML input message on the ESQL_SIMPLE_IN queue. A message is displayed to confirm that the message was successfully put to the queue.
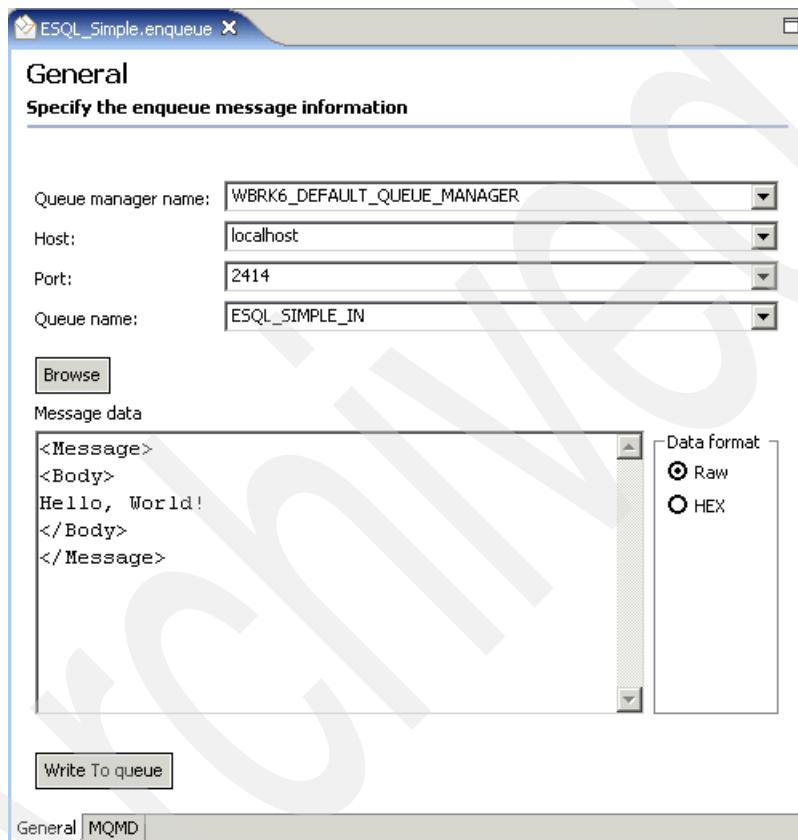
Figure 4-24   The ESQL_Simple.enqueue file

The ESQL_Simple message flow gets the message from the queue, processes the message, then puts the message on the ESQL_SIMPLE_OUT queue.

To get the output message from the ESQL_SIMPLE_OUT queue:

1. Click the **Dequeue** button on the Message Brokers Toolkit toolbar (see Figure 4-25 on page 77) to open the Dequeue Message wizard.

Figure 4-25   The icon on the Dequeue button on the toolbar

2. In the Queue Manager Name field, type `WBRK6_DEFAULT_QUEUE_MANAGER`, and in the Queue Name field type `ESQL_SIMPLE_OUT` (Figure 4-26). The Dequeue Message wizard is case-sensitive.

3. Click **Read From Queue**.

If the message was processed correctly, the content of the output message is displayed in the wizard, as shown in Figure 4-26.
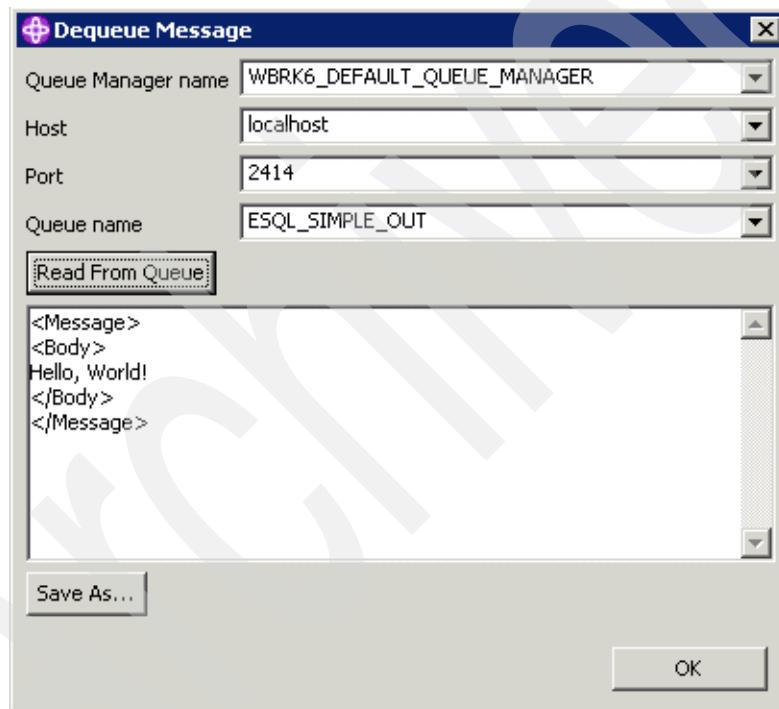
Figure 4-26   Getting the output message from ESQL_SIMPLE_OUT
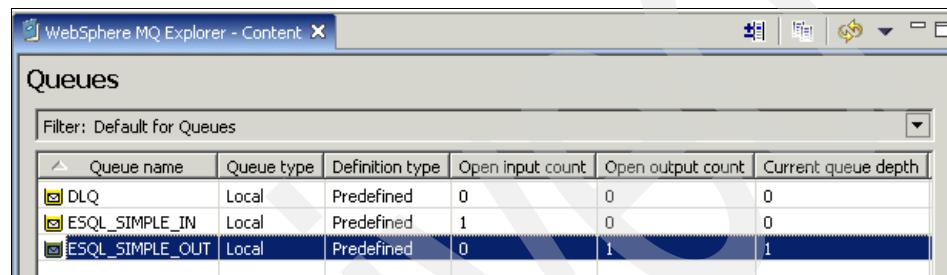
If you receive the following Dequeue error message, the ESQL_Simple message flow has not put an output message on to the ESQL_SIMPLE_OUT queue:

```
BIP0917E No messages found to dequeue
```

4.2.5 Diagnosing problems with the ESQL_Simple message flow

If the Dequeue wizard cannot get a message from the ESQL_SIMPLE_OUT queue, check first that you have entered the correct details in the enqueue file and in the Dequeue wizard. If all the details are correct, perform the following checks to diagnose the problem:

1. Open WebSphere MQ Explorer and check the Current queue depth column for the ESQL_SIMPLE_IN, ESQL_SIMPLE_OUT, and DLQ queues. The Current queue depth column displays the number of messages currently on each queue (Figure 4-27).

Figure 4-27   Checking the queues for messages

2. If the message is still on the ESQL_SIMPLE_IN queue, the message flow failed to get the message from the queue. Reasons for this might be:

   – The wrong queue name is specified in the MQInput node properties. If so, fix the problem, add the message flow to the bar file again (the bar file does not dynamically update itself), then re-deploy the bar file to the broker.

   – The message flow is not running. If so, in the Domains view of the Broker Administration perspective, right-click the message flow then click **Start**.

   – The name of the backout requeue queue, DLQ, was incorrectly entered in the properties of the ESQL_SIMPLE_IN queue and a previous input message was rolled back by the message flow to the ESQL_SIMPLE_IN queue. Because the message flow cannot process the previous message, it is now blocking subsequent input messages from being processed by the message flow. If so, edit the queue's properties so that the Backout requeue queue is DLQ, as described earlier in this chapter.

   Use the Dequeue Message wizard to get the message from the ESQL_SIMPLE_IN queue so that the there are no messages on the ESQL_SIMPLE_IN queue, otherwise subsequent input messages will be blocked by the existing messages on the queue.

3. If the message is on the DLQ queue, the message flow was not able to process the message and so rolled the message back through the message flow to the ESQL_SIMPLE_IN queue. The message was then moved to the DLQ queue. Reasons for this happening might be:

    – The XML in the input message is badly formed; for example, one of the tags is missing or is misspelled.

    – There is a problem with the ESQL in the Compute node so it cannot process the message.

4. In WebSphere MQ Explorer, verify that the following objects are running:

    – WBRK6_DEFAULT_QUEUE_MANAGER queue manager. To start the queue manager, right-click the queue manager, then click **Start**.

    – WBRK6_DEFAULT_QUEUE_MANAGER listener. To start the listener, right-click the listener, then click **Start**.

    – The WBRK6_DEFAULT_QUEUE_MANAGER queue manager's command server. To start the command server, right-click the queue manager, then click **Start Command Server**.

5. In the ESQL_Simple message flow, make sure that the queue names in the MQInput and MQOutput nodes are spelled correctly and are in the correct case.

If none of these suggestions solve the problem, see **Chapter 8, "Troubleshooting and problem determination" on page 241**, for more information about things to check.

4.3 Developing the Bookstore scenario using ESQL

In 4.2, "Developing the Simple message flow application" on page 52, you created the Simple scenario message flow application using ESQL to define the logic of the message flow.

In this section, we create a more complex message flow application that is based around the scenario of an online bookstore. The Bookstore scenario message flows process messages with different structures, and interact with databases to update database tables.

The Bookstore scenario includes two message flows:

► The ESQL_Create_Customer_Account message flow

  This message flow uses ESQL in a Database node to create accounts in a DB2 database table for new customers who have registered their details with the bookstore, for example, their contact details and delivery address.

► The ESQL_Book_Order message flow

This message flow uses ESQL in a Compute node to process an order that has been submitted by an online customer and create a response message to confirm the order with a unique order number.

4.3.1 Creating the Bookstore scenario database

The Bookstore scenario database, BSTOREDB, is a DB2 database that contains some tables of sample data. The ESQL_Create_Customer_Account message flow inserts data (a customer's registration details) into a table in the BSTOREDB database.

In the Web material described in Appendix B, "Code" on page 319, there is an SQL script, BookStoreDB.sql, that you can run to create the BSTOREDB database.

To create the BSTOREDB database, tables, and sample data:

1. Start a DB2 Command Window: **Start** → **Programs** → **IBM DB2** → **Command Line Tools** → **Command Window**.

2. In the Command Window, make sure that DB2 is running by entering the following command:

   ```
   db2start
   ```

3. Change to the directory that contains the BookStoreDB.sql script, for example, if the BookStoreDB.sql file is in C:\Temp:

   ```
   cd C:\Temp
   ```

4. Run the script:

   ```
   db2 -vf BookStoreDB.sql
   ```

The script drops any tables that already exist of the same name, then creates and populates new ones.

4.3.2 Creating the ESQL_Create_Customer_Account message flow

The input message to the ESQL_Create_Customer_Account message flow contains the details of a customer who has registered with the online bookstore Web site. The customer has provided information such as their name, contact details, a delivery address, and payment details. Example 4-3 shows the input message for the ESQL_Create_Customer_Account message flow.

Example 4-3   The message for the ESQL_Create_Customer_Account message flow

```
<Create_Customer_Account_MSG>
   <Personal_Details>
```

```
        <First_Name>Peter</First_Name>
        <Last_Name>Smith</Last_Name>
        <User_ID>PSmith</User_ID>
        <Password>p45sw0rd</Password>
    </Personal_Details>
    <Email_Address>Peter.Smith@nowhere.com</Email_Address>
    <Daytime_Telephone>1234567890</Daytime_Telephone>
    <Evening_Telephone>1234567890</Evening_Telephone>
    <Shipping_Address>
        <Address_1>19 Green Street</Address_1>
        <Address_2>Littleton</Address_2>
        <Town>Southington</Town>
        <Postcode>SU29 8YT</Postcode>
    </Shipping_Address>
    <Billing_Address>
        <Address_1>19 Green Street</Address_1>
        <Address_2>Littleton</Address_2>
        <Town>Southington</Town>
        <Postcode>SU29 8YT</Postcode>
    </Billing_Address>
    <Payment_Details>
        <Card>VISA</Card>
        <Card_Number>1234567890</Card_Number>
        <Expiry_Date>31.12.2009</Expiry_Date>
        <Issue_Date>31.12.2004</Issue_Date>
        <Issue_Number>02</Issue_Number>
        <Security_Code>333</Security_Code>
    </Payment_Details>
</Create_Customer_Account_MSG>
```

All of the ESQL that you need for this message flow is available in the Web material described in Appendix B, "Code" on page 319.

Figure 4-28 shows the finished ESQL_Create_Customer_Account message flow.
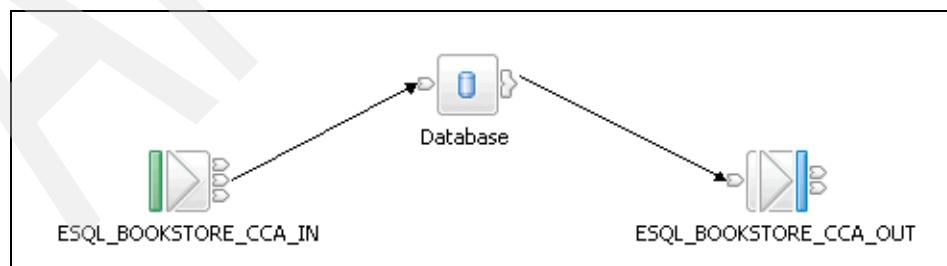
Figure 4-28   The ESQL_Create_Customer_Account message flow

To create the ESQL_Create_Customer_Account message flow:

1. In the Broker Application Development perspective, create a Message Flow project called `ESQL_Bookstore Message Flow Project`.

2. Create a message flow called `ESQL_Create_Customer_Account` in the ESQL_Bookstore Message Flow Project. The ESQL_Create_Customer_Account.msgflow file opens in the Message Flow editor.

3. In the Message Flow editor, add the nodes listed in Table 4-4 to the canvas, then connect the nodes together, as shown in Table 4-5, to build the ESQL_Create_Customer_Account message flow (Figure 4-28 on page 81).

Table 4-4   The ESQL_Create_Customer_Account message flow nodes

Node type	Node name
MQInput	ESQL_BOOKSTORE_CCA_IN
Database	Database
MQOutput	ESQL_BOOKSTORE_CCA_OUT

Table 4-5   Node connections in the ESQL_Create_Customer_Account message flow

Node name	Terminal	Connect to this node
ESQL_BOOKSTORE_CCA_IN	Out	Database
Database	Out	ESQL_BOOKSTORE_CCA_OUT

4. Set the properties of the nodes, as shown in Table 4-6.

Table 4-6   Node properties for the ESQL_Create_Customer_Account message flow

Node name	Page	Property	Value
ESQL_BOOKSTORE_CCA_IN	Basic	Queue Name	ESQL_BOOKSTORE_CCA_IN
	Default	Message Domain	XML
Database	Basic	Data Source	BSTOREDB
ESQL_BOOKSTORE_CCA_OUT	Basic	Queue Name	ESQL_BOOKSTORE_CCA_OUT

5. In the Message Flow editor, right-click the **Database** node, then click **Open ESQL** to create the ESQL module that is referenced in the Database node Properties dialog. The ESQL_Create_Customer_Account.esql file opens in the ESQL editor.

6. In the ESQL editor, edit the ESQL_Create_Customer_Account.esql file. The ESQL selects content from the input message and inserts it into the CUSTACCTB table in the BSTOREDB database:

   a. Use SQL (INSERT INTO Database) to update the database table CUSTACCTB with information from the message. Each field in the database table (for example, LAST_NAME, FIRST_NAME, USERID) is listed in the order in which it occurs in the database.

Example 4-4   Using SQL to update the CUSTACCTB database table

```
    INSERT INTO Database.CUSTACCTB(LAST_NAME, FIRST_NAME, USERID, PASSWORD,
EMAIL, DAY_PHONE, EVE_PHONE, SHIP_ADDRESS1, SHIP_ADDRESS2, SHIP_TOWN,
SHIP_POSTCODE, BILL_ADDRESS1, BILL_ADDRESS2, BILL_TOWN, BILL_POSTCODE,
CARDTYPE, CARDNUM, EXP_DATE, ISS_DATE, ISS_NUM, SECCODE)
```

   b. Specify each value from the message that is to be added to the CUSTACCTB table. One value is inserted into each field of the database table. ESQL specifies fields by navigating through the hierarchical structure of the message from the root of the message body, which is known as *Body*. Compare the values in the ESQL in Example 4-5 with the XML fields in Example 4-3 on page 80 to see how the ESQL navigates the message fields.

   The order in which the values are listed in the ESQL determines the field into which each value is inserted. So the Body.Create_Customer_Account_MSG.Personal_Details.Last_Name value is inserted into the LAST_NAME field of the CUSTACCTB table and the Body.Create_Customer_Account_MSG.Payment_Details.Security_Code value is inserted into the SECCODE field of the CUSTACCTB table.

Example 4-5   Specifying the values to insert into the CUSTACCTB table

```
VALUES(Body.Create_Customer_Account_MSG.Personal_Details.Last_Name,
    Body.Create_Customer_Account_MSG.Personal_Details.First_Name,
    Body.Create_Customer_Account_MSG.Personal_Details.User_ID,
    Body.Create_Customer_Account_MSG.Personal_Details.Password,
    Body.Create_Customer_Account_MSG.Email_Address,
    Body.Create_Customer_Account_MSG.Daytime_Telephone,
    Body.Create_Customer_Account_MSG.Evening_Telephone,
    Body.Create_Customer_Account_MSG.Shipping_Address.Address_1,
    Body.Create_Customer_Account_MSG.Shipping_Address.Address_2,
    Body.Create_Customer_Account_MSG.Shipping_Address.Town,
```

```
            Body.Create_Customer_Account_MSG.Shipping_Address.Postcode,
            Body.Create_Customer_Account_MSG.Billing_Address.Address_1,
            Body.Create_Customer_Account_MSG.Billing_Address.Address_2,
            Body.Create_Customer_Account_MSG.Billing_Address.Town,
            Body.Create_Customer_Account_MSG.Billing_Address.Postcode,
            Body.Create_Customer_Account_MSG.Payment_Details.Card,
            Body.Create_Customer_Account_MSG.Payment_Details.Card_Number,
            Body.Create_Customer_Account_MSG.Payment_Details.Expiry_Date,
            Body.Create_Customer_Account_MSG.Payment_Details.Issue_Date,
            Body.Create_Customer_Account_MSG.Payment_Details.Issue_Number,
            Body.Create_Customer_Account_MSG.Payment_Details.Security_Code);
```

Example 4-6 shows the final ESQL code to use in the
ESQL_Create_Customer_Account_Database module.

Example 4-6   ESQL for the ESQL_Create_Customer_Account_Database module

```
CREATE DATABASE MODULE ESQL_Create_Customer_Account_Database
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN

    INSERT INTO Database.CUSTACCTB(LAST_NAME, FIRST_NAME, USERID, PASSWORD,
EMAIL, DAY_PHONE, EVE_PHONE,  SHIP_ADDRESS1, SHIP_ADDRESS2, SHIP_TOWN,
SHIP_POSTCODE, BILL_ADDRESS1, BILL_ADDRESS2, BILL_TOWN, BILL_POSTCODE,
CARDTYPE, CARDNUM, EXP_DATE, ISS_DATE, ISS_NUM, SECCODE)
    VALUES(Body.Create_Customer_Account_MSG.Personal_Details.Last_Name,
    Body.Create_Customer_Account_MSG.Personal_Details.First_Name,
    Body.Create_Customer_Account_MSG.Personal_Details.User_ID,
    Body.Create_Customer_Account_MSG.Personal_Details.Password,
    Body.Create_Customer_Account_MSG.Email_Address,
    Body.Create_Customer_Account_MSG.Daytime_Telephone,
    Body.Create_Customer_Account_MSG.Evening_Telephone,
    Body.Create_Customer_Account_MSG.Shipping_Address.Address_1,
    Body.Create_Customer_Account_MSG.Shipping_Address.Address_2,
    Body.Create_Customer_Account_MSG.Shipping_Address.Town,
    Body.Create_Customer_Account_MSG.Shipping_Address.Postcode,
    Body.Create_Customer_Account_MSG.Billing_Address.Address_1,
    Body.Create_Customer_Account_MSG.Billing_Address.Address_2,
    Body.Create_Customer_Account_MSG.Billing_Address.Town,
    Body.Create_Customer_Account_MSG.Billing_Address.Postcode,
    Body.Create_Customer_Account_MSG.Payment_Details.Card,
    Body.Create_Customer_Account_MSG.Payment_Details.Card_Number,
    Body.Create_Customer_Account_MSG.Payment_Details.Expiry_Date,
    Body.Create_Customer_Account_MSG.Payment_Details.Issue_Date,
    Body.Create_Customer_Account_MSG.Payment_Details.Issue_Number,
    Body.Create_Customer_Account_MSG.Payment_Details.Security_Code);

        RETURN TRUE;
    END;
```

```
END MODULE;
```

7. Save the ESQL_Create_Customer_Account.esql and ESQL_Create_Customer_Account.msgflow files.

8. Create the following WebSphere MQ queues in WebSphere MQ Explorer on the WBRK6_DEFAULT_QUEUE_MANAGER queue manager:

   – ESQL_BOOKSTORE_CCA_IN
   – ESQL_BOOKSTORE_CCA_OUT

   Remember to enter the value of the Backout requeue queue property on the ESQL_BOOKSTORE_CCA_IN queue as DLQ.

You have created the ESQL_Create_Customer_Account message flow, which is one of the Bookstore scenario message flows. When you have created the other message flow, ESQL_Book_Order, deploy and test them together.

4.3.3 Creating the ESQL_Book_Order message flow

The input message to the ESQL_Book_Order message flow contains an order from a customer on the online bookstore Web site. The order contains the customers' identification and details of the books that they have ordered. When the message flow processes the message, it creates a confirmation message that contains details of the order, including a unique order number and the total price of all the books in the order. Example 4-7 shows the input message for the ESQL_Book_Order message flow.

Example 4-7   The input message for the ESQL_Book_Order message flow

```
<Create_Book_Order_MSG>
   <Customer_ID>0123456789</Customer_ID>
   <Order_Date>2005-09-27 12:55:12</Order_Date>
   <First_Class>Yes</First_Class>
   <Book_Details>
      <ISBN>0123456789</ISBN>
      <Book_Price>15.99</Book_Price>
      <ISBN>1425112342</ISBN>
      <Book_Price>7.99</Book_Price>
      <ISBN>9736316345</ISBN>
      <Book_Price>25.99</Book_Price>
   </Book_Details>
</Create_Book_Order_MSG>
```

Example 4-8 on page 86 shows the output message that the message flow generates based on the input message shown in Example 4-7.

Example 4-8   The Book_Order_Response_MSG message for the ESQL_Book_Order message flow

```
<Book_Order_Response_MSG>
   <Customer_ID>0123456789</Customer_ID>
   <Order_Number>012345678920050927125512</Order_Number>
   <Order_Date>2005-09-27 12:55:12</Order_Date>
   <First_Class>Yes</First_Class>
   <Book_Details>
      <ISBN>0123456789</ISBN>
      <Book_Price>15.99</Book_Price>
   </Book_Details>
   <Book_Details>
      <ISBN>1425112342</ISBN>
      <Book_Price>7.99</Book_Price>
   </Book_Details>
   <Book_Details>
      <ISBN>9736316345</ISBN>
      <Book_Price>25.99</Book_Price>
   </Book_Details>
   <Delivery_Price>18.00</Delivery_Price>
   <Total_Price>49.97</Total_Price>
   <Order_Status>Order Received</Order_Status>
</Book_Order_Response_MSG>
```

All of the ESQL that you need for this message flow is available in the Web material described in Appendix B, "Code" on page 319.

Figure 4-29 shows the finished ESQL_Book_Order message flow.
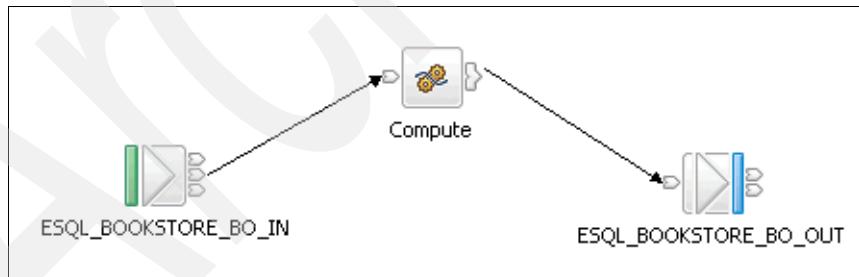

Figure 4-29   The ESQL_Book_Order message flow

To create the ESQL_Book_Order message flow:

1. In the Broker Application Development perspective, create a message flow called ESQL_Book_Order in the ESQL_Bookstore Message Flow Project. The ESQL_Book_Order.msgflow file opens in the Message Flow editor.

2. In the Message Flow editor, add the nodes listed in Table 4-7 to the canvas, then connect the nodes together, as shown in Table 4-8, to build the ESQL_Book_Order message flow (Figure 4-29 on page 86).

Table 4-7   The ESQL_Book_Order message flow nodes

Node type	Node name
MQInput	ESQL_BOOKSTORE_BO_IN
Compute	Compute
MQOutput	ESQL_BOOKSTORE_BO_OUT

Table 4-8   Node connections in the ESQL_Book_Order message flow

Node name	Terminal	Connect to this node
ESQL_BOOKSTORE_BO_IN	Out	Compute
Compute	Out	ESQL_BOOKSTORE_BO_OUT

3. Set the properties of the nodes, as shown in Table 4-9.

Table 4-9   Node properties for the ESQL_Book_Order message flow

Node name	Page	Property	Value
ESQL_BOOKSTORE_BO_IN	Basic	Queue Name	ESQL_BOOKSTORE_BO_IN
	Default	Message Domain	XML
ESQL_BOOKSTORE_BO_OUT	Basic	Queue Name	ESQL_BOOKSTORE_BO_OUT

4. In the Message Flow editor, right-click the **Compute** node, then click **Open ESQL** to create the ESQL module that is referenced in the Compute node Properties dialog. The ESQL_Book_Order.esql file opens in the ESQL editor.

5. In the ESQL editor, edit the ESQL_Book_Order.esql file. The ESQL takes information from the input message, including the customer's selected delivery method. From this information, the message flow determines the delivery price and the total price of the order, then sends out a response message that contains this price information:

   a. Uncomment the fourth line in the ESQL_Book_Order.esql file (`--CALL CopyMessageHeaders();`) of the ESQL_Book_Order_Compute module so

that the Compute node can parse it. To uncomment the line, delete --
from the start of the line, as shown in Example 4-9. Leave the fifth line
commented out so that the whole of the input message is not copied to the
output message.

Example 4-9   Copying the message headers to the output message

```
CREATE COMPUTE MODULE ESQL_Book_Order_Compute
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        CALL CopyMessageHeaders();
        -- SET OutputRoot.XML.Book_Order_Response_MSG =
InputRoot.XML.Create_Book_Order_MSG;
```

b. Copy the Customer_ID field to the output message, as shown in
   Example 4-10.

Example 4-10   Copying the Customer_ID field to the output message

```
SET OutputRoot.XML.Book_Order_Response_MSG.Customer_ID =
InputRoot.XML.Create_Book_Order_MSG.Customer_ID;
```

c. Generate a unique order number for the customer's order. The value of
   the Order_Number element is a concatenation of the Customer_ID value
   and the Order_Date value. You can achieve this with the ESQL in
   Example 4-11, where || is the ESQL operator for concatenate.

Example 4-11   Generating a unique order number

```
SET OutputRoot.XML.Book_Order_Response_MSG.Order_Number =
InputRoot.XML.Create_Book_Order_MSG.Customer_ID || orderDate;
```

However, the ESQL in Example 4-11 produces an order number that looks
like this:

```
01234567892002-10-20 12:00:00
```

To generate a better order number, cast the timestamp field (Order_Date)
from a Date type to a String type. To do this, use the ESQL in
Example 4-12, which produces an order number like this:

```
01234567892005092712551
```

Example 4-12   Generating a more useful unique order number

```
DECLARE input TIMESTAMP InputRoot.XML.Create_Book_Order_MSG.Order_Date;
DECLARE pattern CHARACTER 'yyyyMMddHHmmss';
DECLARE orderDate CHARACTER CAST(input AS CHARACTER FORMAT pattern);
SET OutputRoot.XML.Book_Order_Response_MSG.Order_Number =
InputRoot.XML.Create_Book_Order_MSG.Customer_ID || orderDate;
```

d. Copy the Order_Date field to the output message, as shown in Example 4-13.

Example 4-13   Copying the Order_Date field to the output message

```
SET OutputRoot.XML.Book_Order_Response_MSG.Order_Date =
InputRoot.XML.Create_Book_Order_MSG.Order_Date;
```

e. Determine which delivery method has been selected by the customer. Do this by checking whether the message contains a particular field (First_Class, Second_Class, or Airmail). The ESQL in Example 4-14 checks to see if the First_Class field exists and contains the value Yes; if so, the ESQL copies the First_Class field to the output message. If the Second_Class field does not exist in the input message, the ESQL checks to see if the Second_Class field exists, and so on.

Example 4-14   Checking which delivery method the customer has selected

```
DECLARE deliveryPrice DECIMAL;
   IF InputRoot.XML.Create_Book_Order_MSG.First_Class = 'Yes' THEN
   SET OutputRoot.XML.Book_Order_Response_MSG.First_Class =
InputRoot.XML.Create_Book_Order_MSG.First_Class;
   END IF;

   IF InputRoot.XML.Create_Book_Order_MSG.Second_Class = 'Yes' THEN
   SET OutputRoot.XML.Book_Order_Response_MSG.Second_Class =
InputRoot.XML.Create_Book_Order_MSG.Second_Class;
   END IF;

   IF InputRoot.XML.Create_Book_Order_MSG.Airmail = 'Yes' THEN
   SET OutputRoot.XML.Book_Order_Response_MSG.Airmail =
InputRoot.XML.Create_Book_Order_MSG.Airmail;
   END IF;
```

f. While checking the delivery method, the message flow determines the delivery price based on the delivery method that was selected by the customer (Example 4-15).

Example 4-15   Determining the price of delivery based on which method was selected

```
DECLARE deliveryPrice DECIMAL;
   IF InputRoot.XML.Create_Book_Order_MSG.First_Class = 'Yes' THEN
   SET OutputRoot.XML.Book_Order_Response_MSG.First_Class =
InputRoot.XML.Create_Book_Order_MSG.First_Class;
   SET deliveryPrice = 18.00;
   END IF;

   IF InputRoot.XML.Create_Book_Order_MSG.Second_Class = 'Yes' THEN
```

```
       SET OutputRoot.XML.Book_Order_Response_MSG.Second_Class =
InputRoot.XML.Create_Book_Order_MSG.Second_Class;
   SET deliveryPrice = 12.00;
   END IF;

   IF InputRoot.XML.Create_Book_Order_MSG.Airmail = 'Yes' THEN
   SET OutputRoot.XML.Book_Order_Response_MSG.Airmail =
InputRoot.XML.Create_Book_Order_MSG.Airmail;
   SET deliveryPrice = 8.00;
   END IF;
```

g. Count the number of ISBN fields in the message to count how many books are listed in the order using the CARDINALITY statement. Then, for each instance of the Book_Details element in the message, copy the ISBN and Book_Price fields to the output message (Example 4-16). The output message then contains the XML for the Book_Details field.

Example 4-16   Counting the books in the order and copying their details to the output message

```
DECLARE bookCount INTEGER;
   DECLARE numBooks INTEGER;
   SET bookCount = 1;
   SET numBooks =
CARDINALITY(InputRoot.XML.Create_Book_Order_MSG.Book_Details.ISBN[]);
       WHILE bookCount<= numBooks  DO
           SET
OutputRoot.XML.Book_Order_Response_MSG.Book_Details[bookCount].ISBN =
InputRoot.XML.Create_Book_Order_MSG.Book_Details.ISBN[bookCount];
           SET
OutputRoot.XML.Book_Order_Response_MSG.Book_Details[bookCount].Book_Price =
InputRoot.XML.Create_Book_Order_MSG.Book_Details.Book_Price[bookCount];
           SET bookCount=bookCount+1;
       END WHILE;
```

h. Calculate the total price of the books using the variable sumBookPrice. Declare the sumBookPrice variable before the WHILE statement. At the beginning of the WHILE statement, set the value of sumBookPrice to be itself plus the price of the current book. The price of the current book is cast to Decimal type so that it can be added to the decimal sumBookPrice.

Example 4-17   Calculating the total price of the books in the order

```
DECLARE bookCount INTEGER;
   DECLARE numBooks INTEGER;
   DECLARE sumBookPrice DECIMAL 0;
   SET bookCount = 1;
   SET numBooks =
CARDINALITY(InputRoot.XML.Create_Book_Order_MSG.Book_Details.ISBN[]);
```

```
        WHILE bookCount<= numBooks  DO
            SET sumBookPrice = sumBookPrice +
CAST(InputRoot.XML.Create_Book_Order_MSG.Book_Details.Book_Price[bookCount] AS
DECIMAL);
            SET
OutputRoot.XML.Book_Order_Response_MSG.Book_Details[bookCount].ISBN =
InputRoot.XML.Create_Book_Order_MSG.Book_Details.ISBN[bookCount];
            SET
OutputRoot.XML.Book_Order_Response_MSG.Book_Details[bookCount].Book_Price =
InputRoot.XML.Create_Book_Order_MSG.Book_Details.Book_Price[bookCount];
            SET bookCount = bookCount + 1;
        END WHILE;
```

i. Finally, add three fields to the output message: The Delivery_Price, which
    was calculated in Example 4-15 on page 89; the Total_Price for the order,
    which was calculated in Example 4-16 on page 90; and the Order_Status,
    which is assigned the value of *Order Received* (Example 4-18).

Example 4-18   Adding delivery price, total price, and order status to output message

```
SET OutputRoot.XML.Book_Order_Response_MSG.Delivery_Price = deliveryPrice;
   SET OutputRoot.XML.Book_Order_Response_MSG.Total_Price = sumBookPrice;
   SET OutputRoot.XML.Book_Order_Response_MSG.Order_Status = 'Order Received';
```

The complete ESQL module is shown in Example 4-19.

Example 4-19   The complete ESQL_Book_Order_Compute module

```
CREATE COMPUTE MODULE ESQL_Book_Order_Compute
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        CALL CopyMessageHeaders();
        -- SET OutputRoot.XML.Book_Order_Response_MSG =
InputRoot.XML.Create_Book_Order_MSG;
        SET OutputRoot.XML.Book_Order_Response_MSG.Customer_ID =
InputRoot.XML.Create_Book_Order_MSG.Customer_ID;
        DECLARE input TIMESTAMP InputRoot.XML.Create_Book_Order_MSG.Order_Date;
        DECLARE pattern CHARACTER 'yyyyMMddHHmmss';
        DECLARE orderDate CHARACTER CAST(input AS CHARACTER FORMAT pattern);
        SET OutputRoot.XML.Book_Order_Response_MSG.Order_Number =
InputRoot.XML.Create_Book_Order_MSG.Customer_ID || orderDate;
        SET OutputRoot.XML.Book_Order_Response_MSG.Order_Date =
InputRoot.XML.Create_Book_Order_MSG.Order_Date;

        DECLARE deliveryPrice DECIMAL;
        IF InputRoot.XML.Create_Book_Order_MSG.First_Class = 'Yes' THEN
        SET OutputRoot.XML.Book_Order_Response_MSG.First_Class =
InputRoot.XML.Create_Book_Order_MSG.First_Class;
        SET deliveryPrice = 18.00;
```

```
        END IF;

        IF InputRoot.XML.Create_Book_Order_MSG.Second_Class = 'Yes' THEN
        SET OutputRoot.XML.Book_Order_Response_MSG.Second_Class =
InputRoot.XML.Create_Book_Order_MSG.Second_Class;
        SET deliveryPrice = 12.00;
        END IF;

        IF InputRoot.XML.Create_Book_Order_MSG.Airmail = 'Yes' THEN
        SET OutputRoot.XML.Book_Order_Response_MSG.Airmail =
InputRoot.XML.Create_Book_Order_MSG.Airmail;
        SET deliveryPrice = 8.00;
        END IF;

    DECLARE bookCount INTEGER;
    DECLARE numBooks INTEGER;
    DECLARE sumBookPrice DECIMAL 0;
    SET bookCount = 1;
    SET numBooks =
CARDINALITY(InputRoot.XML.Create_Book_Order_MSG.Book_Details.ISBN[]);
        WHILE bookCount<= numBooks  DO
            SET sumBookPrice = sumBookPrice +
CAST(InputRoot.XML.Create_Book_Order_MSG.Book_Details.Book_Price[bookCount] AS
DECIMAL);
            SET
OutputRoot.XML.Book_Order_Response_MSG.Book_Details[bookCount].ISBN =
InputRoot.XML.Create_Book_Order_MSG.Book_Details.ISBN[bookCount];
            SET
OutputRoot.XML.Book_Order_Response_MSG.Book_Details[bookCount].Book_Price =
InputRoot.XML.Create_Book_Order_MSG.Book_Details.Book_Price[bookCount];
            SET bookCount = bookCount + 1;
        END WHILE;

    SET OutputRoot.XML.Book_Order_Response_MSG.Delivery_Price = deliveryPrice;
    SET OutputRoot.XML.Book_Order_Response_MSG.Total_Price = sumBookPrice;
    SET OutputRoot.XML.Book_Order_Response_MSG.Order_Status = 'Order Received';
        -- CALL CopyEntireMessage();
        RETURN TRUE;
    END;

    CREATE PROCEDURE CopyMessageHeaders() BEGIN
        DECLARE I INTEGER;
        DECLARE J INTEGER;
        SET I = 1;
        SET J = CARDINALITY(InputRoot.*[]);
        WHILE I < J DO
            SET OutputRoot.*[I] = InputRoot.*[I];
            SET I = I + 1;
        END WHILE;
```

```
        END;

        CREATE PROCEDURE CopyEntireMessage() BEGIN
            SET OutputRoot = InputRoot;
        END;
END MODULE;
```

6. Save the ESQL_Book_Order.esql and ESQL_Book_Order.msgflow files.

7. Create the following WebSphere MQ queues in WebSphere MQ Explorer on the WBRK6_DEFAULT_QUEUE_MANAGER queue manager:

   – ESQL_BOOKSTORE_BO_IN
   – ESQL_BOOKSTORE_BO_OUT

   Remember to enter the value of the Backout requeue queue property on the ESQL_BOOKSTORE_BO_IN queue as DLQ.

You have created the ESQL_Book_Order message flow. Next, deploy and test both the ESQL_Create_Customer_Account message flow and the ESQL_Book_Order message flow.

4.3.4 Deploying and testing the ESQL Bookstore message flows

To test the ESQL Bookstore message flows, ESQL_Create_Customer_Account and ESQL_Book_Order, you must deploy them to the broker.

To deploy the ESQL Bookstore message flows to the broker:

1. Switch to the Broker Administration perspective.

2. Create a bar file called ESQL_Bookstore.bar.

3. Add to the bar file the ESQL_Create_Customer_Account.msgflow file and the ESQL_Book_Order.msgflow file, then save the bar file.

4. Create a new execution group on the WBRK6_DEFAULT_BROKER broker called ESQL_Bookstore.

5. Ensure that the WBRK6_DEFAULT_BROKER broker and the WBRK6_DEFAULT_CONFIGURATION_MANAGER Configuration Manager are running, then deploy the ESQL_Bookstore.bar file to the ESQL_Bookstore execution group.

   In the Domains view, the two message flows are displayed under the ESQL_Bookstore execution group.

6. Create a new enqueue file called ESQL_Create_Customer_Account.enqueue and use it to put the message in Example 4-3 on page 80 on the ESQL_BOOKSTORE_CCA_IN queue on the WBRK6_DEFAULT_QUEUE_MANAGER queue manager. You can copy the

message content from the Web material available to download (see Appendix B, "Code" on page 319).

7. Use the Dequeue wizard to get the output message, which should contain the same message data as the input message, from the ESQL_BOOKSTORE_CCA_OUT queue on the same queue manager.

8. Check that the CUSTACCTB table in the BSTOREDB database has been updated with the information from the input message:

    a. Open the DB2 Control Center: Click **Start** → **Programs** → **IBM DB2** → **General Administration Tools** → **Control Center**.

    b. In the left pane of the Control Center window, expand **All Databases** → **BSTOREDB**, then click the **Tables** folder. All the tables in the BSTOREDB database are listed in the top-right pane.

    c. Double-click the **CUSTACCTB** table. The Open Table dialog opens, displaying the data in the table.

       The CUSTACCTB table should contain a row of data for each time you put a message through the ESQL_Create_Customer_Account message flow.

    d. Try changing some of the field values in the input message in the ESQL_Create_Customer_Account.enqueue file (for example, change the name of the customer and the customer's password), then put the message through the message flow. Another row is added to the CUSTACCTB table with the values that you entered in the input message.

9. Create a new enqueue file called ESQL_Book_Order.enqueue in which to put the message in Example 4-7 on page 85. You can copy the message content from the Web material available to download (see Appendix B, "Code" on page 319).

10. Use the Dequeue wizard to get the output message, which should contain the message in Example 4-8 on page 86, from the ESQL_BOOKSTORE_BO_OUT queue on the same queue manager.

11. Try changing the details in the input message in ESQL_Book_Order.enqueue; for example, add another book to the order or change the price of one of the books. Put the message through the message flow and check the output message to see how your changes affected the output message.

If the message does not output the correct message, or if the message flow cannot process the message, see **Chapter 8, "Troubleshooting and problem determination" on page 241**, for information about problem determination.

4.4 Summary

You have now created, deployed, and tested two message flow applications in which you defined the logic of the message flows using ESQL.

In the next chapter we create the same message flow applications using Java in a JavaCompute node instead of ESQL in Compute and Database nodes.

For more information about the built-in nodes that are available in WebSphere Message Broker, see the product documentation: **Developing applications** → **Developing message flow applications** → **Designing a message flow** → **Deciding which nodes to use**.

5

Developing applications with Java

This chapter describes how to develop message flow applications in the Message Brokers Toolkit using Java to define the logic of the message flows.

The following topics are discussed:

► Defining the logic of a message flow using Java
► Java and the Java editor in the Message Brokers Toolkit
► Inserting data into a database using a message flow
► Transforming a message from one XML structure to another

97

5.1 Developing message flow applications with Java

A message flow application is a program that processes messages in the broker. Message flow applications can transform messages between different formats, generate new messages based on other messages, and route messages according to the message's content or according to how the message flow is configured.

See 4.1.1, "Messages in WebSphere Message Broker" on page 48, for more information about messages.

In Chapter 4, "Developing applications with ESQL" on page 47, you created two message flow applications in which the logic of the message flows was defined using ESQL. This chapter describes how to develop and define the logic of the same message flow applications using Java.

5.1.1 Java and the Java editor

Java is an object-oriented language that you can use to define the logic of message flows instead of ESQL. WebSphere Message Broker provides a library of functions that make it easy to reference elements in messages and to manipulate messages from a JavaCompute node. Each JavaCompute node references a Java class that is stored in a special Java project, separate from the Message Flow project that references it. Use the Java editor, in the Java perspective of the Message Brokers Toolkit, to edit the Java class. The Java editor validates your Java class and, while you are editing, you can get assistance by pressing Ctrl+Spacebar (or by selecting Content Assist from the Edit menu) to open the code assist window, as shown in Figure 5-1 on page 99.

Tip: To show line numbers in the Java editor, click **Window** → **Preferences**. On the Java Æ Editor page of the Preferences dialog, select **Show line numbers**.

Figure 5-1   The Java editor

The message flow applications described in this chapter perform the same tasks as the message flow applications described in Chapter 4, "Developing applications with ESQL" on page 47, but they use Java in JavaCompute nodes instead of ESQL in Compute and Database nodes. The JavaCompute node can be configured to perform the same range of tasks as the ESQL nodes (Compute, Database, and Filter), including manipulating messages, accessing and updating database tables, creating new messages, and filtering message content.

5.1.2 Scenarios described in this chapter

This chapter focuses on how to define the logic of message flows with Java. We provide step-by-step instructions to create, deploy, and test two message flow applications:

► Simple message flow application

The Simple message flow application demonstrates how to build a very basic message flow from three nodes. The Java_Simple message flow takes an XML input message from a WebSphere MQ queue, uses Java in a JavaCompute node to build an XML output message that has the same contents as the input message, then puts the output message on another WebSphere MQ queue.

► Bookstore message flow application

The Bookstore message flow application is based around the scenario of an online bookstore. The first message flow, Java_Create_Customer_Account, uses Java in a JavaCompute node to create accounts in a DB2 database table for new customers who have registered their details with the bookstore, for example, their contact details and delivery address. The second message flow, Java_Book_Order, uses Java in a JavaCompute node to process an order that has been submitted by an online customer and create a response message to confirm the order with a unique order number.

You do not need skills or experience in coding Java to be able to create the message flow applications in this chapter because all the code is provided in the Web material described in Appendix B, "Code" on page 319.

5.1.3 Before you start

The instructions in this chapter assume that you have run the Default Configuration wizard to create the default configuration. However, you can create your own broker domain and substitute the component names when following the instructions.

For more information about the Default Configuration wizard, see 3.5, "Verifying the installation" on page 35. For more information about administering components, see "Starting the components" on page 213.

Ensure that the broker and the Configuration Manager are running.

Starting the broker and the Configuration Manager

You cannot start components from the Message Brokers Toolkit; you must start them from the command line. Enter all commands in a WebSphere Message Broker Command Console, which is a command window with additional WebSphere Message Broker Environment settings.

To start the Command Console, click **Start** → **Programs** → **IBM WebSphere Message Brokers 6.0** → **Command Console**.

To start the broker, enter the following command in the Command Console, where WBRK6_DEFAULT_BROKER is the name of the broker in the Default Configuration:

```
mqsistart WBRK6_DEFAULT_BROKER
```

To start the Configuration Manager, enter the following command in the Command Console, where WBRK6_DEFAULT_CONFIGURATION_MANAGER is the name of the Configuration Manager in the Default Configuration:

```
mqsistart WBRK6_DEFAULT_CONFIGURATION_MANAGER
```

Open the Windows Event Viewer to check that the components have started without any problems. See 8.1.5, "Windows Event Viewer" on page 253, for information about how to access and view entries in the Windows Event Viewer.

This chapter also assumes that you have already completed the exercises in Chapter 4, "Developing applications with ESQL" on page 47. The exercises in this chapter do not depend on those in Chapter 4, but less detail is given in the instructions in this chapter. Refer to the step-by-step instructions in Chapter 4 if you need more details when creating, deploying, and testing the Java versions of the message flow applications.

5.2 Developing the Simple message flow application

Each message flow is stored in a message flow file with the extension .msgflow. The message flow file is, in turn, stored in a Message Flow project. Any Java classes (.java and, when compiled, .class) referenced from this project are stored in a separate Java project.

When you have created the files that contain the message flow, add, connect, and configure the message flow nodes in the Message Flow editor. Deploy the message flow to the broker so that you can test it.

5.2.1 Creating the Java_Simple message flow

To create the files in which the Java_Simple message flow is stored:

1. In the Broker Application Development perspective, create a Message Flow project called Java_Simple Message Flow Project.

2. Create a message flow called Java_Simple in the Java_Simple Message Flow Project.

The Java_Simple.msgflow file is displayed in the Java_Simple Message Flow Project in the Resource Navigator view. The Java_Simple.msgflow file opens automatically in the Message Flow editor.

For detailed instructions on creating a message flow, see 4.2.1, "Creating the ESQL_Simple message flow" on page 53.

Adding and connecting the Java_Simple nodes

Figure 5-2 on page 102 shows the finished Java_Simple message flow.
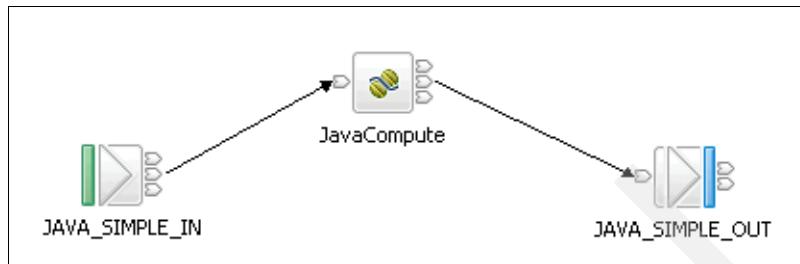
Figure 5-2   The Java_Simple message flow

To create the Java_Simple message flow:

1. In the Message Flow editor, add the nodes listed in Table 5-1 to the canvas,.

2. Connect the nodes together, as shown in Table 5-2.

Table 5-1   The Java_Simple message flow nodes

Node type	Node name
MQInput	JAVA_SIMPLE_IN
JavaCompute	JavaCompute
MQOutput	JAVA_SIMPLE_OUT

Table 5-2   Node connections in the Java_Simple message flow

Node name	Terminal	Connect to this node
JAVA_SIMPLE_IN	Out	JavaCompute
JavaCompute	Out	JAVA_SIMPLE_OUT

3. Save the Java_Simple.msgflow file.

Figure 5-2 shows how the Java_Simple message flow looks when all the nodes are connected together, saved, and validated.

For detailed instructions on adding and connecting nodes, see 4.2.1, "Creating the ESQL_Simple message flow" on page 53.

5.2.2 Configuring the Java_Simple message flow

To configure the Java_Simple message flow:

1. In WebSphere MQ Explorer, create the following queues on the WBRK6_DEFAULT_QUEUE_MANAGER queue manager:

   – JAVA_SIMPLE_IN
   – JAVA_SIMPLE_OUT

2. Set the Backout requeue queue of the JAVA_SIMPLE_IN queue to DLQ so that if the message flow fails to process the message and rolls it back to the JAVA_SIMPLE_IN queue, the message is put on the DLQ queue and does not block the processing of subsequent messages. You do not need to create another DLQ queue; the Java_Simple message flow can use the DLQ queue that you created for the ESQL_Simple message flow.

3. Set the properties of the Java_Simple message flow nodes as shown in Table 5-3.

Table 5-3   Node properties for the Java_Simple message flow

Node name	Page	Property	Value
JAVA_SIMPLE_IN	Basic	Queue Name	JAVA_SIMPLE_IN
	Default	Message Domain	XML
JAVA_SIMPLE_O UT	Basic	Queue Name	JAVA_SIMPLE_O UT

For detailed instructions on configuring a message flow, see 4.2.2, "Configuring the ESQL_Simple message flow" on page 58.

5.2.3 Writing Java for the Java_Simple message flow

All of the Java that belongs to the message flow is stored in a Java project. In this case, all of the Java for the Java_Simple message flow is stored in a Java project called Java_SimpleJava.

Creating the Java project

To create the Java project for the Java_Simple message flow:

1. In the Message Flow editor, right-click the **JavaCompute** node, then click **Open Java**. The New Java Compute Node Class wizard opens.

2. In the wizard, the project name is entered automatically: `Java_Simple Message Flow ProjectJava`. Accept this name by clicking **Next** (Figure 5-3 on page 104).
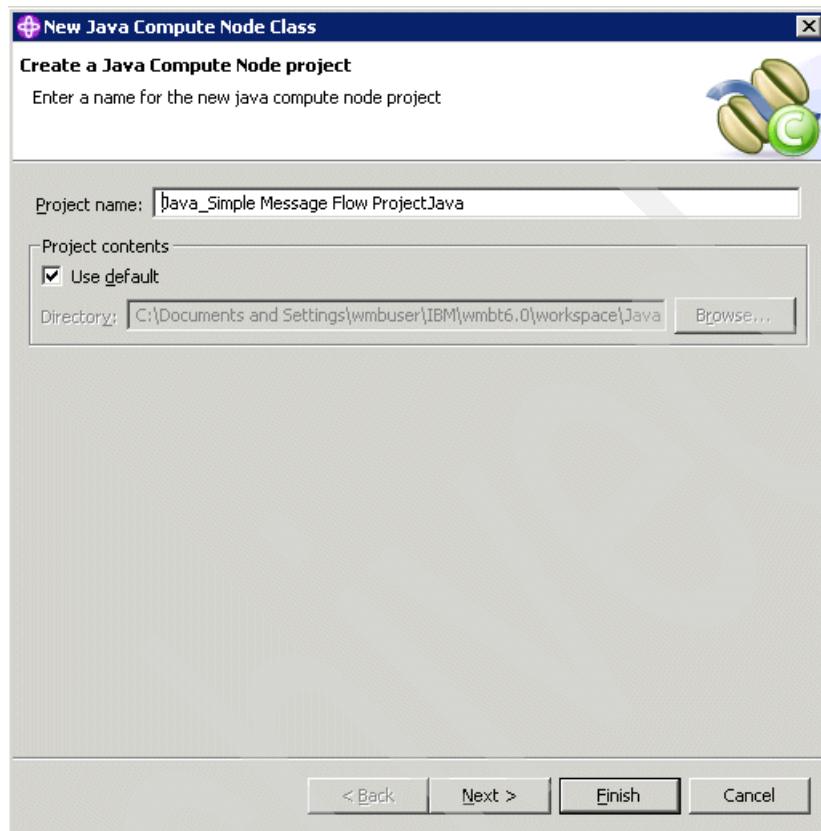
Figure 5-3   Accepting the name of the new Java project

3. Click **Next** twice to accept the default values (Figure 5-4 on page 105 and Figure 5-5 on page 106). While developing this simple scenario, ignore warnings in the wizard banner about not accepting the default package name.
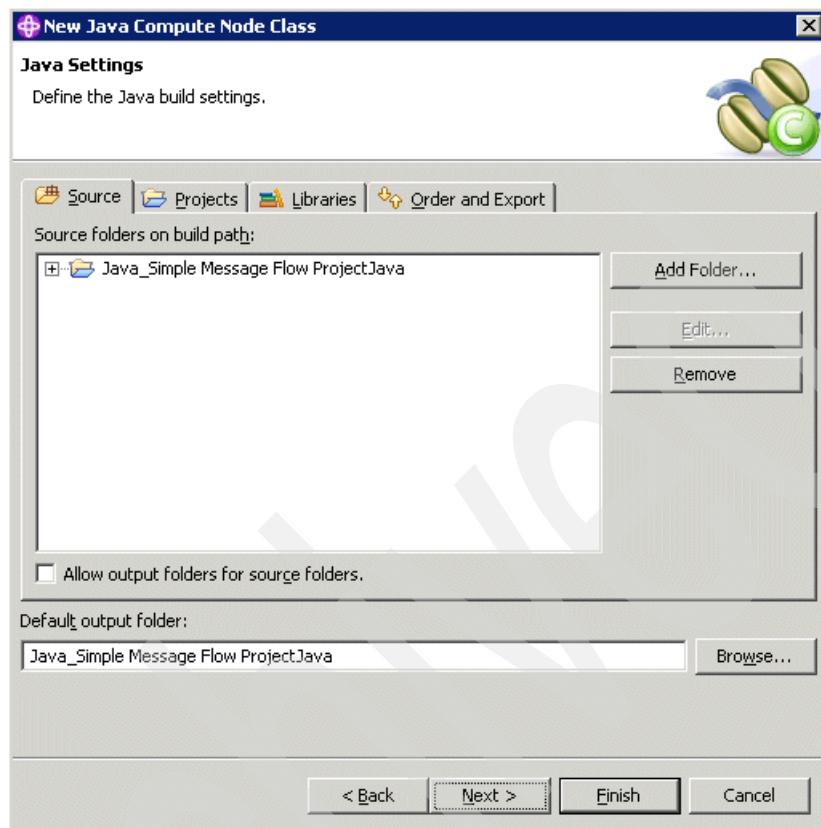
*Figure 5-4   Accepting default values for the Java build settings*
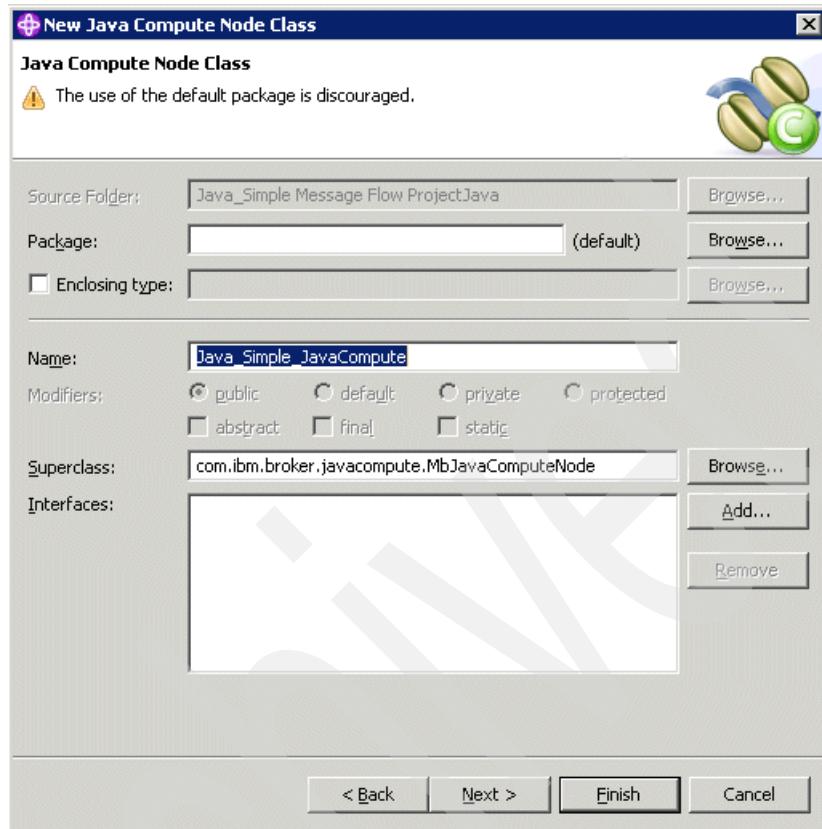
Figure 5-5   Accepting default values for the package name

4. On the Java Compute Node Class Template page of the wizard, click
   Filtering message class, then click **Finish** (Figure 5-6 on page 107).
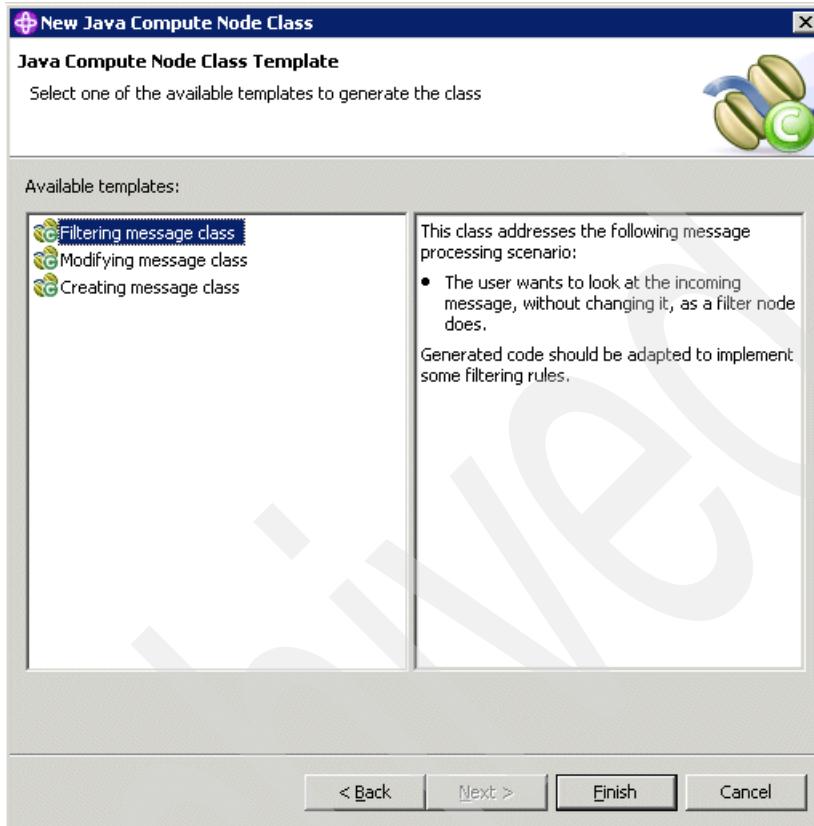
Figure 5-6   Selecting the class template to use

5. If you have not used the Java perspective previously, a message is displayed
    to ask whether to switch to the Java perspective now. Click **Yes**. The
    Message Brokers Toolkit switches to the Java perspective.

    A new Java project, Java_Simple Message Flow ProjectJava, is displayed in
    the Package Explorer view, which is where all the projects are displayed in
    the Java perspective. If you switch back to the Broker Application
    Development perspective, the new project is also displayed in the Resource
    Navigator view.

6. Save the Java_Simple.msgflow file so that the file is validated. Now that the
    Java_Simple_JavaCompute.java file has been created, all the errors in the
    Java_Simple Message Flow Project should be resolved (Figure 5-7 on
    page 108).
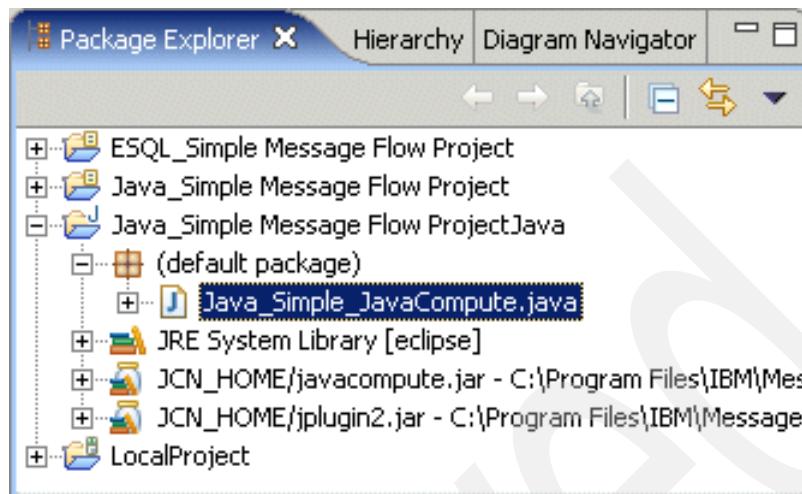
Figure 5-7   The Package Explorer view in the Java perspective

The Java_Simple_JavaCompute class

The Java_Simple_JavaCompute.java file contains the Java class that is referenced from the JavaCompute node Properties dialog. For each JavaCompute node that you add to a message flow, create another Java class.

The Java class that is generated automatically in the Java_Simple_JavaCompute.java file copies the content of the input message to a new output message like the ESQL does in the ESQL_Simple message flow.

The Java_Simple_JavaCompute class configures the Java_Simple message flow to perform the same message manipulations as the ESQL_Simple_Compute module in the ESQL_Simple message flow (see "Writing the ESQL_Simple_Compute ESQL module" on page 66). For example, the line `MbMessage message = assembly.getMessage();` in the Java_Simple_JavaCompute Java class produces the same output as `SET OutputRoot = InputRoot` in the ESQL_Simple_Compute ESQL module.

Do not edit the code in the Java_Simple_JavaCompute.java file; just save the file.

5.2.4 Deploying and testing the Java_Simple message flow

To test the Java_Simple message flow, you must deploy it to the broker.

To deploy the Java_Simple message flow to the broker:

1. Switch to the Broker Administration perspective.

2. Create a bar file called Java_Simple.bar.

3. Add to the bar file the Java_Simple.msgflow file from Java_Simple Message Flow Project, then save the bar file. The Java class is automatically pulled into the bar file with the message flow file and compiled.

4. Create a new execution group on the WBRK6_DEFAULT_BROKER broker called Java_Simple.

5. Ensure that the WBRK6_DEFAULT_BROKER and the WBRK6_DEFAULT_CONFIGURATION_MANAGER Configuration Manager are running, then deploy the Java_Simple.bar file to the Java_Simple execution group.

   In the Domains view, the Java_Simple message flow and the Java_Simple Message Flow ProjectJava jar file are displayed under the Java_Simple execution group.

6. Create a new message enqueue file called Java_Simple.enqueue in Java_Simple Message Flows Project.

7. Edit the Java_Simple.enqueue file so that it connects to the WBRK6_DEFAULT_QUEUE_MANAGER and puts the message on the JAVA_SIMPLE_IN queue.

8. In the Message data field, enter the XML message content shown in Example 5-1. The input message for the Java_Simple message flow is the same as for the ESQL_Simple message flow. You can copy the message content from the Web material available to download (see Appendix B, "Code" on page 319).

Example 5-1   Input message content to test the Java_Simple message flow

```
Message>
    <Body>
        Hello, world!
    </Body>
</Message>
```

9. Save the Java_Simple.enqueue file, then in the Enqueue editor, click **Write To queue**. The message is put on the JAVA_SIMPLE_IN queue.

10. Use the Dequeue wizard to get the output message, which should contain the same message data as the input message, from the JAVA_SIMPLE_OUT queue on the same queue manager.

For detailed instructions, see 4.2.4, "Deploying and testing the ESQL_Simple message flow" on page 67.

If the Java_Simple message flow fails to process the message, see 4.2.5, "Diagnosing problems with the ESQL_Simple message flow" on page 78, for how to diagnose problems. Also see **Chapter 8, "Troubleshooting and problem determination" on page 241**, for more information about problem determination.

> **Restriction:** If you find a problem in the Java class after you have deployed it, you must delete all the contents of the bar file and save the bar file. Then add the message flow files to the bar file again. This ensures that the bar file is correctly updated with your changes to the Java class.

5.3 Developing the Bookstore scenario using Java

In 5.2, "Developing the Simple message flow application" on page 101, you created the Simple scenario message flow application using Java to define the logic of the message flow.

In this section we create a more complex message flow application that is based around the scenario of an online bookstore. The Bookstore scenario message flows process messages with different structures, and interact with databases to update database tables.

The Bookstore scenario includes two message flows:

► The Java_Create_Customer_Account message flow

This message flow uses Java in a JavaCompute node to create accounts in a DB2 database table for new customers who have registered their details with the bookstore, for example, their contact details and delivery address.

► The Java_Book_Order message flow

This message flow uses Java in a JavaCompute node to process an order that has been submitted by an online customer and create a response message to confirm the order with a unique order number.

The message flow applications in this chapter use the same DB2 database that you created for the ESQL Bookstore message flow applications. You do not need to re-create the database for this chapter. For more information about the database, see 4.3.1, "Creating the Bookstore scenario database" on page 80.

5.3.1 Creating the Java_Create_Customer_Account message flow

The input message to the Java_Create_Customer_Account message flow contains the details of a customer who has registered with the online bookstore Web site. The customer has provided information such as their name, contact

details, a delivery address, and payment details. Example 5-2 on page 111 shows the input message for the Java_Create_Customer_Account message flow.

Example 5-2   The message for the Java_Create_Customer_Account message flow

```
<Create_Customer_Account_MSG>
   <Personal_Details>
      <First_Name>Peter</First_Name>
      <Last_Name>Smith</Last_Name>
      <User_ID>PSmith</User_ID>
      <Password>p45sw0rd</Password>
   </Personal_Details>
   <Email_Address>Peter.Smith@nowhere.com</Email_Address>
   <Daytime_Telephone>1234567890</Daytime_Telephone>
   <Evening_Telephone>1234567890</Evening_Telephone>
   <Shipping_Address>
      <Address_1>19 Green Street</Address_1>
      <Address_2>Littleton</Address_2>
      <Town>Southington</Town>
      <Postcode>SU29 8YT</Postcode>
   </Shipping_Address>
   <Billing_Address>
      <Address_1>19 Green Street</Address_1>
      <Address_2>Littleton</Address_2>
      <Town>Southington</Town>
      <Postcode>SU29 8YT</Postcode>
   </Billing_Address>
   <Payment_Details>
      <Card>VISA</Card>
      <Card_Number>1234567890</Card_Number>
      <Expiry_Date>12.09.2009</Expiry_Date>
      <Issue_Date>12.09.2004</Issue_Date>
      <Issue_Number>02</Issue_Number>
      <Security_Code>333</Security_Code>
   </Payment_Details>
</Create_Customer_Account_MSG>
```

All of the Java that you need for this message flow is available in the Web material described in Appendix B, "Code" on page 319.

Figure 5-8 on page 112 shows the finished Java_Create_Customer_Account message flow.
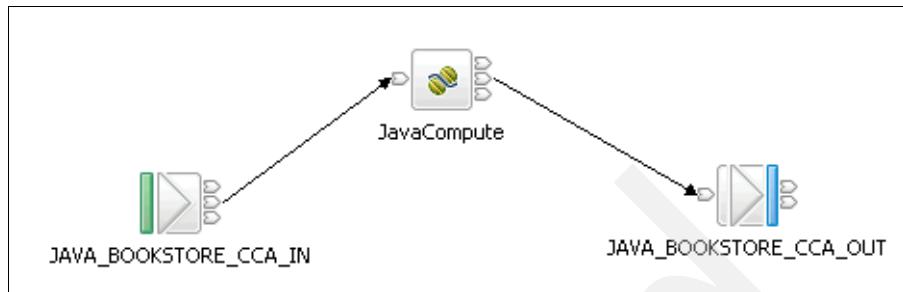
Figure 5-8   The Java_Create_Customer_Account message flow

To create the Java_Create_Customer_Account message flow:

1. In the Broker Application Development perspective, create a Message Flow project called `Java_Bookstore Message Flow Project`.

2. Create a message flow called `Java_Create_Customer_Account` in the Java_Bookstore Message Flow Project. The Java_Create_Customer_Account.msgflow file opens in the Message Flow editor.

3. In the Message Flow editor, add the nodes listed in Table 5-4 to the canvas, then connect the nodes together, as shown in Table 5-4, to build the Java_Create_Customer_Account message flow (Table 5-5).

Table 5-4   The Java_Create_Customer_Account message flow nodes

Node type	Node name
MQInput	JAVA_BOOKSTORE_CCA_IN
JavaCompute	JavaCompute
MQOutput	JAVA_BOOKSTORE_CCA_OUT

Table 5-5   Node connections in the Java_Create_Customer_Account message flow

Node name	Terminal	Connect to this node
JAVA_BOOKSTORE_CCA _IN	Out	JavaCompute
JavaCompute	Out	JAVA_BOOKSTORE_CCA _IN

4. Set the properties of the nodes, as shown in Table 5-6 on page 113.

Table 5-6   Node properties for the Java_Create_Customer_Account message flow

Node name	Page	Property	Value
JAVA_BOOKSTORE_CCA_IN	Basic	Queue Name	JAVA_BOOKSTORE_CCA_IN
	Default	Message Domain	XML
JAVA_BOOKSTORE_CCA_OUT	Basic	Queue Name	JAVA_BOOKSTORE_CCA_OUT

5. In the Message Flow editor, right-click the **JavaCompute** node, then click **Open Java** to create the Java class that is referenced in the JavaCompute node Properties dialog. The New Java Compute Node Class wizard opens.

6. In the wizard, accept the Project name by clicking **Next**.

7. On the Java Settings page of the wizard, click **Next**.

8. In the Package field, type `com.ibm.itso.wmb6.basics.bookstore` (Figure 5-9 on page 114), then click **Next**.
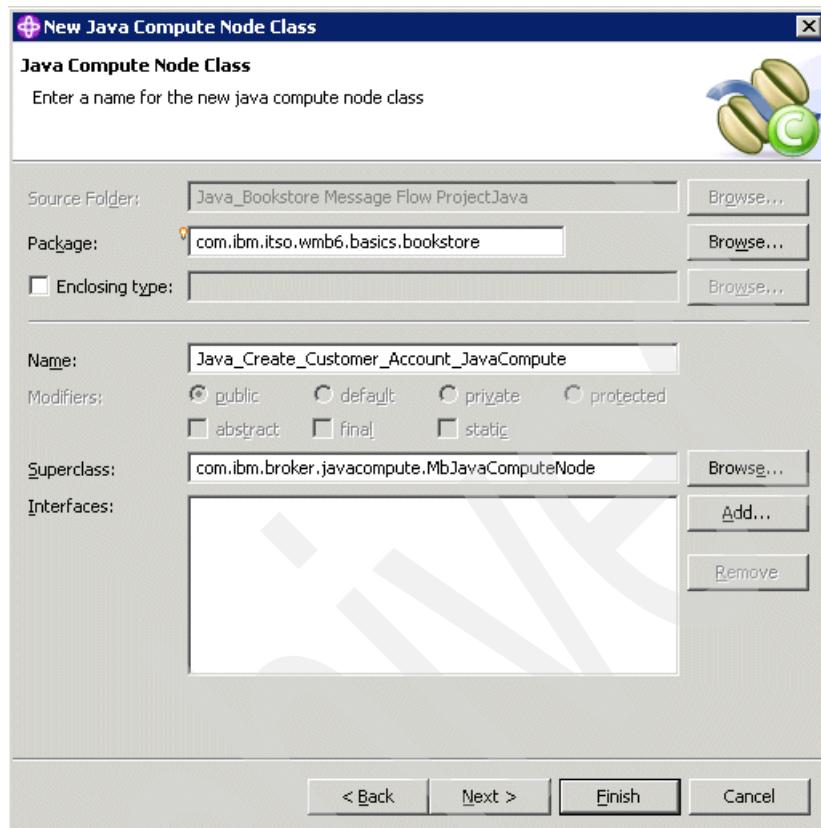
Figure 5-9   Entering package name in New Java Compute Node Class wizard

9. On the Java Compute Node Class Template page of the wizard, click
    Filtering message class, then click **Finish**. The Message Brokers Toolkit
    switches to the Java perspective, the new Java_Bookstore Message Flow
    ProjectJava project is displayed in the Package Explorer view, and the
    Java_Create_Customer_Account_JavaCompute.java file opens in the Java
    editor.

10. In the Java editor, edit the
    Java_Create_Customer_Account_JavaCompute.java file by inserting Java
    code between the `\\Add user code below` and `\\End of user code`
    comments. The Java selects content from the input message and inserts it
    into the CUSTACCTB table in the BSTOREDB database:

    a. Use SQL to update the database table CUSTACCTB with information from
        the message. Because you are writing in Java, you must create the SQL
        statement, as shown in Example 5-3 on page 115. Each field in the

database table (for example, LAST_NAME, FIRST_NAME, USERID) is listed in the order in which it occurs in the database.

Example 5-3   Using SQL to update the CUSTACCTB database table

```
MbSQLStatement state = createSQLStatement( "BSTOREDB",
     "INSERT INTO Database.CUSTACCTB(LAST_NAME, FIRST_NAME, USERID, PASSWORD,
EMAIL, DAY_PHONE, EVE_PHONE, SHIP_ADDRESS1, SHIP_ADDRESS2, SHIP_TOWN,
SHIP_POSTCODE, BILL_ADDRESS1, BILL_ADDRESS2, BILL_TOWN, BILL_POSTCODE,
CARDTYPE, CARDNUM, EXP_DATE, ISS_DATE, ISS_NUM, SECCODE) " +
```

b. Specify each value from the message that is to be added to the CUSTACCTB table. One value is inserted into each field of the database table. Java specifies fields by navigating through the hierarchical structure of the message from the root of the message body, which is known as *Body*. Compare the values in the Java in Example 5-4 with the XML fields in Example 5-2 on page 111 to see how the Java navigates the message fields.

The order that the values are listed in the Java determines the field into which each value is inserted. So the InputRoot.XML.Create_Customer_Account_MSG.Personal_Details.Last_Name value is inserted into the LAST_NAME field of the CUSTACCTB table and the InputRoot.XML.Create_Customer_Account_MSG.Payment_Details.Security_Code value is inserted into the SECCODE field of the CUSTACCTB table.

Example 5-4   Specifying the values to insert into the CUSTACCTB table

```
"VALUES(InputRoot.XML.Create_Customer_Account_MSG.Personal_Details.Last_Name, "
+
    "InputRoot.XML.Create_Customer_Account_MSG.Personal_Details.First_Name, " +
    "InputRoot.XML.Create_Customer_Account_MSG.Personal_Details.User_ID, " +
    "InputRoot.XML.Create_Customer_Account_MSG.Personal_Details.Password, " +
    "InputRoot.XML.Create_Customer_Account_MSG.Email_Address, " +
    "InputRoot.XML.Create_Customer_Account_MSG.Daytime_Telephone, " +
    "InputRoot.XML.Create_Customer_Account_MSG.Evening_Telephone, " +
    "InputRoot.XML.Create_Customer_Account_MSG.Shipping_Address.Address_1, " +
    "InputRoot.XML.Create_Customer_Account_MSG.Shipping_Address.Address_2, " +
    "InputRoot.XML.Create_Customer_Account_MSG.Shipping_Address.Town, " +
    "InputRoot.XML.Create_Customer_Account_MSG.Shipping_Address.Postcode, " +
    "InputRoot.XML.Create_Customer_Account_MSG.Billing_Address.Address_1, " +
    "InputRoot.XML.Create_Customer_Account_MSG.Billing_Address.Address_2, " +
    "InputRoot.XML.Create_Customer_Account_MSG.Billing_Address.Town, " +
    "InputRoot.XML.Create_Customer_Account_MSG.Billing_Address.Postcode, " +
    "InputRoot.XML.Create_Customer_Account_MSG.Payment_Details.Card, " +
    "InputRoot.XML.Create_Customer_Account_MSG.Payment_Details.Card_Number, " +
    "InputRoot.XML.Create_Customer_Account_MSG.Payment_Details.Expiry_Date, " +
```

```
    "InputRoot.XML.Create_Customer_Account_MSG.Payment_Details.Issue_Date, " +
    "InputRoot.XML.Create_Customer_Account_MSG.Payment_Details.Issue_Number, "
+

"InputRoot.XML.Create_Customer_Account_MSG.Payment_Details.Security_Code);");
```

Example 5-5 shows the final Java code to use in the
Java_Create_Customer_Account_JavaCompute class.

Example 5-5   Java for the Java_Create_Customer_Account_JavaCompute class

```
/*
 * Created on 24-Oct-2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.ibm.itso.wmb6.basics.bookstore;

import com.ibm.broker.javacompute.MbJavaComputeNode;
import com.ibm.broker.plugin.*;

/**
 * @author wmbuser
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class Java_Create_Customer_Account_JavaCompute extends MbJavaComputeNode
{

    public void evaluate(MbMessageAssembly assembly) throws MbException {
        MbOutputTerminal out = getOutputTerminal("out");
        MbOutputTerminal alt = getOutputTerminal("alternate");

        MbMessage message = assembly.getMessage();

        // -----------------------------------------------------------
        // Add user code below
        MbMessage newMsg = new MbMessage(assembly.getMessage());
        MbMessageAssembly newAssembly = new MbMessageAssembly(assembly, newMsg);

        String table = "dbTable";

        MbSQLStatement state = createSQLStatement( "BSTOREDB",
        "INSERT INTO Database.CUSTACCTB(LAST_NAME, FIRST_NAME, USERID, PASSWORD,
EMAIL, DAY_PHONE, EVE_PHONE, SHIP_ADDRESS1, SHIP_ADDRESS2, SHIP_TOWN,
SHIP_POSTCODE, BILL_ADDRESS1, BILL_ADDRESS2, BILL_TOWN, BILL_POSTCODE,
CARDTYPE, CARDNUM, EXP_DATE, ISS_DATE, ISS_NUM, SECCODE) " +
```

```
“VALUES(InputRoot.XML.Create_Customer_Account_MSG.Personal_Details.Last_Name,“
+

“InputRoot.XML.Create_Customer_Account_MSG.Personal_Details.First_Name, “ +

“InputRoot.XML.Create_Customer_Account_MSG.Personal_Details.User_ID, “ +

“InputRoot.XML.Create_Customer_Account_MSG.Personal_Details.Password, “ +
              “InputRoot.XML.Create_Customer_Account_MSG.Email_Address, “ +
              “InputRoot.XML.Create_Customer_Account_MSG.Daytime_Telephone,“
+

              “InputRoot.XML.Create_Customer_Account_MSG.Evening_Telephone,“
+

“InputRoot.XML.Create_Customer_Account_MSG.Shipping_Address.Address_1, “ +

“InputRoot.XML.Create_Customer_Account_MSG.Shipping_Address.Address_2, “ +

“InputRoot.XML.Create_Customer_Account_MSG.Shipping_Address.Town, “ +

“InputRoot.XML.Create_Customer_Account_MSG.Shipping_Address.Postcode, “ +

“InputRoot.XML.Create_Customer_Account_MSG.Billing_Address.Address_1, “ +

“InputRoot.XML.Create_Customer_Account_MSG.Billing_Address.Address_2, “ +

“InputRoot.XML.Create_Customer_Account_MSG.Billing_Address.Town, “ +

“InputRoot.XML.Create_Customer_Account_MSG.Billing_Address.Postcode, “ +

“InputRoot.XML.Create_Customer_Account_MSG.Payment_Details.Card, “ +

“InputRoot.XML.Create_Customer_Account_MSG.Payment_Details.Card_Number, “ +

“InputRoot.XML.Create_Customer_Account_MSG.Payment_Details.Expiry_Date, “ +

“InputRoot.XML.Create_Customer_Account_MSG.Payment_Details.Issue_Date, “ +

“InputRoot.XML.Create_Customer_Account_MSG.Payment_Details.Issue_Number, “ +

“InputRoot.XML.Create_Customer_Account_MSG.Payment_Details.Security_Code);”);

        state.setThrowExceptionOnDatabaseError(true);
        state.setTreatWarningsAsErrors(true);
        state.select(assembly, newAssembly);

        int sqlCode = state.getSQLCode();
        if (sqlCode != 0) {
```

```
            // Do error handling here
        }
    // End of user code
    // ----------------------------------------------------------

    // The following should only be changed
    // if not propagating message to the 'out' terminal

    out.propagate(assembly);
    }
}
```

11. Save the Java_Create_Customer_Account_JavaCompute.java and Java_Create_Customer_Account.msgflow files.

12. Create the following WebSphere MQ queues in WebSphere MQ Explorer on the WBRK6_DEFAULT_QUEUE_MANAGER queue manager:

   – JAVA_BOOKSTORE_CCA_IN
   – JAVA_BOOKSTORE_CCA_OUT

   Remember to enter the value of the Backout requeue queue property on the JAVA_BOOKSTORE_CCA_IN queue as DLQ.

You have created the Java_Create_Customer_Account message flow, which is one of the Bookstore scenario message flows. When you have created the other message flow, Java_Book_Order, deploy and test them together.

5.3.2 Creating the Java_Book_Order message flow

The input message to the Java_Book_Order message flow contains an order from a customer on the online bookstore Web site. The order contains the customers' identification and details of the books that they have ordered. When the message flow processes the message, it creates a confirmation message that contains details of the order, including a unique order number. Example 5-6 shows the input message for the Java_Book_Order message flow.

Example 5-6   The input message for the Java_Book_Order message flow

```
<Create_Book_Order_MSG>
   <Customer_ID>0123456789</Customer_ID>
   <Order_Date>2002-10-20 12:00:00</Order_Date>
   <Airmail>Yes</Airmail>
   <Book_Details>
      <ISBN>0123456789</ISBN>
      <Book_Price>15.99</Book_Price>
      <ISBN>1425112342</ISBN>
      <Book_Price>7.99</Book_Price>
      <ISBN>9736316345</ISBN>
```

```
            <Book_Price>25.99</Book_Price>
        </Book_Details>
</Create_Book_Order_MSG>
```

Example 5-7 shows the output message that the message flow generates based on the input message shown in Example 5-6 on page 118.

Example 5-7   Book_Order_Response_MSG message for Java_Book_Order message flow

```
<Book_Order_Response_MSG>
    <Customer_ID>0123456789</Customer_ID>
    <OrderNumber>0123456789200110302200012</OrderNumber>
    <Order_Date>2002-10-20 12:00:00</Order_Date>
    <Book_Details>
        <ISBN>0123456789</ISBN>
        <Book_Price>15.99</Book_Price>
        <ISBN>1425112342</ISBN>
        <Book_Price>7.99</Book_Price>
        <ISBN>9736316345</ISBN>
        <Book_Price>25.99</Book_Price>
    </Book_Details>
    <Total_Price>49.97</Total_Price>
    <Order_Status>Order Received</Order_Status>
</Book_Order_Response_MSG>
```

All of the Java that you need for this message flow is available in the Web material described in Appendix B, "Code" on page 319.

Figure 5-10 shows the finished Java_Book_Order message flow.
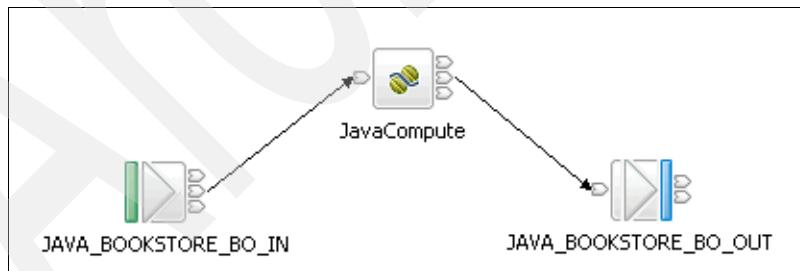
Figure 5-10   The Java_Book_Order message flow

To create the Java_Book_Order message flow:

1. Switch back to the Broker Application Development perspective.

2. Create a message flow called `Java_Book_Order` in the Java_Bookstore Message Flow Project. The Java_Book_Order.msgflow file opens in the Message Flow editor.

3. In the Message Flow editor, add the nodes listed in Table 5-7 to the canvas, then connect the nodes together, as shown in Table 5-8, to build the Java_Book_Order message flow (Figure 5-10 on page 119).

Table 5-7   The Java_Book_Order message flow nodes

Node type	Node name
MQInput	JAVA_BOOKSTORE_BO_IN
JavaCompute	JavaCompute
MQOutput	JAVA_BOOKSTORE_BO_OUT

Table 5-8   Node connections in the Java_Book_Order message flow

Node name	Terminal	Connect to this node
JAVA_BOOKSTORE_BO_IN	Out	JavaCompute
JavaCompute	Out	JAVA_BOOKSTORE_BO_OUT

4. Set the properties of the nodes, as shown in Table 5-9.

Table 5-9   Node properties for the Java_Book_Order message flow

Node name	Page	Property	Value
JAVA_BOOKSTORE_BO_IN	Basic	Queue Name	JAVA_BOOKSTORE_BO_IN
	Default	Message Domain	XML
JavaCompute	Basic	Java Class	com.ibm.itso.wmb6.basics.bookstore.Java_Book_Order_JavaCompute
JAVA_BOOKSTORE_BO_OUT	Basic	Queue Name	JAVA_BOOKSTORE_BO_OUT

5. Create the Java class that is referenced in the JavaCompute node:

   a. In the Message Flow editor, open the Properties dialog for the JavaCompute node. Notice that the Java class that is referenced by the

node is called Java_Book_Order_JavaCompute. To create this Java class in the same Java project as the Java class for the Java_Create_Customer_Account message flow, you must create the Java class manually.

b. Switch to the Java perspective.

c. In the Package Explorer view, right-click **Java_Bookstore Message Flow Project**, then click **New** → **Class**. The New Java Class wizard opens.

d. Make sure that the value in the Source Folder field is Java_Bookstore Message Flow Project.

e. In the Package field, browse for or type com.ibm.itso.wmb6.basics.bookstore.

f. In the Name field, type Java_Book_Order_JavaCompute.

g. Click **Finish**. The new Java_Book_Order.java file opens in the Java editor and is added to the package in the Java_Bookstore Message Flow ProjectJava project.

The Java that is generated automatically in the Java_Book_Order.java file gets the input message, creates a new message based on the input message, and outputs the new message.

6. Edit the Java_Book_Order.java file to add the new fields for the output message, and to make sure that the message flow can build the output message from the decision and repeating elements.

a. Add import statements for the Java utilities that the message flow needs (Example 5-8).

Example 5-8   Adding import statements for the Java utilities

```
import java.math.BigDecimal;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;

import com.ibm.broker.javacompute.MbJavaComputeNode;
import com.ibm.broker.plugin.MbElement;
import com.ibm.broker.plugin.MbException;
import com.ibm.broker.plugin.MbMessage;
import com.ibm.broker.plugin.MbMessageAssembly;
import com.ibm.broker.plugin.MbRecoverableException;
import com.ibm.broker.plugin.MbXML;
```

b. Copy the message headers, as shown in Example 5-9 on page 122, where inMessage is the input message and outMessage is the output message.

Example 5-9   Copying the message headers from input message to output message

```
private void copyMessageHeaders(MbMessage inMessage, MbMessage outMessage)
throws MbException {
      //
      // Retrieve required elements
      //
      MbElement inputHeaderElement =
inMessage.getRootElement().getFirstChild();
      MbElement outputRootElement = outMessage.getRootElement();

      //
      // Copy input message header information to output message root element,
if present
      //
      while((inputHeaderElement != null) &&
(inputHeaderElement.getNextSibling() != null)) {
          outputRootElement.addAsLastChild(inputHeaderElement.copy());
            inputHeaderElement = inputHeaderElement.getNextSibling();
      }
   }
```

c. Calculate the Order_Number value, which is a concatenation of the Customer_ID value and the Order_Date value (Example 5-10). In the code, pCustomerID is the customer identifier and pOrderDate is the order date.

Example 5-10   Calculating the value of Order_Number

```
private static final String ORDER_DATE_FORMAT = "yyyyMMddHHmmss";

private String constructOrderNumber(MbElement pCustomerID, MbElement
pOrderDate) throws MbException {
      String orderNumber = null;

      //
      // Extract values
      //
      String customerIdValue = (String) pCustomerID.getValue();

      //
      // Format Date
      //
      SimpleDateFormat dateFormatter = new
SimpleDateFormat(ORDER_DATE_FORMAT);
      Date date = null;

      try {
          date = dateFormatter.parse((String)pOrderDate.getValue());
```

```
        } catch(ParseException pEx) {
            throw new MbRecoverableException(
                    Java_Book_Order_JavaCompute.class.getName(),
                    "constructOrderNumber()",
                    null,
                    null,
                    "Failed to parse Order_Date to Date object",
                    null);
        }

        String OrderDateString = dateFormatter.format(date);

        //
        // Construct order number
        //
        orderNumber = customerIdValue + OrderDateString;

        return orderNumber;
    }
```

d. Calculate the delivery method (Delivery_Method) and delivery price (Price) (Example 5-11). First class delivery is 18.00, second class delivery is 12.00, and airmail is 08.00. In the code, pDeliveryMethod is the selected delivery method. This uses IF statements to decide which Delivery Method is used based on the content of the message.

Example 5-11   Calculating the delivery method and delivery price

```
/** Delivery Indicator - Yes - Constant */
private static final String DI_YES = "Yes";

/** Delivery Method - First Class - Constant */
private static final String DM_1ST_CLASS = "First_Class";

/** Delivery Method - First Class - Price Constant */
private static final BigDecimal DM_1ST_CLASS_PRICE = new BigDecimal("18.00");

/** Delivery Method - Second Class - Constant */
private static final String DM_2ND_CLASS = "Second_Class";

/** Delivery Method - Second Class - Price Constant */
private static final BigDecimal DM_2ND_CLASS_PRICE = new BigDecimal("12.00");

/** Delivery Method - Airmail - Constant */
private static final String DM_AIRMAIL = "Airmail";

/** Delivery Method - Airmail - Price Constant */
private static final BigDecimal DM_AIRMAIL_PRICE = new BigDecimal("8.00");
```

```java
private BigDecimal determineDeliveryPrice(MbElement pDeliveryMethod) throws
MbException {
        BigDecimal deliveryPrice = null;
        String deliveryMethod = (String) pDeliveryMethod.getName();
        String deliveryIndicator = (String) pDeliveryMethod.getValue();

        //
        // Calculate delivery cost only if indicator is Yes
        //
        if (deliveryIndicator.equals(DI_YES)) {
            if (deliveryMethod.equals(DM_1ST_CLASS)) {
                //
                // Delivery is First_Class (18.00)
                //
                deliveryPrice = DM_1ST_CLASS_PRICE;

            } else if (deliveryMethod.equals(DM_2ND_CLASS)) {
                //
                // Delivery is Second_Class (12.00)
                //
                deliveryPrice = DM_2ND_CLASS_PRICE;

            } else if (deliveryMethod.equals(DM_AIRMAIL)) {
                //
                // Delivery is Airmail (8.00)
                //
                deliveryPrice = DM_AIRMAIL_PRICE;
            }
        }

        return deliveryPrice;
    }
```

e. Calculate the total price of the books (Example 5-12).

Example 5-12   Calculating the total price of the books in the order

```java
/**
    * Calculates Book Total Price
    *
    * @param pBookPriceList the list of book prices to be totalled
    * @return the book total price
    * @throws MbException
    */
   private BigDecimal calculateBookTotalPrice(List pBookPriceTotal) throws
MbException {
        MbElement priceElement = null;
        BigDecimal bookPrice = null;
```

```
        BigDecimal totalPrice = new BigDecimal("0.00");

        //
        // Iterate over all prices and calculate total
        //
        for (int i=0, imax=pBookPriceTotal.size(); i < imax; i++) {
            priceElement = (MbElement) pBookPriceTotal.get(i);
            bookPrice = new BigDecimal((String)priceElement.getValue());
            totalPrice = totalPrice.add(bookPrice);
        }

        return totalPrice;
    }
```

The complete Java class is shown in Example 5-13.

Example 5-13   The complete Java_Book_Order_JavaCompute class

```
/*
 * Created on 8/10/2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package com.ibm.itso.wmb6.basics.bookstore;

import java.math.BigDecimal;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;

import com.ibm.broker.javacompute.MbJavaComputeNode;
import com.ibm.broker.plugin.MbElement;
import com.ibm.broker.plugin.MbException;
import com.ibm.broker.plugin.MbMessage;
import com.ibm.broker.plugin.MbMessageAssembly;
import com.ibm.broker.plugin.MbRecoverableException;
import com.ibm.broker.plugin.MbXML;

/**
 * WebSphere Message Broker Basics Redbook
 *
 * Application Development Chapter, Create Book Order Java Compute Node Example
 *
 * @see com.ibm.broker.javacompute.MbJavaComputeNode
 */
public class Java_Book_Order_JavaCompute extends MbJavaComputeNode {
    /** Delivery Indicator - Yes - Constant */
```

```
        private static final String DI_YES = "Yes";

        /** Delivery Method - First Class - Constant */
        private static final String DM_1ST_CLASS = "First_Class";

        /** Delivery Method - First Class - Price Constant */
        private static final BigDecimal DM_1ST_CLASS_PRICE = new
BigDecimal("18.00");

        /** Delivery Method - Second Class - Constant */
        private static final String DM_2ND_CLASS = "Second_Class";

        /** Delivery Method - Second Class - Price Constant */
        private static final BigDecimal DM_2ND_CLASS_PRICE = new
BigDecimal("12.00");

        /** Delivery Method - Airmail - Constant */
        private static final String DM_AIRMAIL = "Airmail";

        /** Delivery Method - Airmail - Price Constant */
        private static final BigDecimal DM_AIRMAIL_PRICE = new BigDecimal("8.00");

        /** Order Date Format Constant */
        private static final String ORDER_DATE_FORMAT = "yyyyMMddHHmmss";

        /** Order Status Message Constant */
        private static final String ORDER_STATUS_MSG = "Order Received";

    /**
     * Copies Message Headers from Input Message to Output Message
     *
     * @param inMessage the input message
     * @param outMessage the output message
     * @throws MbException
     */
    private void copyMessageHeaders(MbMessage inMessage, MbMessage outMessage)
throws MbException {
        //
        // Retrieve required elements
        //
         MbElement inputHeaderElement =
inMessage.getRootElement().getFirstChild();
        MbElement outputRootElement = outMessage.getRootElement();

        //
        // Copy input message header information to output message root element,
if present
        //
```

```
        while((inputHeaderElement != null) &&
(inputHeaderElement.getNextSibling() != null)) {
            outputRootElement.addAsLastChild(inputHeaderElement.copy());
              inputHeaderElement = inputHeaderElement.getNextSibling();
        }
    }

    /**
     * Constructs the Order Number
     *
     * This is a combination of the Customer_ID and Order_Date
     *
     * @param pCustomerID the customer identifier
     * @param pOrderDate the order date
     * @return the order number
     * @throws MbException
     */
    private String constructOrderNumber(MbElement pCustomerID, MbElement
pOrderDate) throws MbException {
        String orderNumber = null;

        //
        // Extract values
        //
        String customerIdValue = (String) pCustomerID.getValue();

        //
        // Format Date
        //
        SimpleDateFormat dateFormatter = new
SimpleDateFormat(ORDER_DATE_FORMAT);
        Date date = null;

        try {
            date = dateFormatter.parse((String)pOrderDate.getValue());

        } catch(ParseException pEx) {
            throw new MbRecoverableException(
                    Java_Book_Order_JavaCompute.class.getName(),
                    "constructOrderNumber()",
                    null,
                    null,
                    "Failed to parse Order_Date to Date object",
                    null);
        }

        String OrderDateString = dateFormatter.format(date);

        //
```

```
            // Construct order number
            //
            orderNumber = customerIdValue + OrderDateString;

            return orderNumber;
        }

        /**
         * Determines the Delivery Price based on the Delivery Method specified:
         *
         * METHOD PRICE
         * -------------  -------
         * First_Class$18.00
         * Second_Class$12.00
         * Airmail$08.00
         *
         * If NULL is returned, this means an unknown delivery method was specified
         *
         * @param pDeliveryMethod the delivery method selected
         * @return the delivery price
         * @throws MbException
         */
        private BigDecimal determineDeliveryPrice(MbElement pDeliveryMethod) throws
MbException {
            BigDecimal deliveryPrice = null;
            String deliveryMethod = (String) pDeliveryMethod.getName();
            String deliveryIndicator = (String) pDeliveryMethod.getValue();


            //
            // Calculate delivery cost only if indicator is Yes
            //
            if (deliveryIndicator.equals(DI_YES)) {
                if (deliveryMethod.equals(DM_1ST_CLASS)) {
                    //
                    // Delivery is First_Class (18.00)
                    //
                    deliveryPrice = DM_1ST_CLASS_PRICE;

                } else if (deliveryMethod.equals(DM_2ND_CLASS)) {
                    //
                    // Delivery is Second_Class (12.00)
                    //
                    deliveryPrice = DM_2ND_CLASS_PRICE;

                } else if (deliveryMethod.equals(DM_AIRMAIL)) {
                    //
                    // Delivery is Airmail (8.00)
                    //
                    deliveryPrice = DM_AIRMAIL_PRICE;
```

```
            }
        }

        return deliveryPrice;
    }

    /**
     * Calculates Book Total Price
     *
     * @param pBookPriceList the list of book prices to be totalled
     * @return the book total price
     * @throws MbException
     */
    private BigDecimal calculateBookTotalPrice(List pBookPriceTotal) throws
MbException {
        MbElement priceElement = null;
        BigDecimal bookPrice = null;
        BigDecimal totalPrice = new BigDecimal("0.00");

        //
        // Iterate over all prices and calculate total
        //
        for (int i=0, imax=pBookPriceTotal.size(); i < imax; i++) {
            priceElement = (MbElement) pBookPriceTotal.get(i);
            bookPrice = new BigDecimal((String)priceElement.getValue());
            totalPrice = totalPrice.add(bookPrice);
        }

        return totalPrice;
    }

    /*
     * @see
com.ibm.broker.javacompute.MbJavaComputeNode#evaluate(com.ibm.broker.plugin.MbM
essageAssembly)
     */
    public void evaluate(MbMessageAssembly inAssembly) throws MbException {
        //
        // Retrieve input message
        //
        MbMessage inMessage = inAssembly.getMessage();

        //
        // Construct empty output message
        //
        MbMessage outMessage = new MbMessage();
        MbMessageAssembly outAssembly = new MbMessageAssembly(inAssembly,
outMessage);
```

```
        //
        // Build output message
        //
        copyMessageHeaders(inMessage, outMessage);

        MbElement inputRoot = inMessage.getRootElement();
        MbElement inputBody = inputRoot.getLastChild();
        MbElement outputRoot = outMessage.getRootElement();
        MbElement outputBody =
outputRoot.createElementAsLastChild(MbXML.PARSER_NAME);

        // Root element
        MbElement bookOrderResponseMsg =
outputBody.createElementAsLastChild(MbXML.ELEMENT, "Book_Order_Response_MSG",
null);

        // Customer ID
        MbElement inputCustomerIdElement =
inputBody.getFirstElementByPath("./Create_Book_Order_MSG/Customer_ID");
        bookOrderResponseMsg.createElementAsLastChild(MbXML.ELEMENT,
inputCustomerIdElement.getName(), inputCustomerIdElement.getValue());

        // Order Number
        MbElement inputOrderDateElement =
inputBody.getFirstElementByPath("./Create_Book_Order_MSG/Order_Date");
        String orderNumber = constructOrderNumber(inputCustomerIdElement,
inputOrderDateElement);
        bookOrderResponseMsg.createElementAsLastChild(MbXML.ELEMENT,
"OrderNumber", orderNumber);

        // Order Date
        bookOrderResponseMsg.createElementAsLastChild(MbXML.ELEMENT,
inputOrderDateElement.getName(), inputOrderDateElement.getValue());

        // Delivery Method (First_Class/Second_Class/Airmail)
        MbElement inputDeliveryMethodElement =
inputOrderDateElement.getNextSibling();

        if (inputDeliveryMethodElement.getValue().equals(DI_YES)) {
            bookOrderResponseMsg.createElementAsLastChild(MbXML.ELEMENT,
inputDeliveryMethodElement.getName(), inputDeliveryMethodElement.getValue());
        }

        // Book Details
        MbElement inputBookDetails =
inputBody.getFirstElementByPath("./Create_Book_Order_MSG/Book_Details");
        MbElement outputBookDetails =
bookOrderResponseMsg.createElementAsLastChild(MbXML.ELEMENT,
inputBookDetails.getName(), null);
```

```
        outputBookDetails.copyElementTree(inputBookDetails);

        // Delivery Price
        BigDecimal deliveryPrice =
determineDeliveryPrice(inputDeliveryMethodElement);

        if (deliveryPrice != null) {
            bookOrderResponseMsg.createElementAsLastChild(MbXML.ELEMENT,
"Delivery_Price", deliveryPrice);
        }

        // Total Price
        List bookPricesList = (List)
inputBody.evaluateXPath("./Create_Book_Order_MSG/Book_Details/Book_Price");
        bookOrderResponseMsg.createElementAsLastChild(MbXML.ELEMENT,
"Total_Price", calculateBookTotalPrice(bookPricesList));

        // Order Status
        bookOrderResponseMsg.createElementAsLastChild(MbXML.ELEMENT,
"Order_Status", ORDER_STATUS_MSG);

        //
        // Propagate message
        //
        getOutputTerminal("out").propagate(outAssembly);

        //
        // Clear out message
        //
        outMessage.clearMessage();
    }
}
```

7. Save the Java_Book_Order_JavaCompute.java and
   Java_Book_Order.msgflow files.

8. Create the following WebSphere MQ queues in WebSphere MQ Explorer on
   the WBRK6_DEFAULT_QUEUE_MANAGER queue manager:

   – JAVA_BOOKSTORE_BO_IN
   – JAVA_BOOKSTORE_BO_OUT

   Remember to enter the value of the Backout requeue queue property on the
   JAVA_BOOKSTORE_BO_IN queue as DLQ.

You have created the Java_Book_Order message flow. Next, deploy and test
both the Java_Create_Customer_Account message flow and the
Java_Book_Order message flow.

5.3.3 Deploying and testing the Java Bookstore message flows

To test the Java Bookstore message flows, Java_Create_Customer_Account and Java_Book_Order, you must deploy them to the broker.

To deploy the Java Bookstore message flows to the broker:

1. Switch to the Broker Administration perspective.

2. Create a bar file called Java_Bookstore.bar.

3. Add to the bar file the Java_Create_Customer_Account.msgflow file and the Java_Book_Order.msgflow file, then save the bar file. The Java classes are automatically compiled into a single jar file.

4. Create a new execution group on the WBRK6_DEFAULT_BROKER broker called Java_Bookstore.

5. Ensure that the WBRK6_DEFAULT_BROKER broker and the WBRK6_DEFAULT_CONFIGURATION_MANAGER Configuration Manager are running, then deploy the Java_Bookstore.bar file to the Java_Bookstore execution group.

   In the Domains view, the two message flows and the two jar files (compiled Java classes) are displayed under the Java_Bookstore execution group.

6. Create a new enqueue file called Java_Create_Customer_Account.enqueue and use it to put the message in Example 5-2 on page 111 on the JAVA_BOOKSTORE_CCA_IN queue on the WBRK6_DEFAULT_QUEUE_MANAGER queue manager. You can copy the message content from the Web material available to download (see Appendix B, “Code” on page 319).

7. Use the Dequeue wizard to get the output message, which should contain the same message data as the input message, from the JAVA_BOOKSTORE_CCA_OUT queue on the same queue manager.

8. Use the DB2 Control Center to check that the CUSTACCTB table in the BSTOREDB database has been updated with the information from the input message. For instructions about using the DB2 Control Center see 4.3.4, “Deploying and testing the ESQL Bookstore message flows” on page 93.

9. Try changing some of the field values in the input message in the Java_Create_Customer_Account.enqueue file, then put the message through the message flow. Another row is added to the CUSTACCTB table with the values that you entered in the input message.

10. Create a new enqueue file called ESQL_Book_Order.enqueue to put the message in Example 5-6 on page 118. You can copy the message content from the Web material available to download (see Appendix B, “Code” on page 319).

11. Use the Dequeue wizard to get the output message, which should contain the message in Example 5-7 on page 119, from the JAVA_BOOKSTORE_BO_OUT queue on the same queue manager.

12. Try changing the details in the input message in Java_Book_Order.enqueue; for example, add another book to the order or change the price of one of the books. Put the message through the message flow and check the output message to see how your changes affected the output message.

If the message does not output the correct message, or if the message flow cannot process the message, see **Chapter 8, "Troubleshooting and problem determination" on page 241**, for information about problem determination.

5.4 Summary

You have now created, deployed, and tested two message flow applications in which you defined the logic of the message flows using Java.

In the next chapter we create the same message flow applications using a Mapping node instead of a JavaCompute node. We will configure the mapping node using the graphical mapping tools in the Message Brokers Toolkit.

For more information about the built-in nodes that are available in WebSphere Message Broker, see the product documentation: **Developing applications** → **Developing message flow applications** → **Designing a message flow** → **Deciding which nodes to use**.

6

Developing applications with mappings

This chapter describes how to develop message flow applications in the Message Brokers Toolkit using the graphical mapping tools to define the logic of the message flows.

The following topics are discussed:

► Defining the logic of a message flow using mappings
► Message Mapping editor in the Message Brokers Toolkit
► Inserting data into a database using a message flow
► Transforming a message from one XML structure to another

135

6.1 Developing message flow applications with mappings

A message flow application is a program that processes messages in the broker. Message flow applications can transform messages between different formats, generate new messages based on other messages, and route messages according to the message's content or according to how the message flow is configured.

See 4.1.1, "Messages in WebSphere Message Broker" on page 48, for more information about messages.

In Chapter 4, "Developing applications with ESQL" on page 47, you created two message flow applications in which logic of the message flows was defined using ESQL. In Chapter 5, "Developing applications with Java" on page 97, you created the same message flow applications using Java. This chapter describes how to develop and define the logic of the same message flow applications using the graphical mapping tools in the Message Brokers Toolkit to create mappings.

In the ESQL and Java versions of the Simple and Bookstore message flow applications, the XML messages that are used to test the message flows are *self-defining*; that is, all the information about the structure of a message is held within the message itself—in this case, in the form of XML tags. The Compute, Database, and JavaCompute nodes process the XML input message by parsing the XML structure of the message. However, it is possible to define the message's structure externally in a *message set* so that the message flow can refer to this external definition while processing the message.

6.1.1 Message sets and message definitions

A message set is a template that defines the structure of the messages that are processed by a message flow. A message definition is held externally to the message in the message set, and, when the message set is deployed, the definition is compiled into a dictionary. When the message flow is processing a message, the message flow can refer to this dictionary, which is held in the broker.

If the messages do not conform to the structure defined in the message set, the message flow cannot process them. Having an external definition of the message's structure is essential if you want the message flow to validate the message's structure. There are several built-in nodes that can define some of the logic for you if you do predefine messages in a message set. The Mapping node is one of these and enables you to map fields from one message to another without having to write the complex ESQL or Java that you would have to write for a Compute node or JavaCompute node to perform the equivalent function.

Messages have both physical and logical formats. The physical format defines the format or formats that the message flow can process, for example, XML or Tagged Delimited Strings (TDS). You can edit the physical format of messages in the Message Brokers Toolkit using the graphical Message Set editor, which is shown in Figure 6-1.
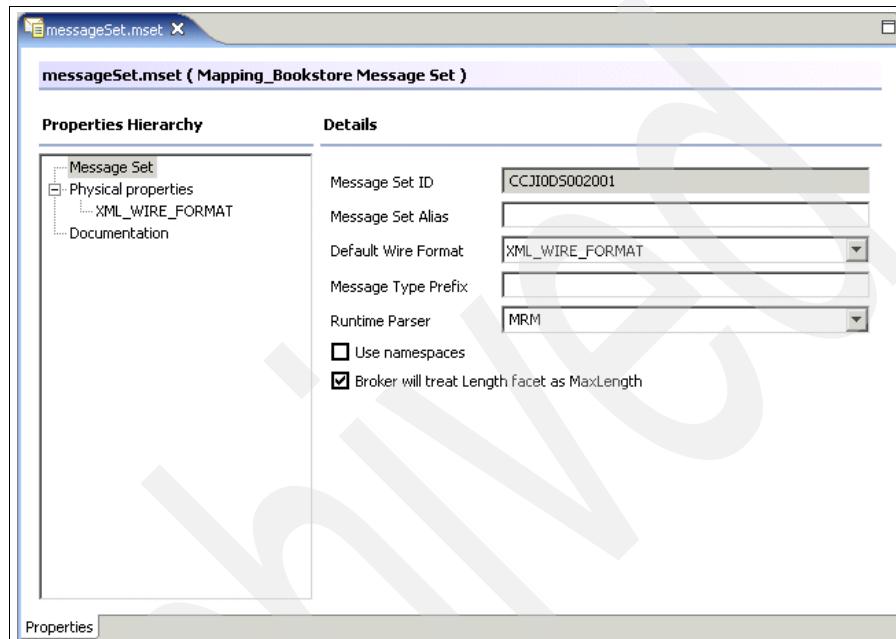
Figure 6-1   The Message Set editor

The logical format defines the organization of the content in the message body—the message structure. You can edit the logical format of messages in the Message Brokers Toolkit using the graphical Message Definition editor, which is shown in Figure 6-2 on page 138.

Figure 6-2   The Message Definition editor

The graphical mapping tools in the Message Brokers Toolkit require you to create external definitions of the message structures so that you can easily drag and drop fields between input and output messages and database table fields.

6.1.2 Mapping and the Message Mapping editor

Mappings are a way of manipulating messages and updating databases using external message definitions and the graphical mapping tools in the Message Brokers Toolkit.

Nodes like the Mapping node or DataInsert node in a message flow enable you to load into the Message Mapping editor (Figure 6-3 on page 139) message structures from the message set, or tables from a database. You can then simply drag and drop elements from one message to another message or database table instead of having to navigate the message structure using ESQL or Java. You can create more complex mappings by editing the mappings using XPath.
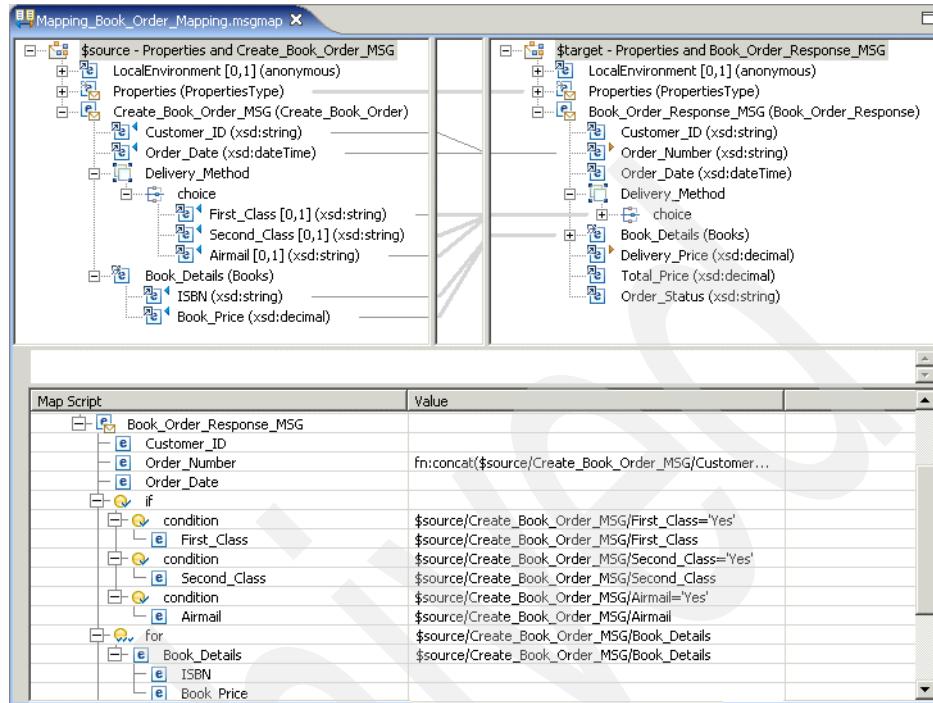
Figure 6-3   The Message Mapping editor

6.1.3 Scenarios described in this chapter

This chapter focuses on how to define the logic of message flows with mappings. We provide step-by-step instructions to create, deploy, and test two message flow applications:

► Simple message flow application

The Simple message flow application demonstrates how to build a very basic message flow from three nodes. The Mapping_Simple message flow takes an XML input message from a WebSphere MQ queue, uses mappings in a Mapping node to build an XML output message that has the same contents as the input message, then puts the output message on another WebSphere MQ queue.

► Bookstore message flow application

The Bookstore message flow application is based around the scenario of an online bookstore. The first message flow, Mapping_Create_Customer_Account, uses mappings in a DataInsert node to create accounts in a DB2 database table for new customers who have

registered their details with the bookstore, for example, their contact details and delivery address. The second message flow, Mapping_Book_Order, uses mappings in a Mapping node to process an order that has been submitted by an online customer and create a response message to confirm the order with a unique order number.

You do not need skills or experience in creating mappings or in coding XPath to be able to create the message flow applications in this chapter because all the code is provided in the Web material described in Appendix B, "Code" on page 319.

6.1.4 Before you start

The instructions in this chapter assume that you have run the Default Configuration wizard to create the default configuration. However, you can create your own broker domain and substitute the component names when following the instructions.

For more information about the Default Configuration wizard see 3.5, "Verifying the installation" on page 35. For more information about administering components see "Starting the components" on page 213.

Ensure that the broker and the Configuration Manager are running.

Starting the broker and the Configuration Manager

You cannot start components from the Message Brokers Toolkit; you must start them from the command line. Enter all commands in a WebSphere Message Broker Command Console, which is a command window with additional WebSphere Message Broker Environment settings.

To start the Command Console, click **Start** → **Programs** → **IBM WebSphere Message Brokers 6.0** → **Command Console**.

To start the broker, enter the following command in the Command Console, where WBRK6_DEFAULT_BROKER is the name of the broker in the default configuration:

```
mqsistart WBRK6_DEFAULT_BROKER
```

To start the Configuration Manager, enter the following command in the Command Console, where WBRK6_DEFAULT_CONFIGURATION_MANAGER is the name of the Configuration Manager in the default configuration:

```
mqsistart WBRK6_DEFAULT_CONFIGURATION_MANAGER
```

Open the Windows Event Viewer to check that the components have started without any problems. See 8.1.5, "Windows Event Viewer" on page 253, for information about how to access and view entries in the Windows Event Viewer.

This chapter also assumes that you have already completed the exercises in Chapter 4, "Developing applications with ESQL" on page 47, and Chapter 5, "Developing applications with Java" on page 97. The exercises in this chapter do not depend on those in Chapter 4 and Chapter 5, but less detail is given in the instructions in this chapter. Refer to the step-by-step instructions in Chapter 4 if you need more details when creating, deploying, and testing the Java versions of the message flow applications.

6.2 Developing the Simple message flow application

The Simple message flow application includes a message set with one message definition and a message flow.

Each message definition file has the extension .mxsd. You can define more than one message in the same message definition file, though you should do this only if the messages are related, for example, if they share the same elements like the input and output messages for the Mapping_Book_Order message flow. All the message definition files are stored in a message set project, which also contains a properties file for the message set (.msgset) that contains the message definitions.

Each message flow is stored in a message flow file with the extension .msgflow. The message flow file is, in turn, stored in a Message Flow project, along with any associated message map files (.msgmap).

When you have created the message set and message flow, deploy the message flow and message set to the broker so that you can test the application.

6.2.1 Defining the message model

The message model is the structure of the message that is defined in the message definition file. You can create one or more message definition files to represent different formats of the message model, such as XML, Tagged Delimited String (TDS), or Custom Wire Format (CWF). For more information about the different message formats that are supported in WebSphere Message Broker, see the product documentation: **Developing applications** → **Developing message models** → **Message modelling overview** → **Physical formats in the MRM domain**.

Like the ESQL_Simple message flow and the Java_Simple message flow, the content of the output message from the Mapping_Simple message flow is copied from the input message. The Mapping_Simple message set contains one message definition for XML messages because both the input message and the output message for the Mapping_Simple message flow are in XML format.

Creating the message set

Before you can start modeling the message structure, create the message set that holds the message definition:

1. In the Broker Application Development perspective, click **File** → **New** → **Message Set Project**. The New Message Set Project wizard opens.

2. In the Project name field, type `Mapping_Simple Message Set Project`, then click **Next**.

3. In the Message Set Name field, type `Mapping_Simple_Message_Set`, then click **Next**.

4. Select the **XML Wire Format Name** check box, then type `XML1` as the name of the XML wire format.

5. Click **Finish**.

A new project called Mapping_Simple Message Set Project is displayed in the Resource Navigator view. A message set called Mapping_Simple message set is displayed in Mapping_Simple Message Set Project. The messageSet.msgset file, in which you configure the message set, opens automatically in the Message Set editor.

> **Note:** If, when you create a new message set project, two folders are created in the message set project to represent the message set, click and reopen the project so that the second folder disappears.

Configuring the message set

In case the Mapping_Simple message flow cannot deduce the format of the message (in this case, XML) from the message header, the message flow uses the default setting that is defined in the message set. This default format is set in the message set properties file, messageSet.msgset.

To configure the default format of the message set:

1. Open messageSet.msgset in the Message Set editor.

2. In the Properties Hierarchy, click **Message Set**.

3. From the Default Wire Format menu, click **XML1**, then save the file.

For more information about physical message formats, see the product documentation at **Developing applications → Developing message models → Message modelling overview → Physical formats in the MRM domain**. For more information about the logical message model, see **Developing applications → Developing message models → Message modelling overview → The message model**.

For a useful example of how to model the same message in different physical formats and transform the message from one physical format to another, explore the Video Rental sample in the Message Brokers Toolkit Samples Gallery.

Defining the logical structure of the message

When you created the Mapping_Simple message set, you defined the physical format of the message as XML. Now define the logical structure of the content within the message.

The message for the Mapping_Simple message flow (see Example 6-1) is constructed from just two elements, as shown in Figure 6-4.

Example 6-1   The message for the Mapping_Simple message flow

```
<Message>
    <Body>
        Hello, World!
    </Body>
</Message>
```
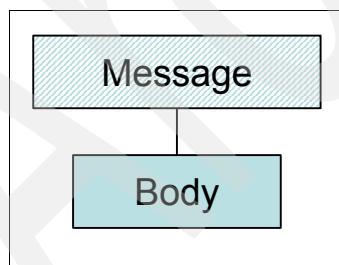

Figure 6-4   Logical structure of message for Mapping_Simple message flow

In Figure 6-4, the Body element represents the Body field in the message in Example 6-1. In the message, the Body field contains the string `Hello, World!`. Each element in a message is based on a data type, and elements such as the Body element are based on simple types, such as *string*, *integer*, and so on. The data in the Body field in the message has the *string* data type. In diagrams in this

book, elements that are based on simple data types are shown in solid blocks of color, like the Body element in Figure 6-4 on page 143.

Elements that contain other elements, such as the Message element, are not based on simple types. Instead, they are based on complex types. Complex types define the structure of a message and the relationship between the elements and other parts of the message. In diagrams in this book, elements that are based on complex types are shown as shaded rectangles, like the Message element in Figure 6-4 on page 143. The Message element is based on the MessageType complex type.

To define the logical structure of the message for the Mapping_Simple message flow:

1. Create the Mapping_Simple message definition file in Mapping_Simple Message Set Project:

   a. In the Resource Navigator view, click **Mapping_Simple Message Set Project** to highlight it.

   b. Click **File** → **New** → **Message Definition File**. The New Message Definition File wizard opens.

   c. Click **Create a new message definition file**, then click **Next**.

   d. Click **Mapping_Simple Message Set** to highlight it, then in the File Name field, type `Mapping_Simple`.

   e. Click **Finish**.

   A new message definition file called Mapping_Simple.mxsd is displayed in Mapping_Simple message set in the Resource Navigator view. The Mapping_Simple.mxsd file opens automatically in the Message Definition editor. The Resource Navigator view should look similar to Figure 6-5.
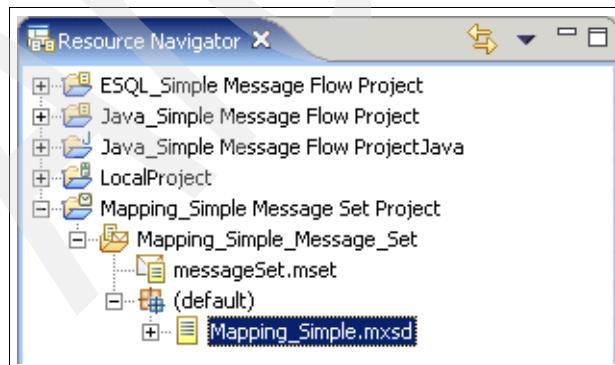
Figure 6-5   The Mapping_Simple message set resources

Mapping_Simple.mxsd is displayed in the default namespace of the Mapping_Simple Message Set Project. For more information about namespaces, see the product documentation: **Developing applications → Message modelling overview → The message model → Namespaces**.

2. Add the main message element, which contains the rest of the message:

   a. In Mapping_Simple.mxsd, right-click **Messages**, then click **Add Message** (Figure 6-6). A message element, Message1, is added. Ensure that the cursor is in the Message1 cell and rename Message1 to Message, as shown in Figure 6-7 on page 146; then press Enter.
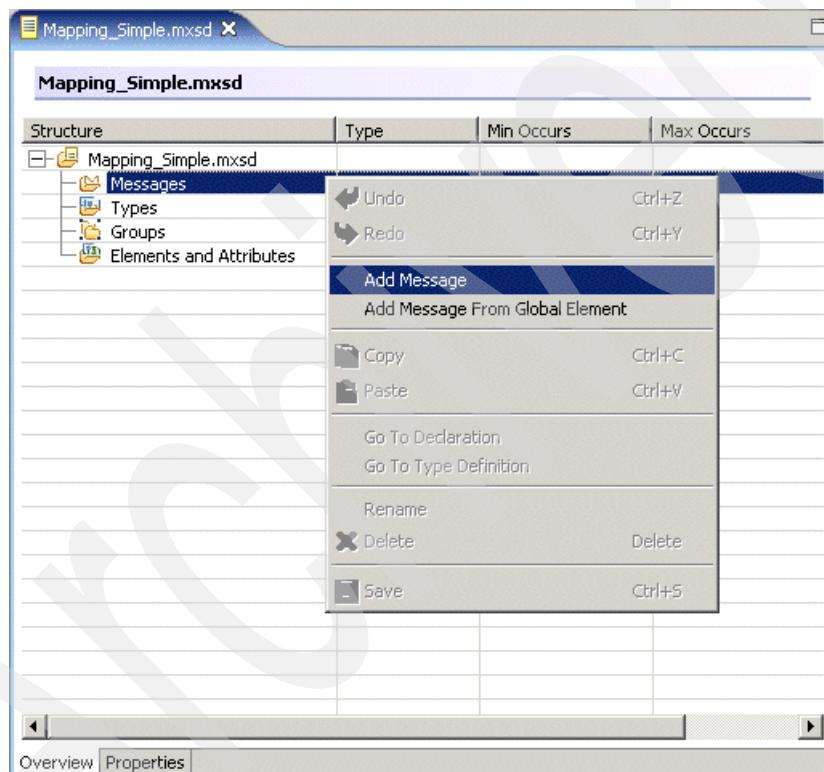
Figure 6-6   Adding message element to Mapping_Simple message definition

Figure 6-7   Renaming the message element

    b. Expand **Types** to display the complex type, complexType1, which was created automatically for the Message element.

    c. Click **complexType1** to highlight it, then click it again to place the cursor in the cell.

    d. Rename complexType1 to `MessageType` then press Enter (Figure 6-8).

Figure 6-8   Renaming complexType1 to MessageType

3. Add the Body element:

    a. Right-click **Elements and Attributes**, then click **Add Global Element** (Figure 6-9 on page 147).

    Elements and Attributes expands to display the Message element and a new element called globalElement1.
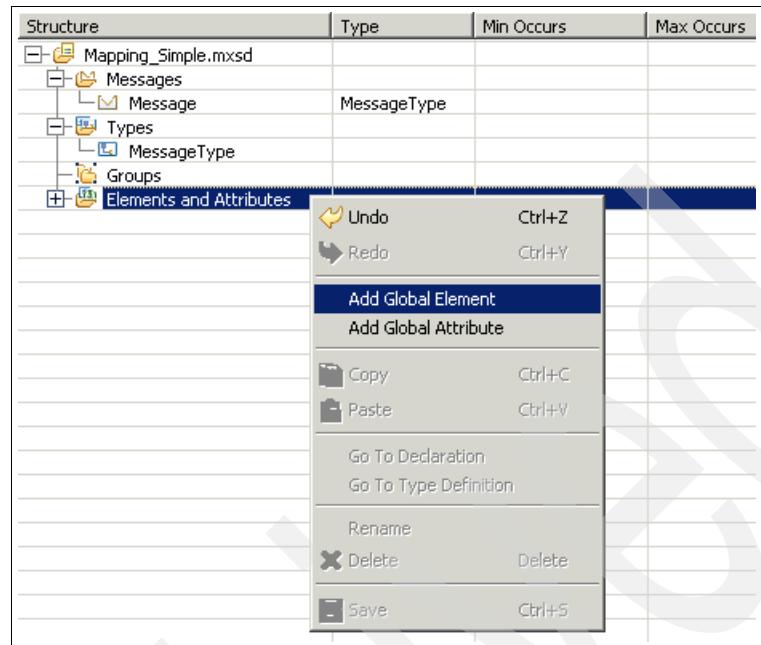
Figure 6-9   Adding a new element to the Mapping_Simple message definition

   b. Rename globalElement1 to Body (Figure 6-10).
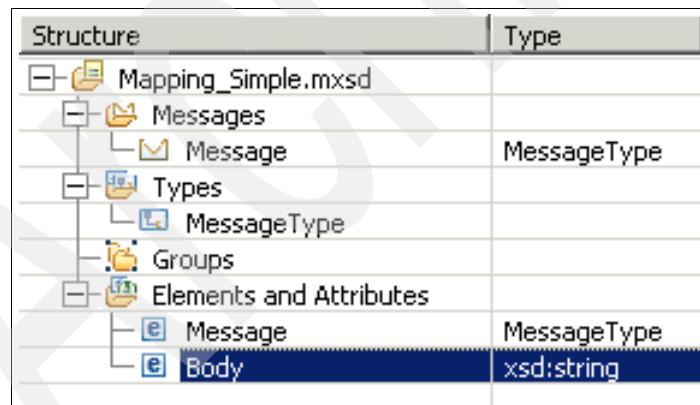
Figure 6-10   Renaming globalElement1 to Body

4. Add a reference from the Message element to the Body element to define the order of the elements in the message:

   a. In the Messages section, right-click **Message**, then click **Add Element Reference** (Figure 6-11 on page 148).
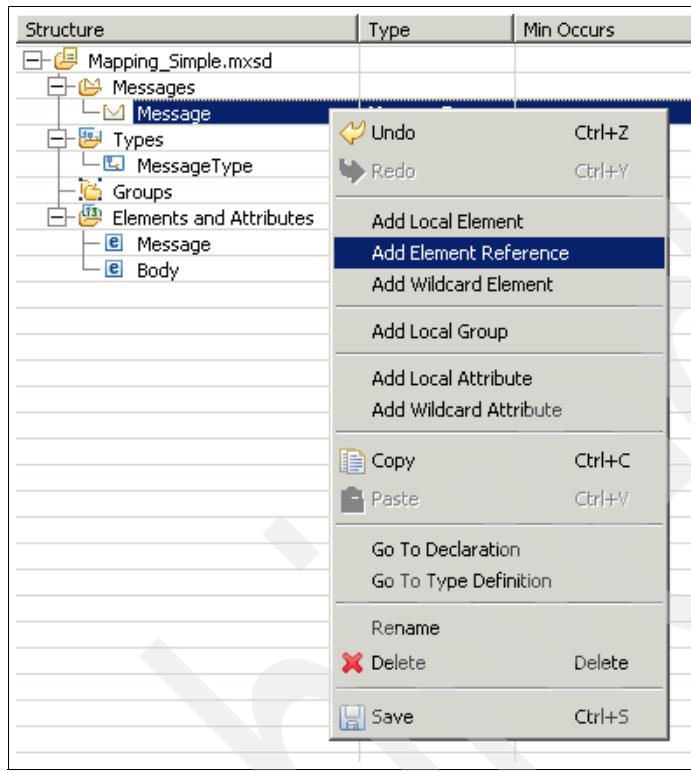
Figure 6-11   Adding a reference from the Message element to the Body element

b. Click an empty cell to accept the first option, Body, in the list (Figure 6-12 on page 149).

The Body element is displayed under the Message element to show the hierarchical arrangement of the elements in the message. In this case, there are only two elements so the hierarchy is very simple.

Figure 6-12   The complete Mapping_Simple message definition

> The value 1 in each of the Min Occurs and Max Occurs columns shows that the Body element, or field, must be present in the message once. That is, you cannot have multiple instances of the Body field in the message but you cannot have zero instances of the Body field either.

5. Save the Mapping_Simple.mxsd file.

You have now created the message set for the Mapping_Simple message flow application.

The next section describes how to create the Mapping_Simple message flow to process messages that are structured as defined in the Mapping_Simple message definition.

6.2.2 Creating the Mapping_Simple message flow

In the ESQL_Simple and Java_Simple message flows, the Compute and JavaCompute nodes are capable of modifying the message and generating a new message. However, all information about the message structure is held in the message itself (the XML message is self-defining), so the logic in the node must be defined, using ESQL or Java, to perform the processing on the message. The more complex the processing, the more complex the ESQL or Java must be, which makes the message flow more complex to maintain and debug.

If the logical structure of the XML message is externally defined in a message set, you do not have to write such complex ESQL or Java to manipulate the message. The message flow refers, instead, to the external message set for information about each message that it processes. If you create a message set, you can use a Mapping node instead of the Compute or JavaCompute node.

The Mapping node enables you to load into the Message Mapping editor the input message and output message structures, which in this case are the same, from the message set. You can then simply drag and drop elements from the input message onto elements in the output message to show which data should be used from the input message to create the output message.

Mapping fields from one message to another is useful if, for example, the business application that sends the message to the broker structures the data in the message differently from the business application that will receive the message. If the first business application positions the customer's first name before their last name but the second business application expects the customer's last name to precede their first name in the message, the Mapping node can map the fields so that each business application can understand the message.

To create the Mapping_Simple message flow, first create the files in which the message flow is stored (see 4.2.1, "Creating the ESQL_Simple message flow" on page 53):

1. In the Broker Application Development perspective, create a Message Flow project called `Mapping_Simple Message Flow Project.`

2. Create a message flow called `Mapping_Simple` in Mapping_Simple Message Flow Project.

The Mapping_Simple.msgflow file is displayed in Mapping_Simple Message Flow Project in the Resource Navigator view. The Mapping_Simple.msgflow file opens automatically in the Message Flow editor.

Adding and connecting the Mapping_Simple nodes

Figure 6-13 shows the finished Mapping_Simple message flow.
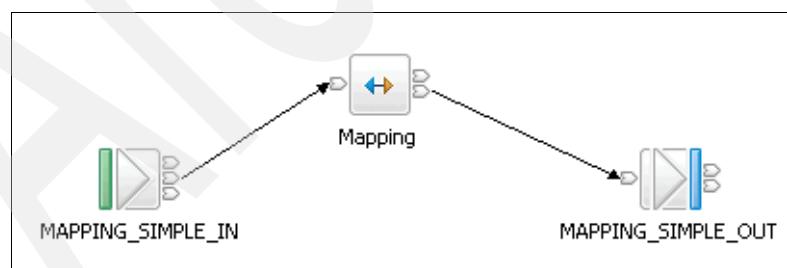
Figure 6-13   The Mapping_Simple message flow

To create the Mapping_Simple message flow:

1. In the Message Flow editor, add the nodes listed in Table 6-1 on page 151 to the canvas.

2. Connect the nodes together, as shown in Table 6-2.

Table 6-1   The Mapping_Simple message flow nodes

Node type	Node name
MQInput	MAPPING_SIMPLE_IN
Mapping	Mapping
MQOutput	MAPPING_SIMPLE_OUT

Table 6-2   Node connections in the Mapping_Simple message flow

Node name	Terminal	Connect to this node
MAPPING_SIMPLE_IN	Out	Mapping
Mapping	Out	MAPPING_SIMPLE_OUT

3. Save the Mapping_Simple.msgflow file.

For detailed instructions on creating a message flow, see 4.2.1, "Creating the ESQL_Simple message flow" on page 53.

6.2.3 Configuring the Mapping_Simple message flow

To configure the Mapping_Simple message flow:

1. In WebSphere MQ Explorer, create the following queues on the WBRK6_DEFAULT_QUEUE_MANAGER queue manager:

   – MAPPING_SIMPLE_IN
   – MAPPING_SIMPLE_OUT

2. Set the Backout requeue queue of the MAPPING_SIMPLE_IN queue to DLQ.

3. Set the properties of the Mapping_Simple message flow nodes as shown in Table 6-3 on page 152.

   Notice that because the message flow is using a message set, you must configure the MQInput node, MAPPING_SIMPLE_IN, with information about the message set and how to parse the messages (Figure 6-14 on page 152). Select the details from the drop-down lists in the Properties dialog; the unique message set identifier that is shown in the Message Set field is different from the identifier shown in Figure 6-14 on page 152.

Table 6-3   Node properties for the Mapping_Simple message flow

Node name	Page	Property	Value
MAPPING_SIMPLE_IN	Basic	Queue Name	MAPPING_SIMPLE_IN
	Default	Message Domain	MRM
		Message Set	Mapping_Simple Message Set
		Message Type	Message
		Message Format	XML1
MAPPING_SIMPLE_OUT	Basic	Queue Name	MAPPING_SIMPLE_OUT
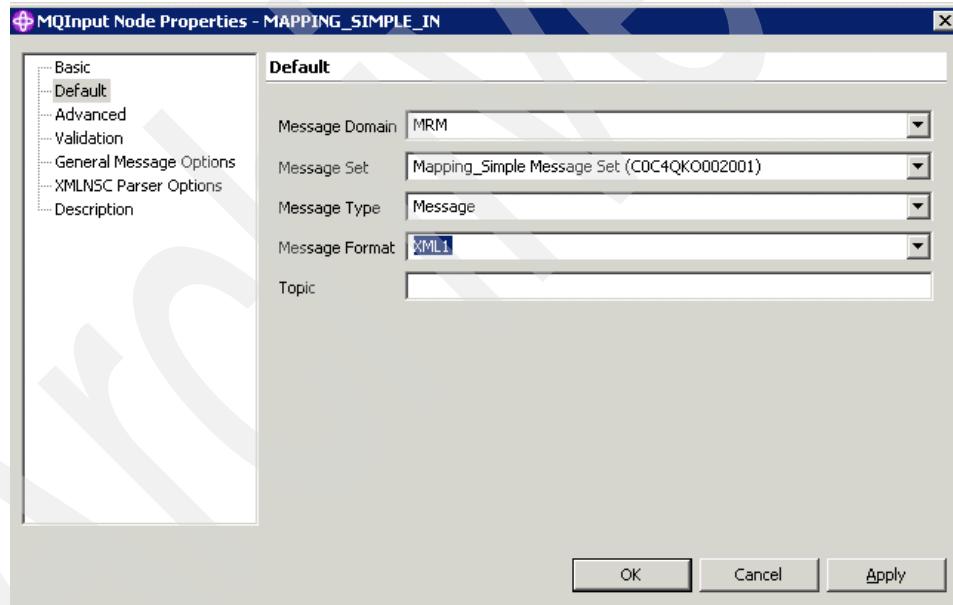
Figure 6-14   Configuring MQInput node with information about message set

For detailed instructions on configuring a message flow, see 4.2.2, "Configuring the ESQL_Simple message flow" on page 58.

6.2.4 Creating the mappings for the Mapping_Simple message flow

The Mapping node enables you to drag and drop fields from the input message to fields of the output message, to map the content of the input message to the output message.

To create the message map file:

1. In Mapping_Simple.msgflow, right-click the **Mapping** node, then click **Open Map**. The New Message Map wizard opens.

2. Accept the values on the first page of the wizard, then click **Next** (Figure 6-15).
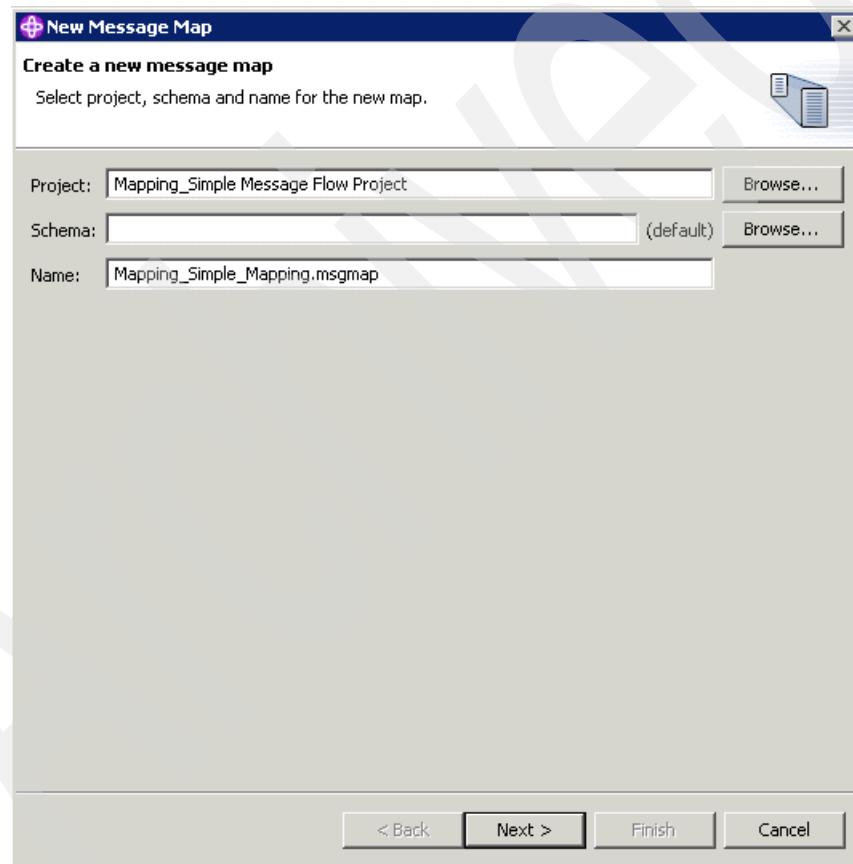
Figure 6-15   Naming the new message map for the Mapping_Simple message flow

3. Make sure that **This map is called from the message flow and maps properties and message body** is selected, then click **Next** (Figure 6-16).
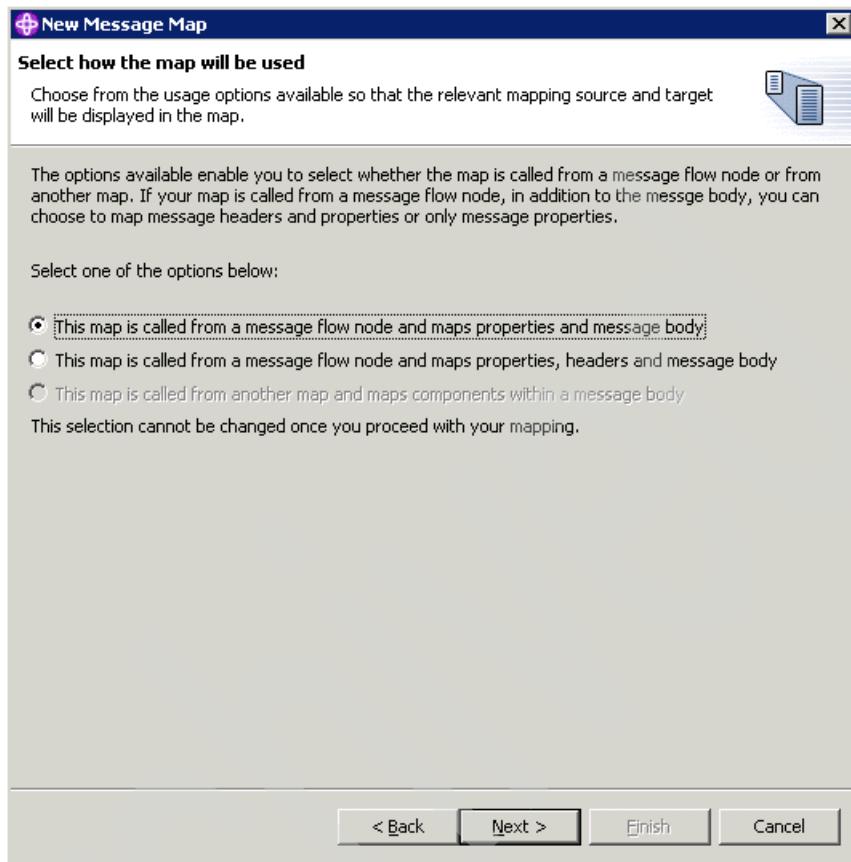
Figure 6-16   Selecting how the message map will be used

4. Clear the database records check box so that only the **input message** check box is selected, then click **Next** (Figure 6-17 on page 155).
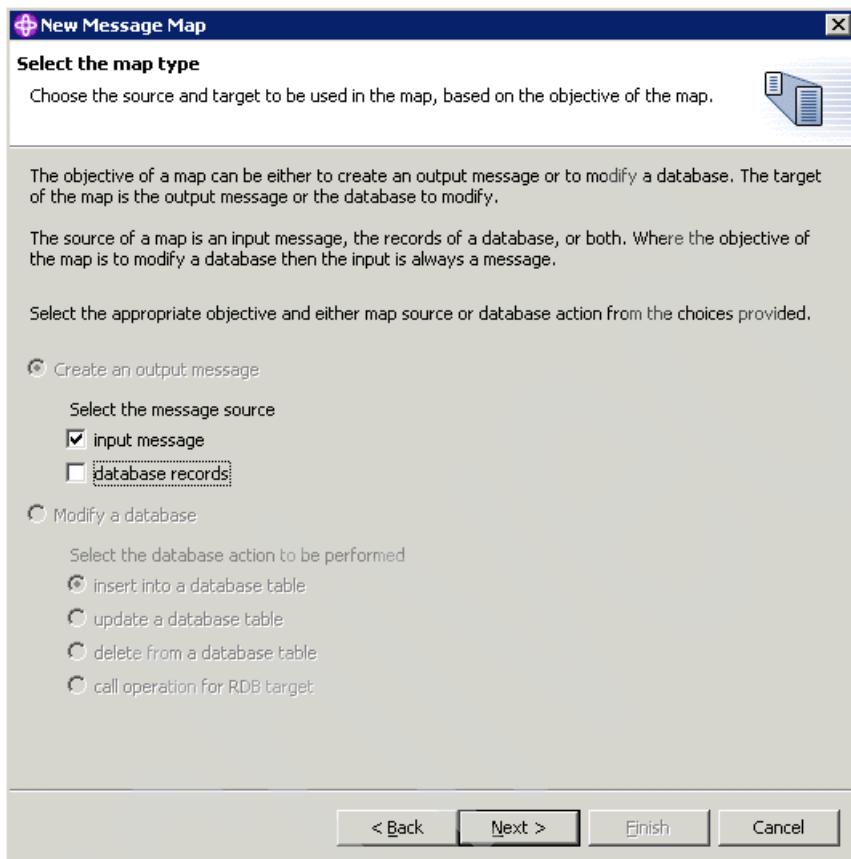
Figure 6-17   Select message flow to create output message from input message

5. On the Source and Target Mappables page, in both the Source and Target panes, expand the **Mapping_Simple Message Set** then click **Message** to highlight it (Figure 6-18 on page 156).
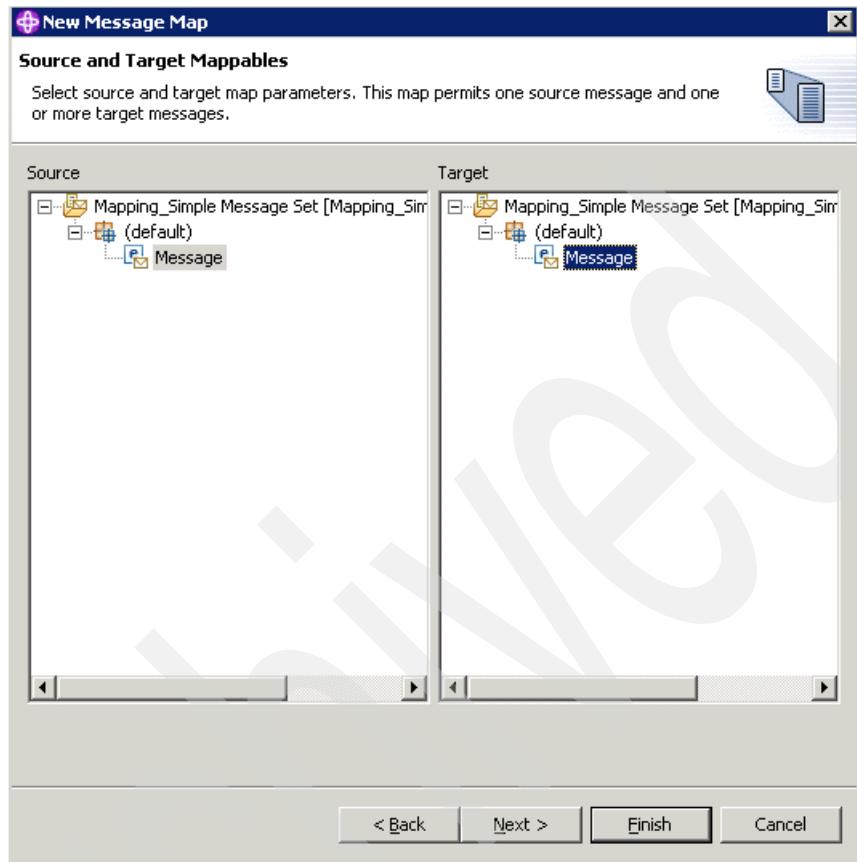
Figure 6-18   Select source and target message definitions

6. Click **Finish**.

    The Mapping_Simple_Mapping.msgmap file is displayed in the
    Mapping_Simple Message Flow Project in the Resource Navigator view, and
    is automatically opened in the Message Mapping editor.

    Now that you have created the message map file, create the mappings.

7. In the Message Mapping editor, expand **$source - Properties and Message**
    and the **$target - Properties and Message**, as shown in Figure 6-19 on
    page 157.
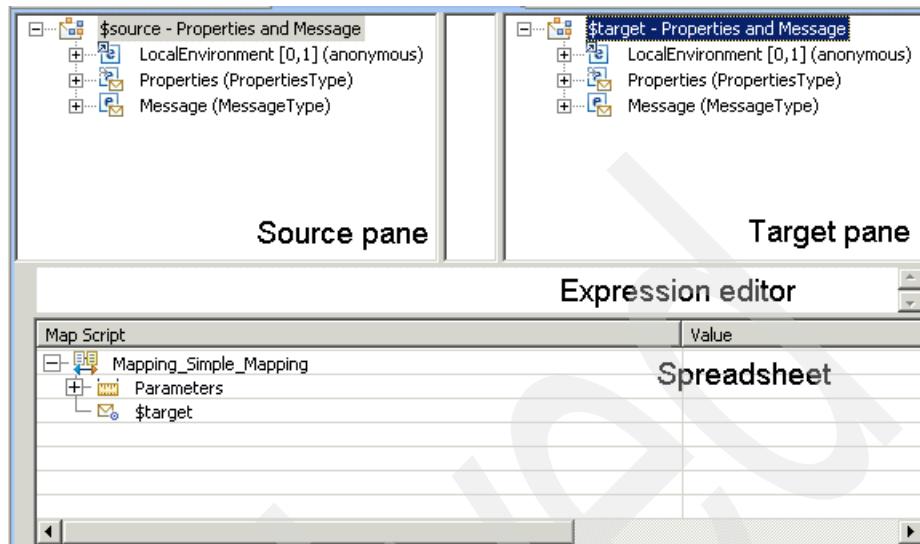
Figure 6-19   The Message Mapping editor

8. In the Source pane (see Figure 6-19 for the parts of the Message Mapping
    editor), drag and drop **Properties** on to Properties in the Target pane
    (Figure 6-20).

    The properties of the message include the information that is held in the
    message's header. The Mapping_Simple message flow will create an output
    message with the same header information as the input message.
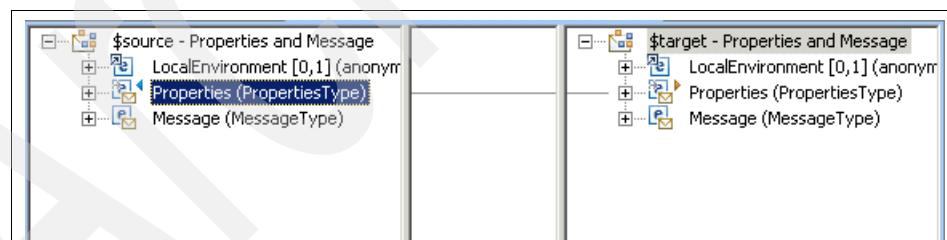
Figure 6-20   Mapping input message properties to output message properties

9. In both the Source and the Target panes, expand **Message (MessageType)**
    to display the message's structure.
10. Drag and drop **Message** in the Source pane to Message in the Target pane
    (Figure 6-21 on page 158).
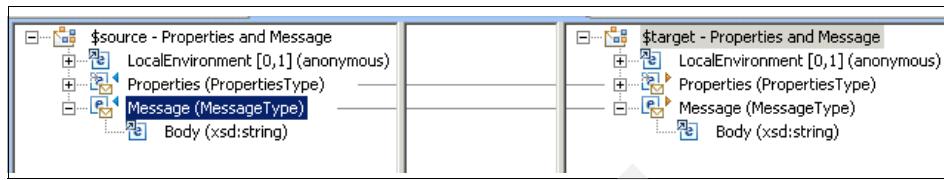
Figure 6-21   Mapping the input message body to the output message body

11.Save the Mapping_Simple_Mapping.msgmap file.

You have completed the mappings for the Mapping_Simple message flow.

6.2.5 Deploying and testing the Mapping_Simple message flow

To test the Mapping_Simple message flow, you must deploy both the Mapping_Simple message flow and the Mapping Simple message set to the broker.

To deploy the Mapping_Simple message flow resources to the broker:

1. Switch to the Broker Administration perspective.

2. Create a bar file called Mapping_Simple.bar.

3. Add to the bar file the following resources (as shown in Figure 6-22 on page 159):

   – Mapping_Simple.msgflow file from Mapping_Simple Message Flow Project

   – Mapping_Simple message set

   When the files have been added, the Mapping_Simple.bar file contains two files:

   – Mapping_Simple.cmf (the compiled message flow)

   – Mapping_Simple Message Set.dictionary (the Dictionary file, which is created from the message set)

4. Save the Mapping_Simple.bar file.
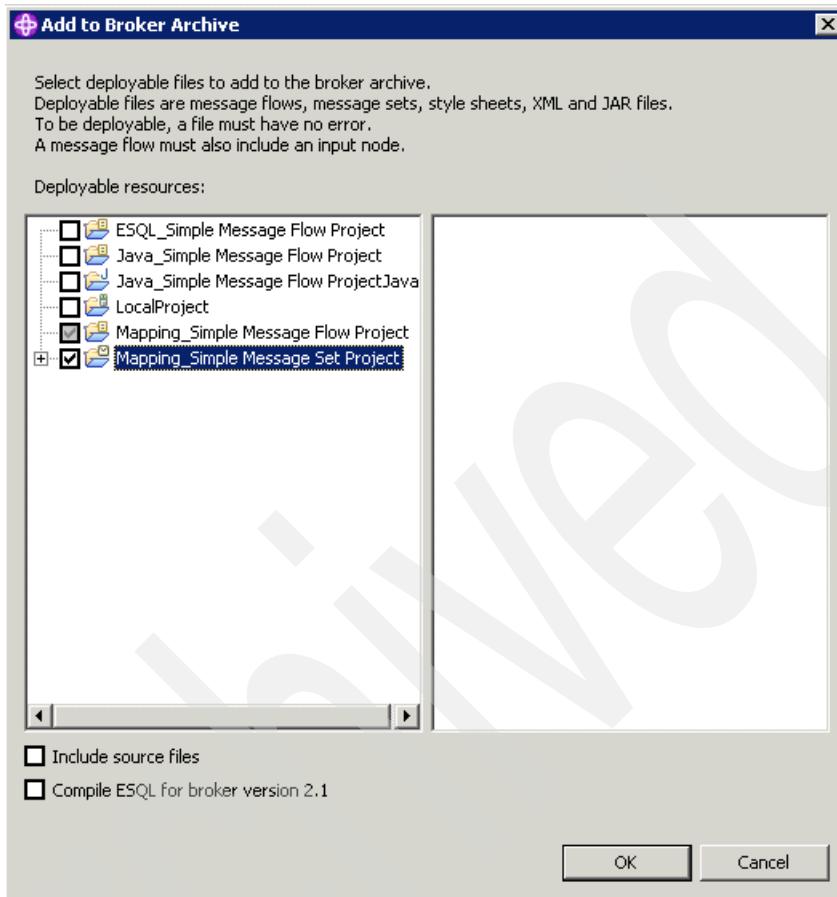
Figure 6-22 Adding the Mapping_Simple message flow resources to the bar file

5. Create a new execution group on the WBRK6_DEFAULT_BROKER broker called Mapping_Simple.

6. Ensure that the WBRK6_DEFAULT_BROKER broker and the WBRK6_DEFAULT_CONFIGURATION_MANAGER Configuration Manager are running, then deploy the Mapping_Simple.bar file to the Mapping_Simple execution group.

   When the files have deployed to the broker, the Mapping_Simple message flow and the Mapping_Simple message set are displayed in the Domains view.

7. Create a new enqueue file called Mapping_Simple.enqueue in Mapping_Simple Message Flows Project.

8. Edit the Mapping_Simple.enqueue file so that it connects to the WBRK6_DEFAULT_QUEUE_MANAGER and puts the message in Example 6-1 on page 143 on the MAPPING_SIMPLE_IN queue. You can copy the message content from the Web material available to download (see Appendix B, "Code" on page 319).

9. In the Message data field, type the XML message content shown in Example 6-1 on page 143. The input message for the Mapping_Simple message flow is the same as for the Java_Simple and ESQL_Simple message flows.

10. Save the Mapping_Simple.enqueue file; then in the Enqueue editor, click **Write To queue**. The message is put on the MAPPING_SIMPLE_IN queue.

11. Use the Dequeue wizard to get the output message, which should contain the same message data as the input message, from the MAPPING_SIMPLE_OUT queue on the same queue manager. Notice that the message flow has inserted an XML declaration statement at the beginning of the output message so that the XML in the message is well formed (Example 6-2).

Example 6-2   The output message from the Mapping_Simple message flow

```
<?xml version="1.0"?>
<Message>
    <Body>
        Hello, World!
    </Body>
</Message>
```

For detailed instructions on deploying and testing a message flow, see 4.2.4, "Deploying and testing the ESQL_Simple message flow" on page 67, and 4.2.5, "Diagnosing problems with the ESQL_Simple message flow" on page 78.

6.3 Developing the Bookstore scenario with mappings

In 6.2, "Developing the Simple message flow application" on page 141, you created the Simple scenario message flow application using mappings to define the logic of the message flow.

In this section we create a more complex message flow application that is based around the scenario of an online bookstore. The Bookstore scenario message flows process messages with different structures, and interact with databases to update database tables.

The Bookstore scenario includes two message flows:

- ► The Mapping_Create_Customer_Account message flow

  This message flow uses mappings in a DataInsert node to create accounts in a DB2 database table for new customers who have registered their details with the bookstore, for example, their contact details and delivery address.

- ► The Mapping_Book_Order message flow

  This message flow uses mappings in a Mapping node to process an order that has been submitted by an online customer and create a response message to confirm the order with a unique order number.

The message flow applications in this chapter use the same DB2 database that you created for the ESQL and Java Bookstore message flow applications. You do not need to re-create the database for this chapter. For more information about the database, see 4.3.1, "Creating the Bookstore scenario database" on page 80.

6.3.1 Defining the message model

The message model is the structure of the message that is defined in the message definition file. You can create one or more message definition files to represent different formats of the message model, such as XML, Tagged Delimited String (TDS), or Custom Wire Format (CWF). For more information about the different message formats that are supported in WebSphere Business Integration Message Broker, see the product documentation: **Developing applications → Developing message models → Message modelling overview → Physical formats in the MRM domain**.

The Bookstore message flow application includes three message definitions:

- ► The Create_Customer_Account_MSG message structure is for the input message for the Mapping_Create_Customer_Account message flow.

- ► The Create_Book_Order_MSG message structure is for the input message for the Mapping_Book_Order message flow.

- ► The Book_Order_Response_MSG message structure is for the output message from the Mapping_Book_Order message flow.

The definition of the Create_Customer_Account_MSG message structure is stored in the one message definition file (Create_Customer_Account.mxsd), while the definitions of the Create_Book_Order_MSG and Book_Order_Response_MSG message structures are stored together in another message definition file (Book_Order.mxsd) because they share several elements.

Creating and configuring the message set

To define the Mapping Bookstore messages, first create the message set project files:

1. Create a message set project:

   a. Click **File** → **New** → **Message Set Project**. The New Message Set Project wizard opens.

   b. In the Project name field, type `Mapping_Bookstore Message Set Project`, then click **Next**.

   c. In the Message Set Name field, type `Bookstore Message Set`, then click **Next**.

   d. Select the **XML Wire Format Name** check box, then type `XML_WIRE_FORMAT` as the name of the XML wire format.

   e. Click **Finish**.

   A new project called Mapping_Bookstore Message Set Project is displayed in the Resource Navigator view. A message set called Mapping_Bookstore Message Set is displayed in Mapping_Bookstore Message Set Project. The messageSet.msgset file (in which you configure the message set) opens automatically in the Message Set editor.

2. Configure Mapping_Bookstore message set:

   a. In the Message Set editor, in the Properties Hierarchy, click **Message Set**.

   b. From the Default Wire Format list, select **XML_WIRE_FORMAT**, which is the wire format that you created when you created the message set.

   c. Save the messageSet.msgset file, then close it.

The Mapping_Bookstore message set is now configured to use the XML_WIRE_FORMAT format by default.

If you need more detailed instructions when creating the message set and message definitions, see 6.2.1, "Defining the message model" on page 141.

Defining the Create_Customer_Account message

An example message for the Mapping_Bookstore message flow is shown in Example 6-3. Figure 6-23 on page 164 shows how the logical structure of the message is constructed from a number of elements arranged hierarchically.

Example 6-3   The message for the Mapping_Create_Customer_Account message flow

```
<Create_Customer_Account_MSG>
   <Personal_Details>
      <First_Name>Peter</First_Name>
      <Last_Name>Smith</Last_Name>
```

```
            <User_ID>PSmith</User_ID>
            <Password>p45sw0rd</Password>
        </Personal_Details>
        <Email_Address>Peter.Smith@nowhere.com</Email_Address>
        <Daytime_Telephone>1234567890</Daytime_Telephone>
        <Evening_Telephone>1234567890</Evening_Telephone>
        <Shipping_Address>
            <Address_1>19 Green Street</Address_1>
            <Address_2>Littleton></Address_2>
            <Town>Southington</Town>
            <Postcode>SU29 8YT</Postcode>
        </Shipping_Address>
        <Billing_Address>
            <Address_1>19 Green Street</Address_1>
            <Address_2>Littleton></Address_2>
            <Town>Southington</Town>
            <Postcode>SU29 8YT</Postcode>
        </Billing_Address>
        <Payment_Details>
            <Card>VISA</Card>
            <Card_Number>1234567890</Card_Number>
            <Expiry_Date>2009-09-12</Expiry_Date>
            <Issue_Date>2009-09-12</Issue_Date>
            <Issue_Number>02</Issue_Number>
            <Security_Code>333</Security_Code>
        </Payment_Details>
</Create_Customer_Account_MSG>
```
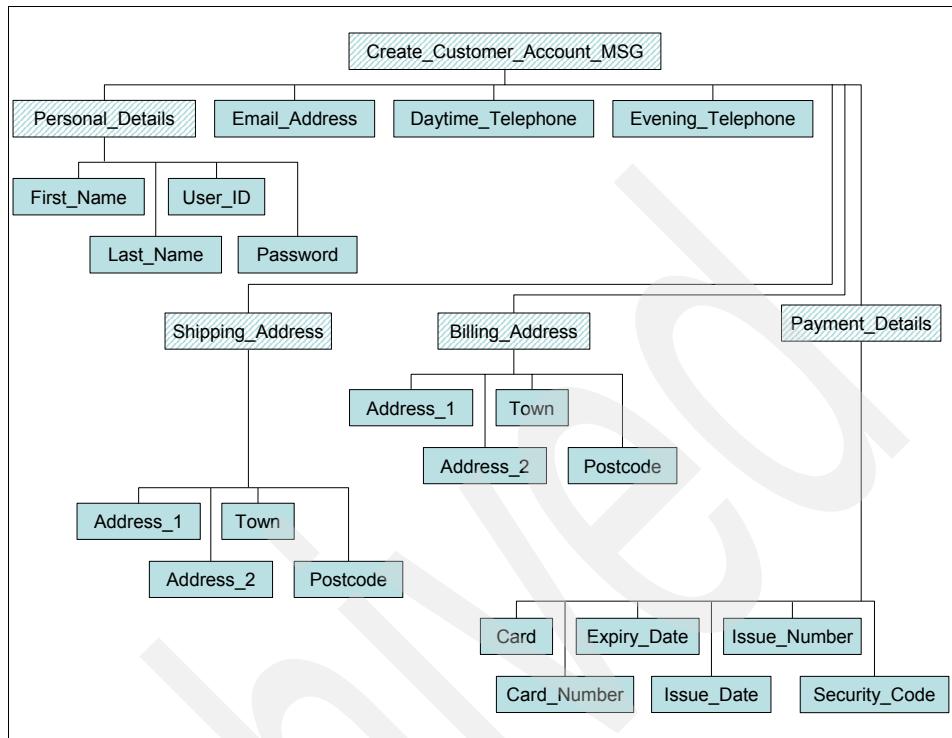
Figure 6-23   The logical structure of the Create_Customer_Account message

In Figure 6-23, the element called Create_Customer_Account_MSG represents the root of the message body, while all the other elements represent fields within the message body. The shaded elements, which are based on complex types, do not directly contain data in the message; all the unshaded elements are based on simple data types, such as string or integer, and do not directly contain data in the message.

To define the logical structure of the message for the Mapping_Create_Customer_Account message flow:

1. Create a new message definition file called Create_Customer_Account in the Mapping_Bookstore Message Set Project. The new message definition is displayed in the project and opens automatically in the Message Definition editor.

2. In the Message Definition editor, add the global elements listed in the Global element name column of Table 6-4 on page 165.

Table 6-4   The global elements and types in the logical structure of Mapping_Create_Cu

Global element name	Type
First_Name	xsd:string
Last_Name	xsd:string
User_ID	xsd:string
Password	xsd:string
Email_Address	xsd:string
Daytime_Telephone	xsd:string
Evening_Telephone	xsd:string
Address_1	xsd:string
Address_2	xsd:string
Town	xsd:string
Postcode	xsd:string
Card	xsd:string
Card_Number	xsd:string
Expiry_Date	xsd:date
Issue_Date	xsd:date
Issue_Number	xsd:int
Security_Code	xsd:int
Personal_Details	Personal_Details
Shipping_Address	Shipping_Address
Billing_Address	Billing_Address
Payment_Details	Payment_Details
Create_Customer_Account_MSG	Create_Customer_Account_MSG

3. Create the complex types for the elements shown shaded in Figure 6-23 on page 164. See Table 6-4 for the names of the complex types.

   a. In the Type column of the Message Definition editor, click the cell that contains the type for the Personal_Details element to highlight it; the value in the cell is, by default, xsd:string.

b. Click the cell again to display the drop-down list of available types.

c. Scroll to the end of the list, then click **New Complex Type** (Figure 6-24). The New Complex Type dialog opens.
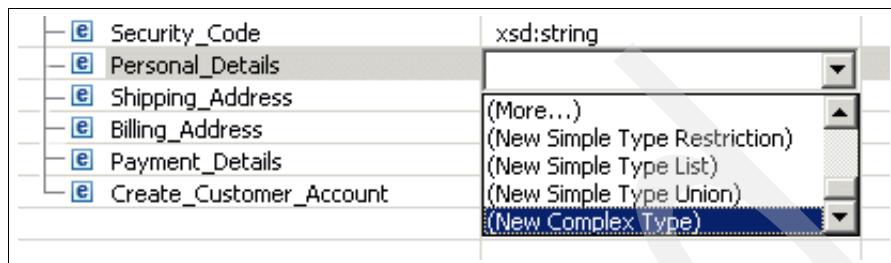
Figure 6-24   Creating a new complex type for the Personal_Details element

d. In the dialog, select the **Create as Global Complex Type** check box; in the Name field, type the name of the complex type, `Personal_Details`, then click **OK** (Figure 6-25).
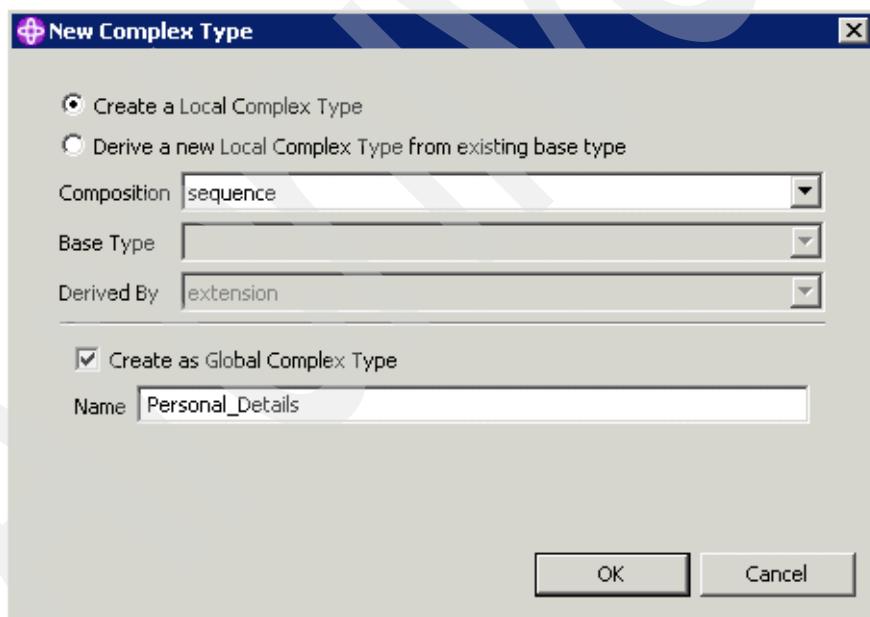
Figure 6-25   Naming new global complex type for Personal_Details element

e. Repeat the steps to create new complex types for the Shipping_Address, Billing_Address, Payment_Details, and Create_Customer_Account_MSG elements, as shown in Table 6-4 on page 165.

4. Define the Issue_Number and Security_Code elements as being of type *integer*:

   a. In the Type column of the Message Definition editor, click the cell that contains the type of the Issue_Number element to highlight the row, then click the cell again to display the drop-down list of available types.

   b. From the list, select **xsd:int**.

   c. Repeat these steps to select the type of the Security_Code element.

5. Define the Expiry_Date and Issue_Date elements as being of type *date*:

   a. In the Type column of the Message Definition editor, click the cell that contains the type of the Expiry_Date element to highlight the row, then click the cell again to display the drop-down list of available types.

   b. From the list, select **xsd:date**.

   c. Ensure that the Expiry_Date row is still highlighted, then click the **Properties** tab at the bottom of the Message Definition editor.

   d. On the Properties page, in the Properties Hierarchy, click **Physical Properties** → **XML_WIRE_FORMAT** → **Global Element**.

   e. In the DataTimeFormat field, type MM-yyyy to define the format of the date in the Expiry_Date field of the message. Click the **Overview** tab to return to the main page of the Message Definition editor.

   f. Repeat these steps to select the type of the Issue_Date element; define the format of the date to be the same as the date in the Expiry_Date field.

6. Organize the types to build the logical message structure:

   a. Expand **Types** to display the complex types that you created.

   b. Under the Types heading, right-click the **Personal_Details** complex type, then click **Add Element Reference** (Figure 6-26 on page 168).
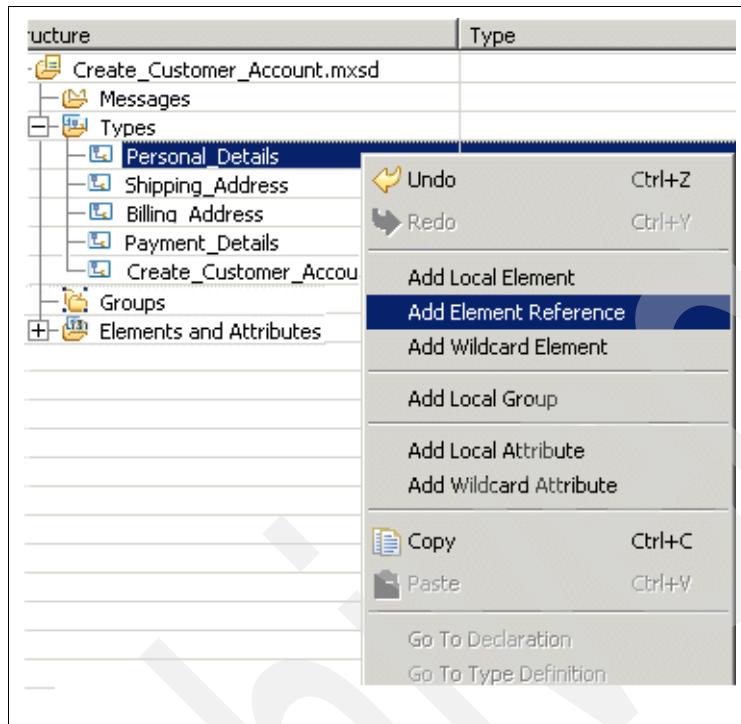
Figure 6-26   Adding an element reference to the Personal_Details complex type

c. A list of available element references is added below the Personal_Details complex type.

d. From the list of element references, select **First_Name**.

e. Add the following element references to the Personal_Details complex type, as shown in Figure 6-27 on page 169:

- Last_Name
- User_ID
- Password

f. Add the following element references to both the Shipping_Address and Billing_Addess complex types, as shown in Figure 6-27 on page 169:

- Address_1
- Address_2
- Town
- Postcode

g. Add the following element references to the Payment_Details complex type, as shown in Figure 6-27:

- Card
- Card_Number
- Expiry_Date
- Issue_Date
- Issue_Number
- Security_Code

h. Add the following element references to the Create_Customer_Account_MSG complex type, as shown in Figure 6-27:

- Personal_Details
- Email_Address
- Daytime_Telephone
- Evening_Telephone
- Shipping_Address
- Billing_Address
- Payment_Details

Figure 6-27   The element references added to the complex types

Notice in the Min Occurs and Max Occurs columns that each row has a 1 in it. These values constrain the number of times that the element, or field, can exist in the message. Positive integers (for example, 1, 2, 3) indicate the minimum and maximum number of times that the field can appear in the message, while a 0 (zero) value in the Min Occurs cell indicates that the field does not have to appear in the message.

7. In the Issue_Date row, click the **Min Occurs** cell and replace the 1 with a 0 (zero). Do the same for the Issue_Number element (Figure 6-28).

The Issue_Date and Issue_Number fields apply only to certain types of payment cards and are not applicable to credit cards. Setting the Min Occurs value to 0 means that these two fields do not have to exist in the message unless the order contains the details of a payment card to which the fields apply.

Figure 6-28   The Create_Customer_Account_MSG message structure

8. Create the root of the message body from the
   Create_Customer_Account_MSG element:

   a. Right-click **Messages**, then click **Add Message From Global Element**
      (Figure 6-29). The Add Message From Global Element dialog opens.
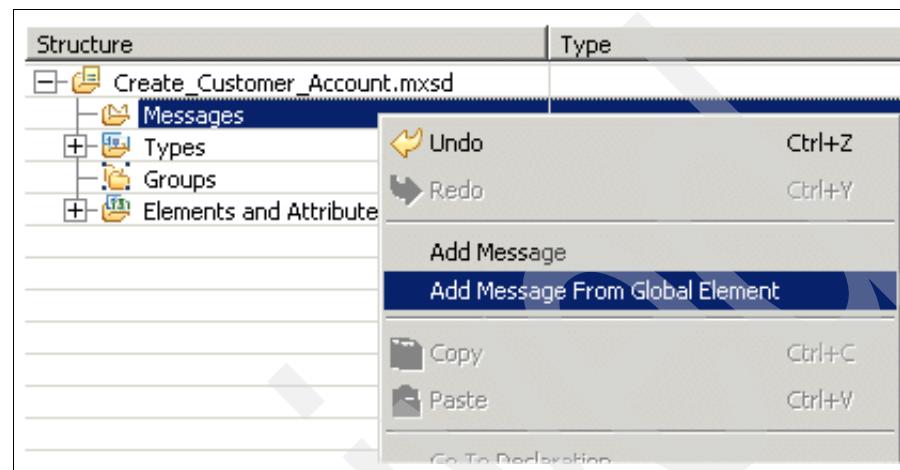
Figure 6-29   Adding a message from the Create_Customer_Account element

   b. In the dialog, select **Create_Customer_Account_MSG** from the list, then
      click **OK**. The Create_Customer_Account_MSG element is added to the
      Messages section of the Message Definition editor with its full hierarchy of
      elements below it (Figure 6-30 on page 172).

Figure 6-30   The complete Create_Customer_Account_MSG message structure

9. Save the Create_Customer_Account.mxsd file.

You have now completed the definition of the Create_Customer_Account_MSG message. Next, define the Create_Book_Order_MSG.

Defining the Create_Book_Order message

The Create_Book_Order message definition defines the structure of the input message for the Mapping_Create_Book_Order message flow. The Mapping_Create_Book_Order message flow transforms the input message to an output message that has a different structure. After you have defined the Create_Book_Order message, you will define the output message, Book_Order_Response, in the same message definition file.

An example input message for the Mapping_Bookstore message flow is shown in Example 6-4. Figure 6-31 on page 173 shows how the logical structure of the Create_Book_Order message is constructed.

Example 6-4   The input message for the Mapping_Create_Book_Order message flow

```
<Create_Book_Order_MSG>
```

```
        <Customer_ID>0123456789</Customer_ID>
        <Order_Date>2002-10-20 12:00:00</Order_Date>
        <Airmail>Yes</Airmail>
        <Book_Details>
            <ISBN>0123456789</ISBN>
            <Book_Price>15.99</Book_Price>
            <ISBN>1425112342</ISBN>
            <Book_Price>7.99</Book_Price>
            <ISBN>9736316345</ISBN>
            <Book_Price>25.99</Book_Price>
        </Book_Details>
</Create_Book_Order_MSG>
```
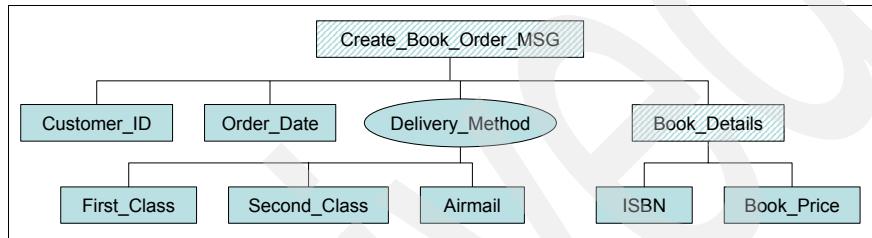

Figure 6-31   The logical structure of the Create_Book_Order message

Example 6-5 shows the output message that the message flow generates based
on the input message shown in Example 6-4 on page 172. Figure 6-32 on
page 174 shows the logical structure of the Book_Order_Response message.
The message flow has added the XML declaration to the beginning of the
message.

*Example 6-5   The Book_Order_Response_MSG message for the Mapping_Book_Order
            message flow*

```
<?xml version="1.0"?>
<Book_Order_Response_MSG>
    <Customer_ID>0123456789</Customer_ID>
    <Order_Number>0123456789TIMESTAMP &apos;2002-10-20
12:00:00&apos;</Order_Number>
    <Order_Date>2002-10-20T12:00:00-08:00</Order_Date>
    <Airmail>Yes</Airmail>
    <Delivery_Price>8</Delivery_Price>
    <Book_Details>
        <ISBN>0123456789</ISBN>
        <Book_Price>15.99</Book_Price>
        <ISBN>1425112342</ISBN>
        <Book_Price>7.99</Book_Price>
        <ISBN>9736316345</ISBN>
        <Book_Price>25.99</Book_Price>
    </Book_Details>
```

```
   <Total_Price>49.97</Total_Price>
   <Order_Status>Order Received</Order_Status>
</Book_Order_Response_MSG>
```
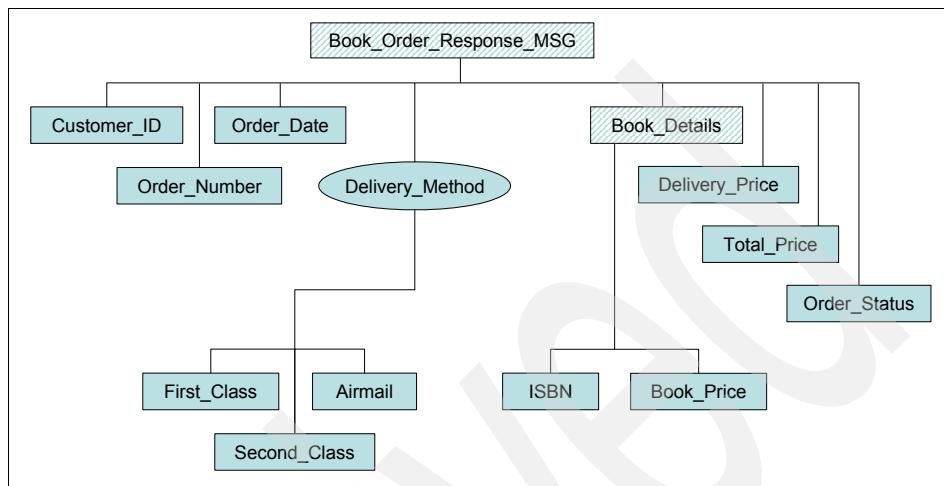

Figure 6-32   The logical structure of the Book_Order_Response message

In Figure 6-31 on page 173, the element called Delivery_Method is a *group*,
rather than an element; it represents a grouping of elements: First_Class,
Second_Class and Airmail. In a message that has the Create_Book_Order
structure, only one of these three elements can be present.

To define the logical structure of the input message for the Mapping_Book_Order
message flow:

1. Create a new message definition file called Book_Order in the
   Mapping_Bookstore Message Set Project. The new message definition
   opens in the Message Definition editor.

2. In the Message Definition editor, add the global elements and set their types
   as listed in Table 6-5.

Table 6-5   The global elements and types in the Create_Book_Order message

Global element name	Type
Customer_ID	xsd:string
Order_Number	xsd:string
Order_Date	xsd:dateTime
ISBN	xsd:string

Global element name	Type
Order_Status	xsd:string
Book_Price	xsd:decimal
Delivery_Price	xsd:decimal
Total_Price	xsd:decimal
First_Class	xsd:string
Second_Class	xsd:string
Airmail	xsd:string

3. Set the DateTime format of the value in the Order_Date field of the message:

   a. Click the **Order_Date** element to highlight it, then click the **Properties** tab at the bottom of the Message Mapping editor.

   b. On the Properties page, in the Properties Hierarchy, click **Physical properties** → **XML_WIRE_FORMAT** → **Element Reference**.

   c. In the DateTime Format field, type:

   `yyyy-MM-dd HH-mm-ss`

   d. Click the **Overview** tab to return to the main page.

4. Create the Delivery_Method group:

   a. In the Message Definition editor, right-click **Groups**, then click **Add Group**. A new group, globalGroup1, is added.

   b. Rename the group to Delivery_Method.

   c. Add the following element references to the Delivery_Method group:

      • First_Class
      • Second_Class
      • Airmail

   d. For each of the element references, change the value in the Min Occurs column to 0 (zero), as shown in Figure 6-33 on page 176. This means that each of the delivery methods can be present in the message once, or not at all.

Figure 6-33   Elements and group in the Create_Book_Order message definition

5. Change the properties of the Delivery_Method group so that only one of the three delivery methods can be present in a message, that is, a bookstore customer can select only one delivery method. To change the properties:

   a. Click **Delivery_Method** to highlight it.

   b. Click the **Properties** tab at the bottom of the Message Definition editor to view the Properties page for the Delivery_Method group.

   c. On the Properties page, ensure that **Logical properties** → **Global Group** is selected in the Properties Hierarchy.

   d. In the Composition field, select **choice** (Figure 6-34 on page 177).

   e. Click the **Overview** tab to return to the main page.
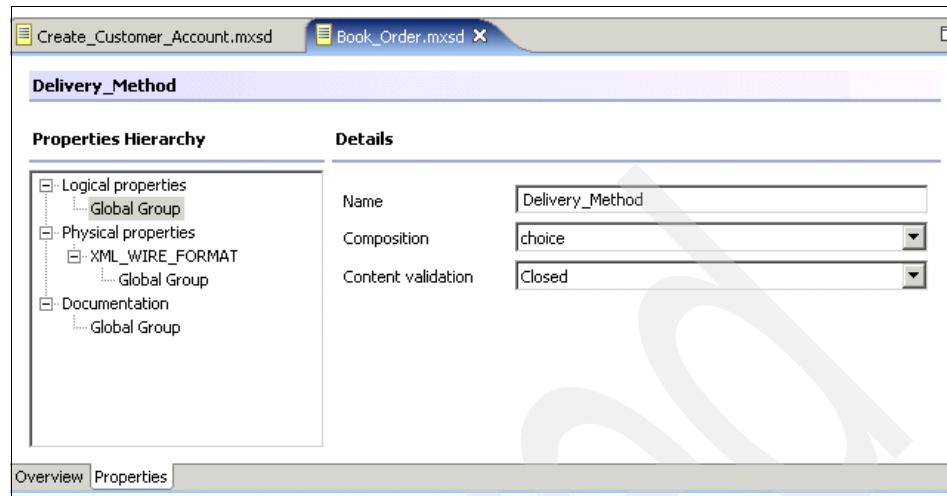
Figure 6-34   Setting the Delivery_Method group's properties

6. Define the structure for the list of books that the customer has ordered from the bookstore:

   a. Right-click **Types**, then click **Add Complex Type**. A new complex type, complexType1, is added.

   b. Rename complexType1 to Books.

   c. Add the following element references to the Books complex type (Figure 6-35):

      - ISBN
      - Book_Price

Figure 6-35   Adding the Books complex type and element references

   d. Create a new global element called Book_Details and assign it type Books:

      i. Right-click **Elements and Attributes**, then click **Add Global Element**.

      ii. Rename the element to Book_Details.

      iii. On the Book_Details row, click the cell in the Type column, then from the list click **(More...)** (Figure 6-36 on page 178). The Type Selection dialog opens.
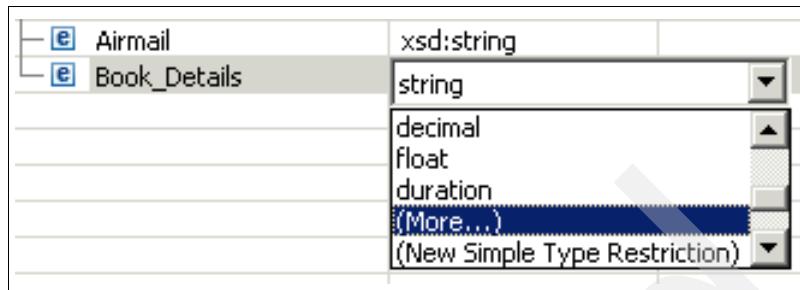
Figure 6-36   Assigning type Books to the Book_Details element

    iv. In the dialog, click **Books**, then click **OK** (Figure 6-37).
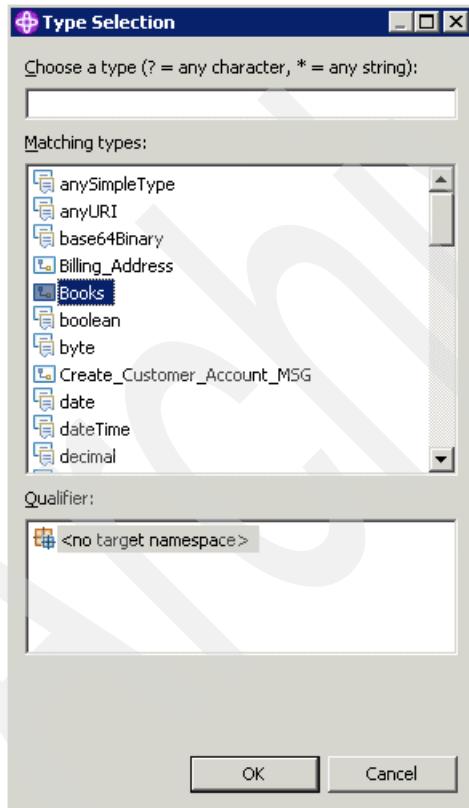
Figure 6-37   Selecting the Books complex type

  e. Set the properties of the Books complex type:

    i. Under Types, click **Books** to highlight it.

ii. Click the **Properties** tab to view the properties of the Books complex type.

iii. Ensure that **Logical properties** → **Complex Type** is selected in the Properties Hierarchy.

iv. In the Content validation field, select **Open Defined** (Figure 6-38).
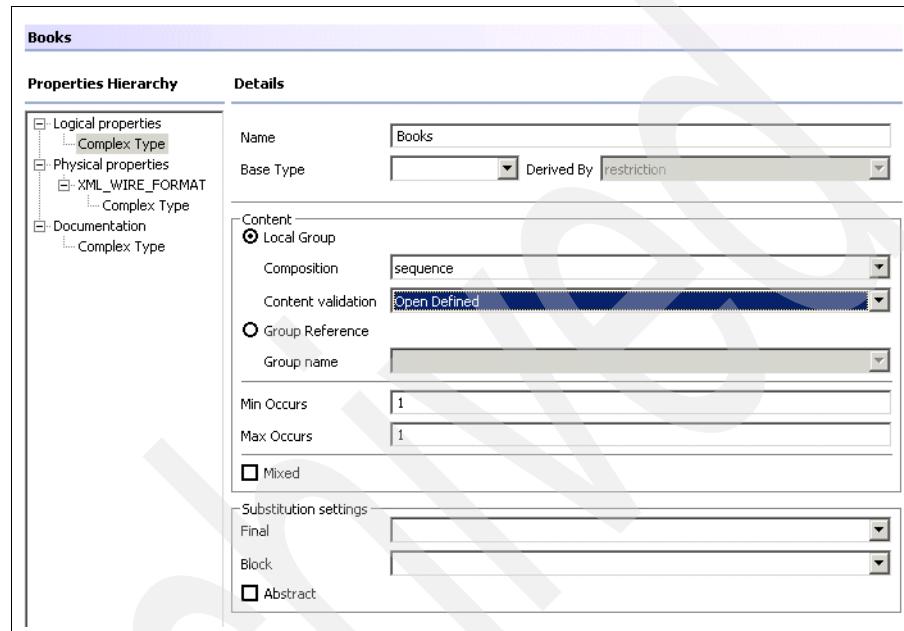
Figure 6-38   Setting the properties of the Books complex type

v. Click the **Overview** tab to return to the main page.

Now that the Book_Details element is associated with the Books complex type, the elements that were referenced by the Books complex type are now also referenced by the Book_Details element.

7. Create a message called Create_Book_Order_MSG and rename the complex type that is automatically created to Create_Book_Order.

8. Add the following element and group references to Create_Book_Order_MSG, as shown in Figure 6-39 on page 180:

   – Customer_ID (element reference)
   – Order_Date (element reference)
   – Delivery_Method (group reference)
   – Book_Details (element reference)

9. Edit the Book_Details element reference so that the Book_Details field can be repeated an unlimited number of times in a single message:

   a. On the Book_Details element reference row, click the cell in the Max Occurs column.

   b. Replace the 1 with -1. This means that the Book_Details field must appear at least once in a message (this is defined by the 1 in the Min Occurs cell) and can be repeated an infinite number of times in the same message.

Figure 6-39   The Create_Book_Order_MSG message structure

You have now defined the structure of the Create_Book_Order message, the input message for the Mapping_Create_Book_Order message flow. Now define the Book_Order_Response message, the output message from the Mapping_Create_Book_Order message flow. Define the output message in the same message definition file as the input message so that you can reuse some of the same elements and the Delivery_Method group.

To define the Book_Order_Response message:

1. In the Book_Order.mxsd file in the Message Definition editor, add a new message called Book_Order_Response_MSG, and rename its complex type to Book_Order_Response.

2. Add the following element and group references to Book_Order_Response_MSG:

   – Customer_ID (element reference)
   – Order_Number (element reference)
   – Order_Date (element reference)
   – Delivery_Method (group reference)
   – Book_Details (element reference)
   – Delivery_Price (element reference)
   – Total_Price (element reference)
   – Order_Status (element reference)

3. Edit the Book_Details element reference so that the Book_Details field can be repeated an unlimited number of times in a single message:

   a. On the Book_Details element reference row, click the cell in the Max Occurs column.

   b. Replace the 1 with -1. This means that the Book_Details field must appear at least once in a message (this is defined by the 1 in the Min Occurs cell) and can be repeated an infinite number of times in the same message.

Figure 6-40 Element and group references in Book_Order_Response message

You have now defined the Book_Order_Response message. Next, create the Mapping_Create_Customer_Account and Mapping_Book_Order message flows so that you can test the message definitions.

6.3.2 Creating the Create_Customer_Account message flow

The Mapping_Create_Customer_Account message flow uses mappings in a DataInsert node to create accounts in a DB2 database table for new customers who have registered their details with the bookstore, for example, their contact details and delivery address.

Because you are mapping message elements to fields in a database table, the following instructions include how to create a connection to the database from the Message Brokers Toolkit. The files for the database connection are stored in the Message Flow project.

To create the Mapping_Create_Customer_Account message flow, first create the files in which the message flow is stored:

1. In the Broker Application Development perspective, create a Message Flow project called Mapping_Bookstore Message Flow Project. When you have entered the name for the project, click **Next**.

2. Select the **Mapping_Bookstore Message Set Project** check box. This means that the message flows in this project can find the message set that contains the message definitions.

3. Create a new relational database (RDB) connection file so that you can access the Bookstore scenario database, BSTOREDB, from the Message Brokers Toolkit:

   a. Click **File** → **New** → **RDB Definitions Files**. The New Database Connection wizard opens (Figure 6-41).

   b. In the wizard, click **Next**.
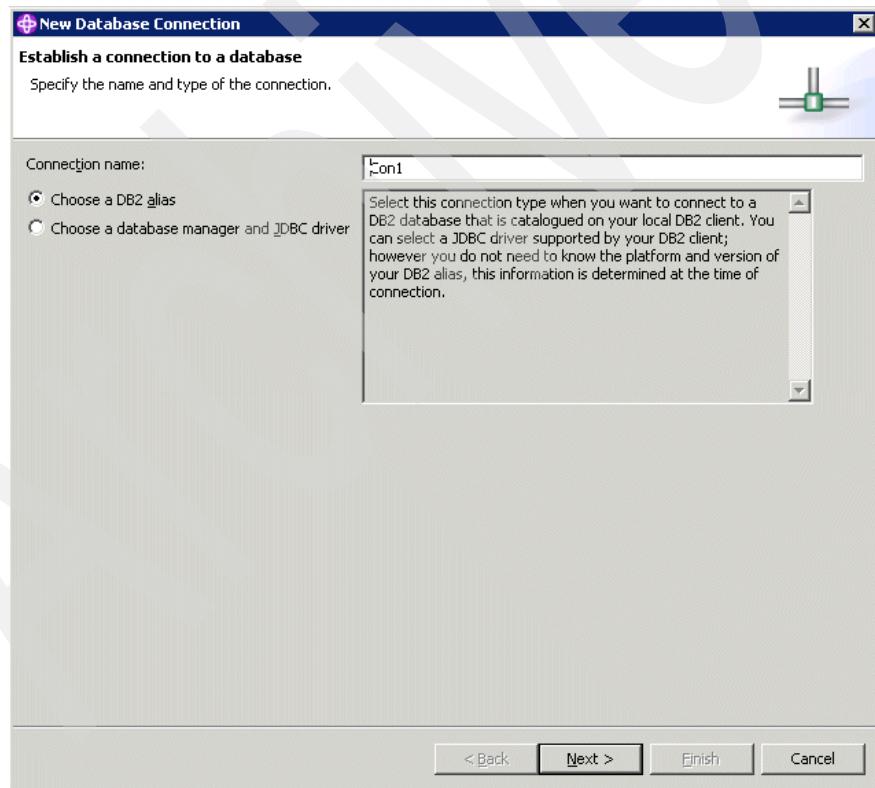
Figure 6-41   Creating database connection files from Message Brokers Toolkit

c. Click **Test Connection** to ensure that the Message Brokers Toolkit can connect to the BSTOREDB database (Figure 6-42). When the message is displayed saying that the connection is successful, click **OK**.
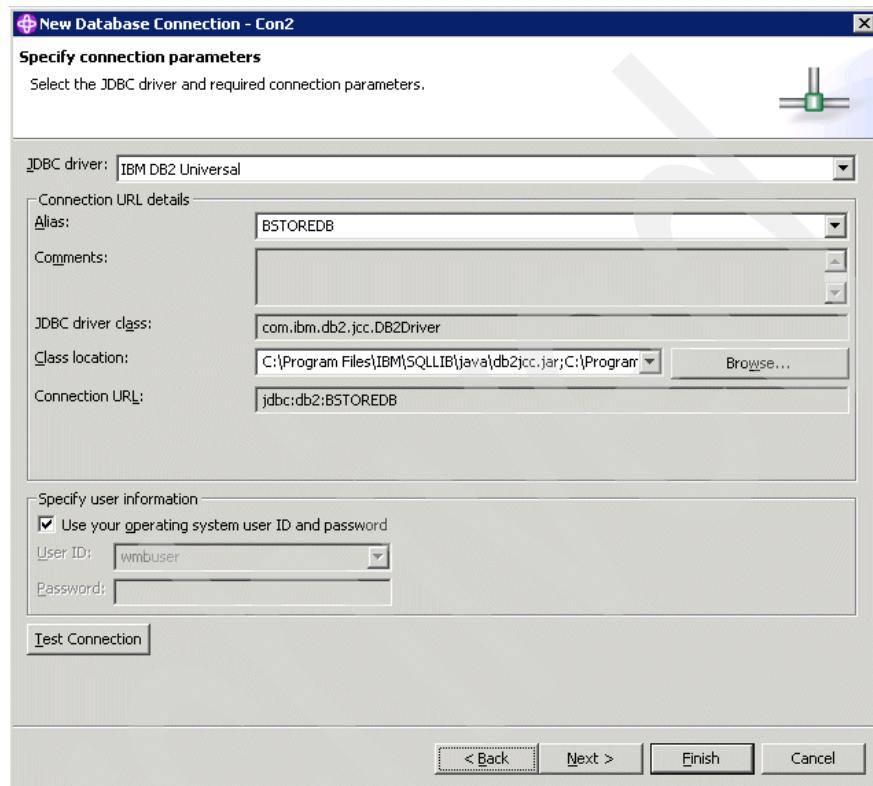
Figure 6-42   Testing the connection to the BSTOREDB database

d. Click **Finish**. The Import RDB Definition wizard opens.

e. In the Project field, browse for or type: `Mapping_Bookstore Message Flow Project` (Figure 6-43 on page 184). This is where the database connection files are stored so that your Mapping_Bookstore message flows can access them.

f. Click **Finish**.
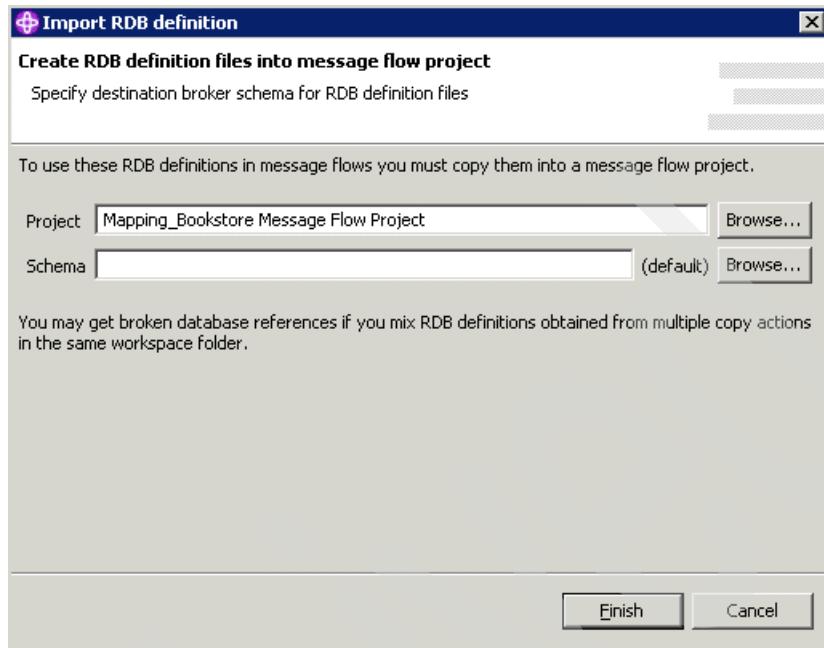
Figure 6-43   Specifying where to store the database connection files

The database connection files are displayed in the Resource Navigator view in Mapping_Bookstore Message Flow Project (Figure 6-44).
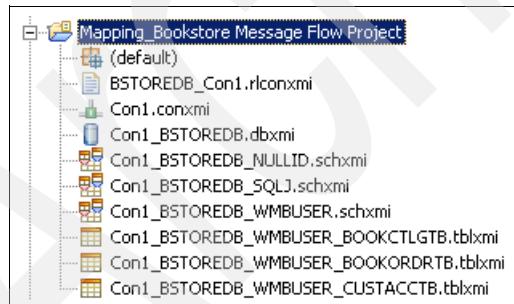
Figure 6-44   The database connection files in the Message Flow project

> **Tip:** If the database changes after you have created the connection to the
> Message Brokers Toolkit:
>
> 1. Switch to the Data perspective.
>
> 2. In the Database Explorer view, ensure that the database is connected. If
>   not, right-click the connection, then click **Reconnect**.
>
> 3. In the Data Definition view, delete from the Message Flow project the
>   tables that have changed.
>
> 4. In the Database Explorer view, right-click the connection, then click
>   **Refresh**.
>
> 5. Right-click the table that has changed, then click **Copy to Project...**. The
>   Copy to Project wizard opens.
>
> 6. In the wizard, select the Message Flow project folder, then click **Finish**.
>
> The table is added into the Data Definition view.

Adding, connecting, and configuring the nodes

Figure 6-45 shows the finished Mapping_Create_Customer_Account message
flow.
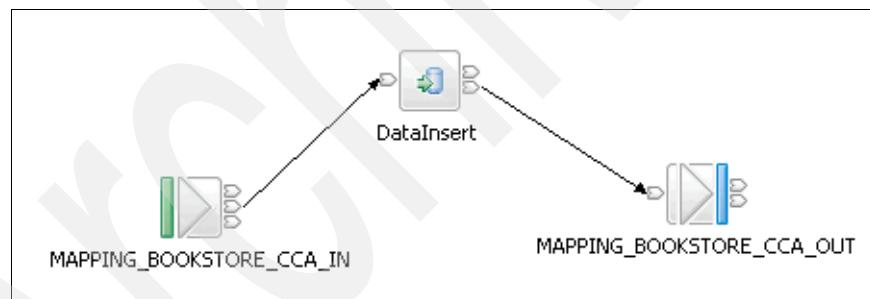
Figure 6-45   The Mapping_Create_Customer_Account message flow

To create the Mapping_Create_Customer_Account message flow:

1. Create a message flow called Mapping_Create_Customer_Account in the
   Mapping_Bookstore Message Flow Project. The
   Mapping_Create_Customer_Account.msgflow file opens in the Message
   Flow editor.

2. In the Message Flow editor, add the nodes listed in Table 6-6 on page 186 to
   the canvas.

3. Connect the nodes together, as shown in Table 6-7 on page 186.

Chapter 6. Developing applications with mappings   **185**

Table 6-6   The Mapping_Create_Customer_Account message flow nodes

Node type	Node name
MQInput	MAPPING_BOOKSTORE_CCA_IN
DataInsert	DataInsert
MQOutput	MAPPING_BOOKSTORE_CCA_OUT

Table 6-7   Node connections in the Mapping_Create_Customer_Account message flow

Node name	Terminal	Connect to this node
MAPPING_BOOKSTORE _CCA_IN	Out	DataInsert
DataInsert	Out	MAPPING_BOOKSTORE _CCA_OUT

4. Set the properties of the nodes, as shown in Table 6-8.

Table 6-8   Node properties for the Mapping_Create_Customer_Account message flow

Node name	Page	Property	Value
MQInput	Basic	Queue Name	MAPPING_BOOK STORE_CCA_IN
	Default	Message Domain	MRM
		Message Set	Mapping_Bookstor e Message Set
		Message Type	Create_Customer_ Account_MSG
		Message Format	XML_WIRE_FOR MAT
DataInsert	Basic	Data Source	BSTOREDB
MQOutput	Basic	Queue Name	MAPPING_BOOK STORE_CCA_OU T

Creating the mappings for the message flow

The DataInsert node is similar to the Mapping node in that you can use the graphical Message Mapping editor to create the mappings, but the DataInsert node is specialized for inserting data into a database table.

To create the message map file:

1. In Mapping_Create_Customer_Account.msgflow, right-click the **DataInsert** node, then click **Open Map**. The New Message Map wizard opens.

2. Accept the values on the first page of the wizard and click **Next**.

3. Ensure that **This map is called from a message flow and maps properties and message body** is selected, then click **Next**.

4. Ensure that **insert into a database table** is selected, then click **Next**.

5. On the Source and Target Mappables page, in the **Source** pane, expand **Mapping_Bookstore Message Set**, then select **Create_Customer_Account_MSG**.

6. In the Target pane, expand **BSTOREDB**, expand the database schema (probably your Windows user name), then click the **CUSTACCTB** table. This is the table into which the Mapping_Create_Customer_Account message flow inserts data from the input message.

7. Click **Finish**.

The new message map file, Mapping_Create_Customer_Account.msgmap, opens in the Message Mapping editor. In the Message Mapping editor, map the fields in the message (the source) to their equivalent fields in the CUSTACCTB database table (the target).

To map the input message fields to the database table:

1. In the Source pane, expand the whole of the Create_Customer_Accout_MSG message structure; in the Target pane, expand the database table.

2. Drag and drop the fields from the message in the Source pane to their equivalent fields in the database table in the Target pane, as listed in Table 6-9, "Mapping the input message elements to the database table fields" on page 187. Figure 6-46 on page 189 shows the completed mappings. Notice that only the elements based on simple types (that is, the fields that directly contain data) are mapped.

Table 6-9   Mapping the input message elements to the database table fields

Source element	Target field
First_Name	FIRST_NAME
Last_Name	LAST_NAME
User_ID	USERID
Password	PASSWORD
Email_Address	EMAIL

Source element	Target field
Daytime_Telephone	DAY_PHONE
Evening_Telephone	EVE_PHONE
Address_1 (under Shipping_Address)	SHIP_ADDRESS1
Address_2 (under Shipping_Address)	SHIP_ADDRESS2
Town (under Shipping_Address)	SHIP_TOWN
Postcode (under Shipping_Address)	SHIP_POSTCODE
Address_1 (under Billing_Address)	BILL_ADDRESS1
Address_2 (under Billing_Address)	BILL_ADDRESS2
Town (under Billing_Address)	BILL_TOWN
Postcode (under Billing_Address)	BILL_POSTCODE
Card	CARDTYPE
Card_Number	CARDNUM
Expiry_Date	EXP_DATE
Issue_Date	ISS_DATE
Issue_Number	ISS_NUM
Security_Code	SECCODE
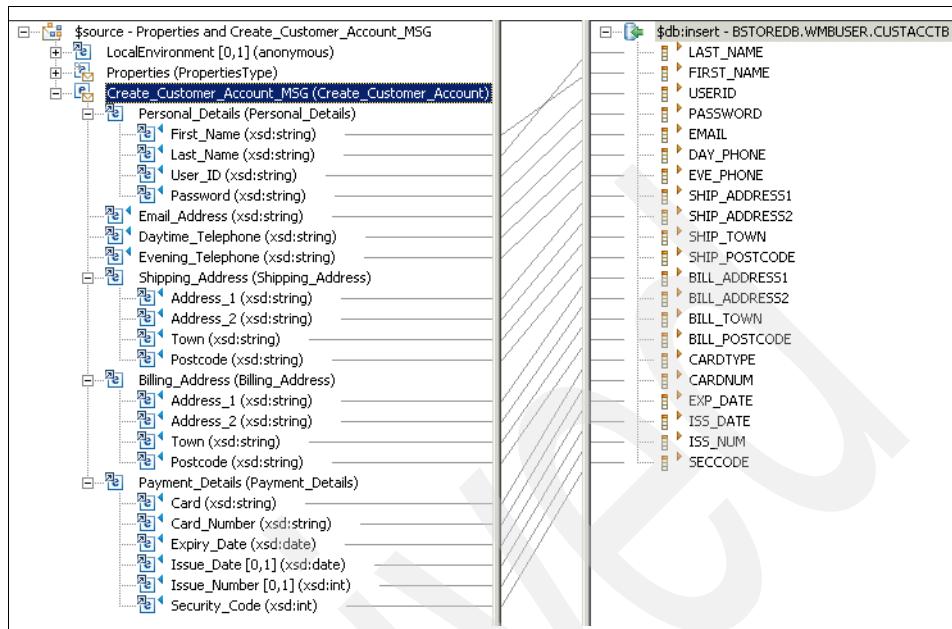
Figure 6-46   Mapping the input message elements to the database table fields

3. Save the Mapping_Create_Customer_Account.msgmap and Mapping_Create_Customer_Account.msgflow files.

4. Create the following WebSphere MQ queues in WebSphere MQ Explorer on the WBRK6_DEFAULT_QUEUE_MANAGER queue manager:

   – MAPPING_BOOKSTORE_CCA_IN
   – MAPPING_BOOKSTORE_CCA_OUT

   Remember to enter the value of the Backout dequeue queue property on the MAPPING_BOOKSTORE_CCA_IN queue as `DLQ`.

You have created the Mapping_Create_Customer_Account message flow. Next, create the Mapping_Book_Order message flow.

> **Tip:** If you change a message definition after using it in a message map, the message map is automatically updated with the changes.

6.3.3 Creating the Mapping_Book_Order message flow

The Mapping_Book_Order message flow uses mappings in a Mapping node to process an order that has been submitted by an online customer. The order contains the customer's identification and details of the books that they have ordered. When the message flow processes the message, it creates a

confirmation message that contains details of the order, including a unique order number.

To create the Mapping_Book_Order message flow, first create the file in which the message flow is stored. In the Broker Application Development perspective, create a message flow called Mapping_Book_Order in the Mapping_Bookstore Message Flow Project.

Adding, connecting, and configuring nodes

Figure 6-47 shows the finished Mapping_Book_Order message flow.
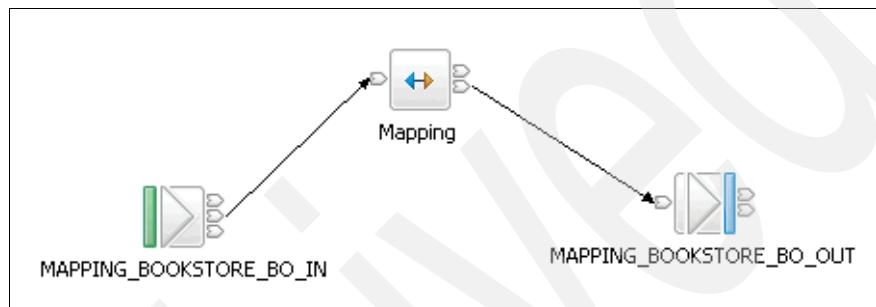
Figure 6-47   The Mapping_Book_Order message flow

1. In the Message Flow editor, add the nodes listed in Table 6-10 to the canvas.

2. Connect the nodes together, as shown in Table 6-11.

Table 6-10   The Mapping_Book_Order message flow nodes

Node type	Node name
MQInput	MAPPING_BOOKSTORE_BO_IN
Mapping	Mapping
MQOutput	MAPPING_BOOKSTORE_BO_OUT

Table 6-11   Node connections in the Mapping_Book_Order message flow

Node name	Terminal	Connect to this node
MAPPING_BOOKSTORE _BO_IN	Out	Mapping
Mapping	Out	MAPPING_BOOKSTORE _BO_OUT

3. Set the properties of the node as shown in Table 6-12 on page 191.

Table 6-12   Node properties for the Mapping_Book_Order message flow

Node name	Page	Property	Value
MQInput	Basic	Queue Name	MAPPING_BOOK STORE_BO_IN
	Default	Message Domain	MRM
		Message Set	Mapping_Bookstor e Message Set
		Message Type	Create_Book_Ord er_MSG
		Message Format	XML
MQOutput	Basic	Queue Name	MAPPING_BOOK STORE_BO_OUT

Creating the mappings for the message flow

To create the message map file:

1. In Mapping_Book_Order.msgflow, right-click the **Mapping** node, then click **Open Map**. The New Map wizard opens.

2. Accept the values on the first page of the wizard and click **Next**.

3. Ensure that **This map is called from a message flow and maps properties and message body** is selected, then click **Next**.

4. Select the input message as the message source; clear the database records check box, then click **Next**.

5. In the Source pane, expand **Mapping_Bookstore Message Set**, then select **Create_Book_Order_MSG**. This is the input message to the message flow.

6. In the Target pane, expand **Mapping_Bookstore Message Set**, then select **Book_Order_Response_MSG**. This is the output message from the message flow.

7. Click **Finish**.

The new message map file, Mapping_Book_Order.msgmap, opens in the Message Mapping editor. In the Message Mapping editor, map the fields in the input message (the source) to fields in the output message (the target).

To map the input message fields to the output message fields:

1. In the Source pane, expand the whole of the Create_Book_Order_MSG message structure, and in the Target pane, expand the whole of the Book_Order_Response_MSG message structure (Figure 6-48 on page 192).

2. In the Source pane, click **Create_Book_Order_MSG**.

3. In the Target pane, click **Book_Order_Response_MSG**.

4. Click **Finish**. The source and target structures are added to the Source and Target panes, respectively, in the Message Mapping editor.
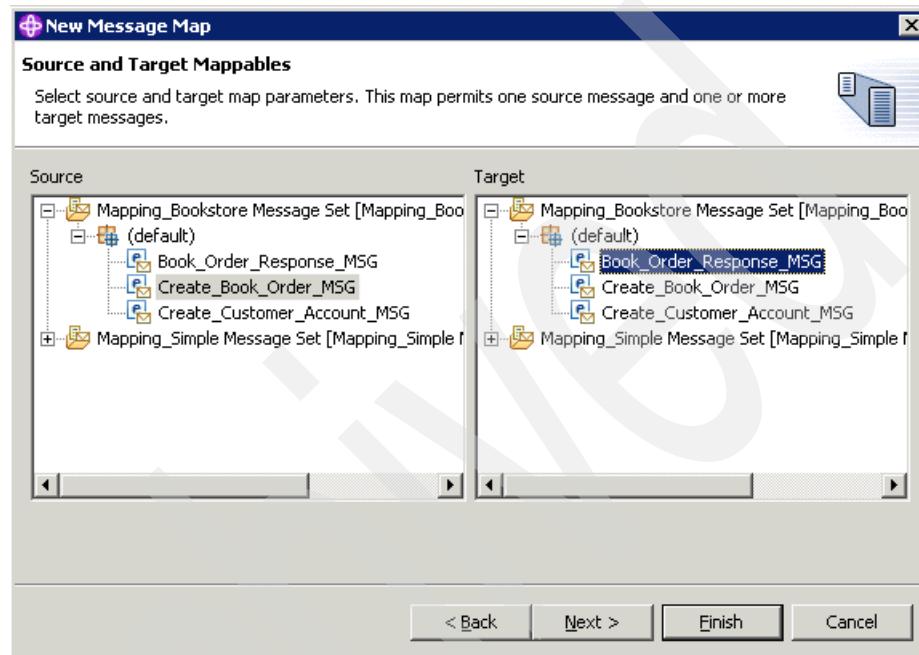
Figure 6-48   Selecting the source and target messages

> **Tip:** When you are creating mappings, try to map the fields in the order in which they occur in the message definition. You can go back and edit earlier mappings, but the automatic generation of statements can change later mappings. If this happens, make sure that you remove any automatically generated statements that you do not need.

5. In the Message Mapping editor copy message properties that are associated with the message set from the input message to the output message:

   a. Expand **Properties** in both the Source and Target panes.

   b. Drag and drop the following properties from the Source pane to the Target pane (Figure 6-49 on page 193):

      • Map MessageSet to MessageSet
      • Map MessageType to MessageType

– Map MessageFormat to MessageFormat
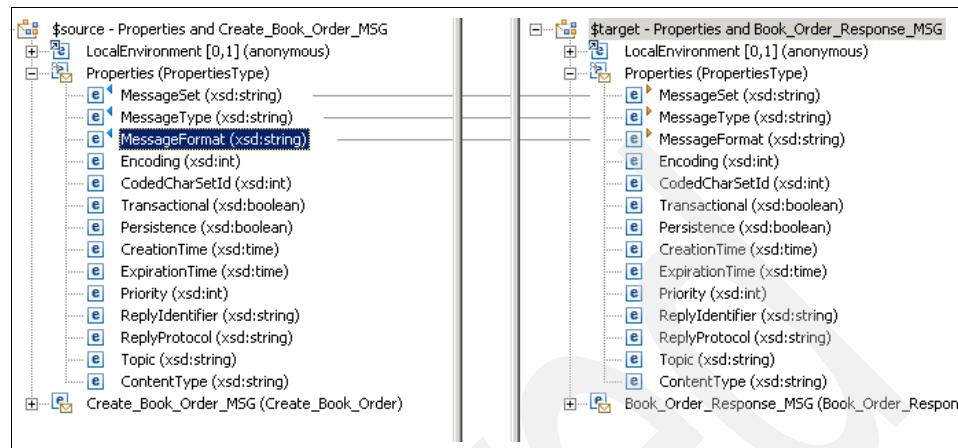
Figure 6-49   Mapping properties from input message to output message

c. Edit the MessageType property so that the message type in the output message is the Book_Order_Response_MSG message:

   i. In the spreadsheet at the bottom of the Message Mapping editor, click the cell in the Value column for the MessageType property.

   ii. In the cell, type '`Book_Order_Response_MSG`' (including the single quotation marks). You can either type directly into the spreadsheet cell, or you can highlight the cell and then type in the Expression editor above the spreadsheet (Figure 6-50 on page 194).

      Notice, in the top part of the Message Mapping editor, that the line between the MessageType property in the source and target has disappeared now, although small colored arrows are still displayed to show that there is an association despite not being a direct mapping.

Figure 6-50   Edit the value of the MessageType property in the output message

6. Map each delivery method so that whatever the delivery method, the correct choice goes to the output message. Do this using *if* and *condition* statements in the spreadsheet:

   a. Expand the **Create_Book_Order_MSG** message in the Source pane, and the **Book_Order_Response_MSG** in the Target pane.

   b. Drag and drop each of the delivery methods from the source to the target so that First_Class maps to First_Class, Second_Class maps to Second_Class, and Airmail maps to Airmail (Figure 6-51 on page 195).

      The three delivery method elements are displayed in the spreadsheet with *if* and *condition* statements. By default, the condition is fn:false() so you need change it so that the expression is to check if the message is true. That is, if the message contains the First_Class field, the message flow can determine that the customer selected the first class delivery method; if the First_Class field is not present, the message flow checks for whether the Second_Class field is present, and so on.

   c. In the spreadsheet, click the Value cell on the condition row for First_Class (the cell currently contains the value fn:false()).

d. Delete the existing value so that the cell is empty.
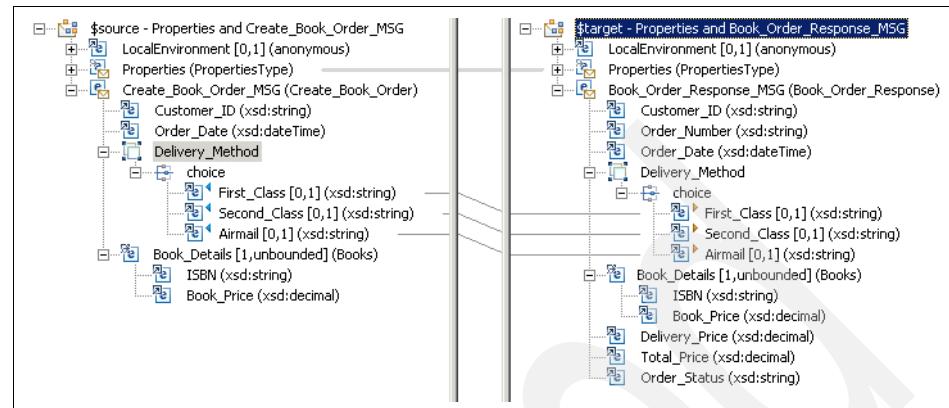
Figure 6-51   Mapping the delivery methods

e. Drag and drop the First_Class element from the Source pane to the Expression editor, then edit the expression by adding =ʼYesʼ to the end of it, and press Enter to create the following expression (as shown in Figure 6-52):

```
$source/Create_Book_Order_MSG/First_Class='Yes'
```
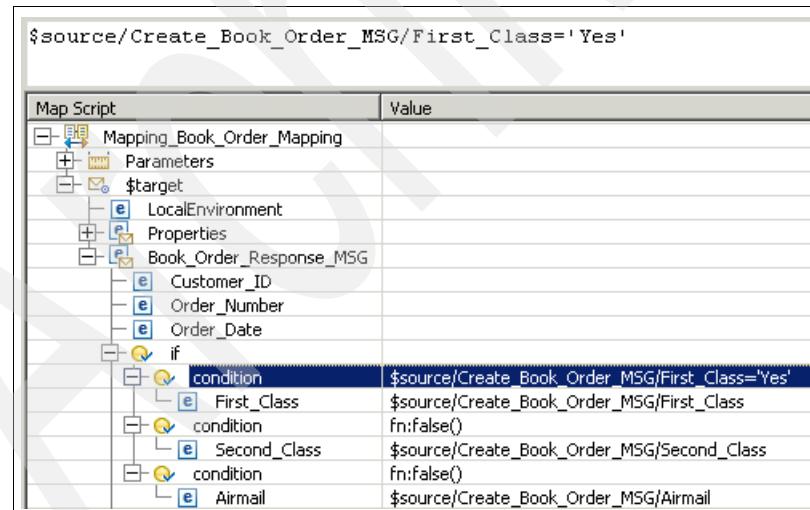

Figure 6-52   Editing the condition expression for the First_Class element

f. Edit the condition expressions in the same way for the Second_Class and Airmail elements so that the spreadsheet matches the one shown in Figure 6-53 on page 196.

Figure 6-53   The complete delivery method mappings in the spreadsheet

7. Drag and drop the **Book_Details** element from the Source pane to the Target pane to create a *for* statement. This means that for each Book_Details element that is present in the input message, the ISBN and Book_Price elements are copied to the output message.

   Ensure that the Book_Details, ISBN, and Book_Price elements are arranged as shown in Figure 6-54.

Figure 6-54   Creating a for statement for the Book_Details element

8. Map the Order_ID and Order_Date elements in the input message to their equivalent elements in the output message, as shown in Figure 6-55 on page 197:

   a. Drag and drop the **Order_ID** element from the Source pane to the Order_ID element in the Target pane.

   b. Drag and drop the **Order_Date** element from the Source pane to the Order_Date element in the Target pane.

9. Map the Order_ID and Order_Date elements in the input message to the Order_Number element in the output message. The values in the Order_ID and Order_Date fields of the input message are concatenated to make a unique order number, which is put in the Order_Number field.

   a. In the Target pane, right-click the **Order_Number** element, then click **Enter Expression**.

   b. In the Expression editor, press Ctrl+Spacebar to display the code assist list.

   c. From the list, click **fn:concat**. The expression fn:concat() is added to the the Expression editor.

d. Drag and drop **Customer_ID** from the Source pane to between the parentheses:

```
fn:concat($source/Create_Book_Order_MSG/Customer_ID)
```

e. Add a comma (**,**) between the $source statement and the closing parenthesis:

```
fn:concat($source/Create_Book_Order_MSG/Customer_ID,)
```

f. Drag and drop **Order_Date** from the source pane to between the comma and the closing parenthesis (Figure 6-55.):

```
fn:concat($source/Create_Book_Order_MSG/Customer_ID,$source/Create_Book_
Order_MSG/Order_Date)
```


Figure 6-55   Writing expression to create unique order number in output message

10. Change the cost of the delivery depending on the delivery type:

a. In the Target pane, right-click **Delivery_Price**, then click **Enter Expression**.

b. In the spreadsheet, right-click **Delivery_Price**, then click **If**. An *if* and *condition* statement is added to the spreadsheet to contain the Delivery_Price element.

c. In the spreadsheet, right-click **condition** (in the row above Delivery_Price), then click **Else**. An *else* statement and another Delivery_Price entry are added to the spreadsheet.

d. Right-click the second **Delivery_Price** entry, then click **If**. An *if* and *condition* statement is added within the *else* statement to contain the second Delivery_Price entry.

e. Right-click **condition** (above the second Delivery_Price entry), then click **Else**. An *else* statement and a third Delivery_Price entry are added to the spreadsheet.

f. Right-click the third **Delivery_Price** entry, then click **If**. An *if* and *condition* statement are added within the *else* statement to contain the third Delivery_Price entry.

Figure 6-56 shows how the Delivery_Price entries in the spreadsheet should now look.
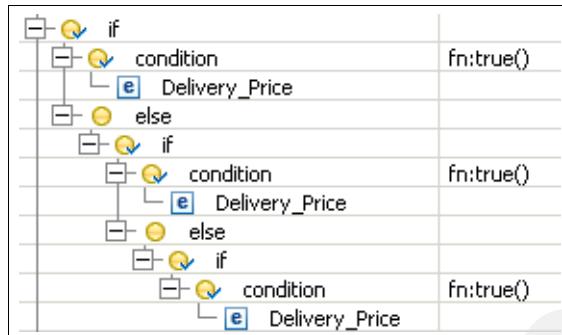
Figure 6-56   Entering the statements for determining delivery price

g. In the first cell (as shown in Figure 6-56) that contains `fn:true()`, delete the contents of the cell, then drag and drop **First_Class** from the source pane to the Expression editor.

h. In the Expression editor, add `='Yes'` to the end of the expression, then press Enter:

    `$source/Create_Book_Order_MSG/First_Class='Yes'`

i. Replace the other two fn:true() expressions in the same way for the Second_Class and Airmail elements.

j. Click the **Value** cell for the first Delivery_Price entry and type `18.00`.

k. Click the **Value** cell for the second **Delivery_Price** entry and type `12.00`.

l. Click the **Value** cell for the third Delivery_Price entry and type `8.00`.

Figure 6-57 shows the completed expressions in the spreadsheet.

Figure 6-57   Completing the delivery price expressions

Figure 6-60 on page 201 shows the spreadsheet in the finished mapping file.

11. Calculate the total price (Total_Price) of the book order:

   a. In the Target pane, right-click the **Total_Price** element, then click **Enter Expression**.

   b. In the Expression editor, press Ctrl+Spacebar to open the Content Assist menu, then from the menu click **fn:sum**, as shown in Figure 6-58.

   c. Drag and drop the **Book_Price** element from the Source pane to between the parentheses in the Expression editor:

   ```
   fn:sum($source/Create_Book_Order_MSG/Book_Details/Book_Price)
   ```
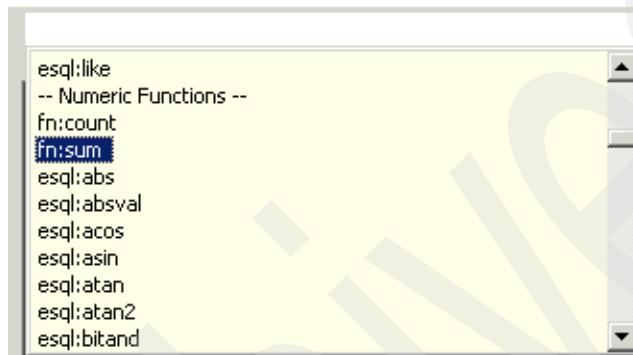

Figure 6-58   Selecting the fn:sum function from Content Assist

   d. Press Enter to update the spreadsheet, as shown in Figure 6-59 on page 200.

Figure 6-59   Creating the function to calculate the total price of the books ordered

12. Insert the value of the Order_Status element in the output message:

   a. In the Target pane, right-click the **Order_Status** element, then click **Enter Expression**.

   b. In the Expression editor, type `'Order received'` (including the single quotation marks), then press Enter to update the spreadsheet, as shown in Figure 6-60 on page 201.

Figure 6-60   The finished mapping file for the Mapping_Book_Order message flow

13.Save the Mapping_Book_Order.msgmap file.

14.Create the following WebSphere MQ queues in WebSphere MQ Explorer on the WBRK6_DEFAULT_QUEUE_MANAGER queue manager:

   – MAPPING_BOOKSTORE_BO_IN
   – MAPPING_BOOKSTORE_BO_OUT

15.Remember to enter the value of the Backout dequeue queue property on the MAPPING_BOOKSTORE_BO_IN queue as DLQ.

Next, deploy and test the Mapping_Create_Customer_Account and Mapping_Book_Order message flows.

6.3.4 Deploying and testing the Mapping Bookstore message flows

To test the Mapping Bookstore message flows,
Mapping_Create_Customer_Account and Mapping_Book_Order, you must
deploy them to the broker.

To deploy the Mapping Bookstore message flows to the broker:

1. Switch to the Broker Administration perspective**.**

2. Create a bar file called Mapping_Bookstore.bar.

3. Add to the bar file the Mapping_Create_Customer_Account.msgflow file the
   Mapping_Book_Order.msgflow file, and the Mapping_Bookstore Message
   Set Project, then save the bar file.

4. Create a new execution group on the WBRK6_DEFAULT_BROKER broker
   called Mapping_Bookstore.

5. Ensure that the WBRK6_DEFAULT_BROKER broker and the
   WBRK6_DEFAULT_CONFIGURATION_MANAGER Configuration Manager
   are running, then deploy the Mapping_Bookstore.bar file to the
   Mapping_Bookstore execution group.

   In the Domains view, the two message flows and the message set are
   displayed under the Mapping_Bookstore execution group.

6. Create a new enqueue file called
   Mapping_Create_Customer_Account.enqueue and use it to put the message
   in Example 6-3 on page 162 on the MAPPING_BOOKSTORE_CCA_IN
   queue on the WBRK6_DEFAULT_QUEUE_MANAGER queue manager. You
   can copy the message content from the Web material available to download
   (see Appendix B, "Code" on page 319).

7. Use the Dequeue wizard to get the output message, which should contain the
   same message data as the input message, from the
   MAPPING_BOOKSTORE_CCA_OUT queue on the same queue manager.

8. Use the DB2 Control Center to check that the CUSTACCTB table in the
   BSTOREDB database has been updated with the information from the input
   message. For instructions about using the DB2 Control Center, see 4.3.4,
   "Deploying and testing the ESQL Bookstore message flows" on page 93.

9. Try changing some of the field values in the input message in the
   Mapping_Create_Customer_Account.enqueue file, then put the message
   through the message flow. Another row is added to the CUSTACCTB table
   with the values that you entered in the input message.

10. Create a new enqueue file called Mapping_Book_Order.enqueue to put the
    message in Example 6-4 on page 172 on the
    MAPPING_BOOKSTORE_BO_IN queue on the
    WBRK6_DEFAULT_QUEUE_MANAGER queue manager.

11. Use the Dequeue wizard to get the output message, which should contain the message in example, from the MAPPING_BOOKSTORE_BO_OUT queue on the same queue manager.

12. Try changing the details in the input message in Java_Book_Order.enqueue; for example, add another book to the order or change the price of one of the books. Put the message through the message flow and check the output message to see how your changes affected the output message.

If the message does not output the correct message, or if the message flow cannot process the message, see **Chapter 8, "Troubleshooting and problem determination" on page 241**, for information about problem determination.

6.4 Summary

You have now created, deployed, and tested two message flow applications in which you defined the logic of the message flows using mappings.

For more information about the built-in nodes available in WebSphere Message Broker, see the product documentation: **Developing applications** → **Developing message flow applications** → **Designing a message flow** → **Deciding which nodes to use**.

7

Administration

This chapter provides an overview of the administration of the runtime environment for WebSphere Message Broker using the Message Brokers Toolkit and the command-line interface. The following topics are discussed in this chapter:

► Creating a broker domain
► Adding a remote broker to a domain
► Deploying message flow applications
► Adding version information to resources
► Publish\subscribe

205

7.1 WebSphere Message Broker administration

WebSphere Message Broker is comprised of two principle parts, a *development time* for the creation of message flows, message sets, and other message flow application resources; and a *runtime*, which contains the components for running those message flow applications created in the Message Brokers Toolkit. Administration in WebSphere Message Broker is concerned with the creation, control, maintenance, and deletion of objects in the runtime, including deployed message flow applications.

Although a great deal of the administration of WebSphere Message Broker can be performed in the Message Brokers Toolkit, some administration can only be performed on the command line. Information is given in this chapter about performing administration using both of these methods where appropriate as well as some brief discussion of the third alternative, the Configuration Manager Proxy API.

This chapter provides a background to the administration of WebSphere Message Broker and related concepts such as message flow application deployment, adding version information to message flow application resources, and the basics of publish/subscribe.

7.2 Creating a broker domain

A broker domain is one or more brokers that share a common configuration, together with a single Configuration Manager that controls them. A broker domain is used to test and run message flow applications such as message flows and message sets. The broker domain is often described as the WebSphere Message Broker's runtime environment. In addition to the Configuration Manager and brokers, the broker domain may also contain a User Name Server if authentication is required for publish\subscribe applications.

Information about the resources and components in the broker domain are stored in the Configuration Manager in an internal repository. The broker domain can be administered in one of three ways, of which only two are covered in this chapter. The ways of administering the broker domain are as follows:

► Using the Broker Administration perspective
► Using the Command Console
► Using the Configuration Manager Proxy API

The first stage of creating a broker domain is to create the required components. The most basic configuration uses a broker and a Configuration Manager that share the same queue manager. This chapter details the step-by-step

instructions that are required to create a simple broker domain manually. This requires the manual creation of a number of prerequisite resources such as databases and WebSphere MQ resources.

Alternatively, it is possible to create a simple broker domain using the Default Configuration wizard, as discussed in "Creating the default configuration" on page 35. This wizard automatically creates all of the components and prerequisite resources, and is designed to enable users to use WebSphere Message Broker quickly, for example, to verify that installation was successful or to run the WebSphere Message Broker samples.

It is recommended that a manual broker domain configuration is created for purposes other than install verification or running the WebSphere Message Broker samples.

7.2.1 Resources required for a simple broker domain

For the most basic broker domain the following resources are required

► WebSphere MQ queue manager
► A TCP/IP listener created on the queue manager
► A database for the broker using one of the following supported databases:
  – Derby (Windows only)
  – DB2 Universal Database
  – Oracle
  – Sybase
  – SQL Server (Windows only)
► An Open Database Connectivity (ODBC) connection for the broker database
► A broker
► A Configuration Manager
► A domain connection

7.3 Steps for manually creating a simple broker domain

This section gives instructions for creating the resources and components required for a simple broker domain. This configuration is enhanced in the following sections to provide a more complex broker domain with multiple brokers.

7.3.1 WebSphere MQ resources

In the most basic broker domain configuration, the broker and the Configuration Manager share the same queue manager. This makes setting up WebSphere MQ simpler, as there is no requirement to create channels to handle

communication between multiple queue managers. However, in a production environment it is recommended that each component has its own queue manager defined for performance reasons.

There are a number of ways to create WebSphere MQ components, either on the command line using programs and utilities such as runmqsc, or using the WebSphere MQ Explorer graphical interface. For this simple broker domain configuration, the WebSphere MQ Explorer is used to demonstrate how to create the required WebSphere MQ resources. The WebSphere MQ Explorer must be installed in order to use the instructions provided in this chapter. Refer to "Installing WebSphere MQ" on page 31 for instructions for installing the WebSphere MQ Explorer as part of an Express install or install the WebSphere MQ Explorer and IBM WebSphere Eclipse Platform v3.0 from the Advanced Panel on the LaunchPad.

Note that the commands that are used to create a broker and Configuration Manager create a queue manager if it does not already exist, but they do not create a listener on the queue manager. A listener is required on the Configuration Manager's queue manager to create a broker domain. Also, these commands do not set up channels that are required for broker domains with multiple brokers, or where the components in the domain do not share the same queue manager.

A listener is required on the queue manager to monitor incoming network connections and enable communication between the Message Brokers Toolkit and the Configuration Manager. A unique port number, not in use by any other application, needs to be specified for the listener. The default port number for a queue manager in WebSphere MQ is 1414, and this port is in use if a default queue manager has been created on the machine, for example, if WebSphere MQ has been configured with the FirstSteps tool. The WebSphere Message Broker Default Configuration uses port 2414 for the listener on the default queue manager.

The exact range of port numbers that is available to the user is dependent upon the hardware and software environment of the system, and different port numbers may be in use. There are a variety of applications that can be used to view the port usage on the machine, such as the `netstat` command. Use the following command to display a list of active TCP connections and ports on a Windows machine:

```
netstat -a
```

The ports in use are displayed as numbers after the machine name and a colon under the Local Address column. When choosing a port for the queue manager, select a port that is not already in use.

An option is provided to create a listener at the same time as a queue manager using the Create Queue Manager wizard in the WebSphere MQ Explorer.

Creating a queue manager

A queue manager is required as the mechanism for transport between the broker, Configuration Manager, User Name Server, and the Message Brokers Toolkit. Use the following instructions to create a new queue manager using the WebSphere MQ Explorer.

1. Select **Start** → **Programs** → **IBM WebSphere MQ** → **WebSphere MQ Explorer** to open the WebSphere MQ Explorer.

2. In the WebSphere MQ Explorer - Navigator view, right-click **Queue Managers**.

3. Select **New** → **Queue Manager...**. Figure 7-1 shows step 1 of the Create Queue Manager wizard.
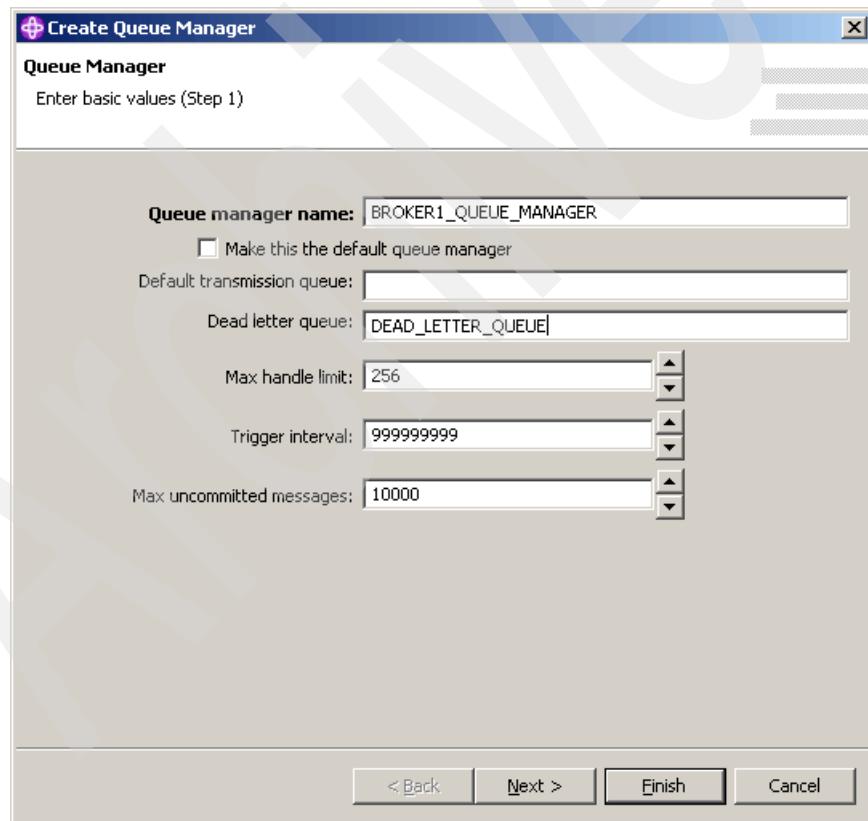
Figure 7-1   Create Queue Manager Wizard (Step 1)

4. Enter a name for the queue manager in the Queue manager name field, for example, BROKER1_QUEUE_MANAGER. Note that the name of the queue manager is case sensitive.

5. Add a name for a dead letter queue for the queue manager in the Dead letter queue field, for example, DEAD_LETTER_QUEUE.

6. Click **Next**, accepting the defaults until step 4, "Enter listener options," in the wizard.

   Step 4 of the Create Queue Manager wizard enables the creation of the listener.

7. At step 4 ensure that **Create listener configured for TCP/IP** is checked

8. Enter a value for the Listen on port number section. This needs to be a port that is not currently in use by another application. An example port number is 4444. This panel can be seen in Figure 7-2.
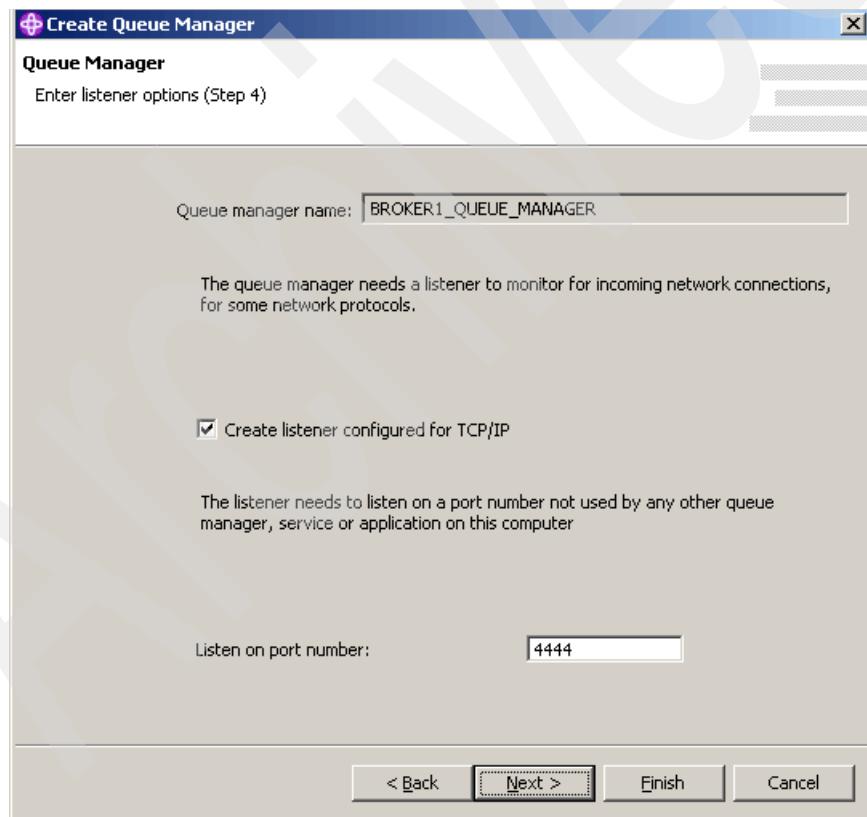
Figure 7-2   Create Queue Manager Wizard (Step 4)

9. Click **Finish**.

The new queue manager is displayed in the WebSphere MQ Explorer - Navigator view. The listener can be viewed by expanding the folders beneath the queue manager and selecting the listeners folder.

Creating a Configuration Manager

A queue manager has been created before the Configuration Manager in order to define a listener at the same time. Alternatively, the Configuration Manager could be created first and the listener added to the queue manager later.

All WebSphere Message Broker commands that have the 'mqsi' prefix need to be run in the WebSphere Message Broker Command Console on Windows. This is a command prompt that has been configured with the required environment for WebSphere Message Broker.

> **Note:** Names of WebSphere Message Broker components used in WebSphere Message Broker commands on Windows are not case sensitive. However, the flags used in the commands are case sensitive. For example, -n is not the same as -N. BIP8001 errors are generated if the wrong flags are specified on the command.
>
> The names of WebSphere MQ components are case sensitive. For example, the following commands create two different queue managers; even though they have the same name, they are a different case:
>
> ```
> mqsicreatebroker BROKER1 -i userid -a password -q QM1 -n BRKDB1
> mqsicreatebroker BROKER2 -i userid -a password -q qm1 -n BRKDB1
> ```

The Command Console can be accessed from the Start menu by selecting **Start** → **Programs** → **IBM WebSphere Message Brokers 6.0** → **Command Console**.

Multiple Configuration Managers can be created on a machine, but only one Configuration Manager exists in a broker domain. Follow the instructions below to create a Configuration Manager called CONFIG_MANAGER.

1. Enter the following command syntax in the Command Console to create the Configuration Manager:

   ```
   mqsicreateconfigmgr CONFIG_MANAGER -i userid -a password -q queuename
   ```

   The parameters required for the **mqsicreateconfigmgr** command are:

   – -i is a user ID (which must have Administrator authority and be a member of the mqbrkrs security group).

   – -a is the matching password.

– -q is the name of the queue manager, for example, BROKER1_QUEUE_MANAGER.

If the queue manager specified with the -q parameter does not already exist, it is created by the `mqsicreateconfigmgr` command. When the command has completed successfully a BIP8071 message is displayed in the Command Console.

If any errors occur during the creation of the Configuration Manager refer to **Chapter 8, "Troubleshooting and problem determination" on page 241**, for assistance with resolving the problem.

Creating a broker database

The next step in creating the broker domain is to create a database for the broker. This database is used to store configuration data for the broker. The broker database can be created in any of the supported broker database types—DB2 Universal Database,Oracle, Sybase, and SQL Server.

In addition, on Windows it is possible to use an embedded database known as Derby. This database is provided for verification, evaluation, and test purposes, and is not supported for use on production systems.

For Windows systems with DB2 Universal Database or Derby, two commands are provided to create and delete the databases for the broker.

For the other supported database types or other platforms, the databases must be created manually.

Use the instructions below to create a broker database on DB2 Universal Database or Derby.

1. Open the Command Console by selecting **Start → Programs → IBM WebSphere Message Brokers 6.0 → Command Console**.

2. At the command prompt type the following command to create a database for the broker called BRKDB1:

   ```
   mqsicreatedb BRKDB1 -i userid -a password -e dbtype
   ```

   The parameters required for this command are as follows:

   – -i is a user ID (which must have Administrator authority).

   – -a is the matching password.

   – -e is the database type. This parameter is optional and defaults to Derby if it is not specified. The options for the database type are 'DB2' or 'Derby'.

This initiates the creation of a new database called BRKDB1, which may take some time. After creating the database an Open Database Connectivity (ODBC)

connection is created for the database. The ODBC connection enables the broker to communicate with the database. Messages are displayed on the command prompt to show the status of the command.

Creating a broker

When the broker database has been created, the **mqsicreatebroker** command can be run in the Command Console to create a new broker. The **mqsicreatebroker** command has similar syntax to the **mqsicreateconfigmgr** command. Use the instructions below to create a broker called BROKER1. In the Command Console type the following command:

```
mqsicreatebroker BROKER1 -i userid -a password -q queuemanager -n dbname
```

The parameters required for this command are as follows:

- ► -i is a user ID (which must have Administrator authority).
- ► -a is the matching password.
- ► -q is the name of the queue manager, for example, BROKER1_QUEUE_MANAGER.
- ► -n is the name of the broker database, for example, BRKDB1.

For a simple configuration keep the queue manager used for the broker and Configuration Manager the same. When the same queue manager is used for both components no extra communications need to be set up.

When the command has completed a BIP8071 message is displayed in the Command Console if the creation of the new broker was successful.

If any errors occur during the creation of the broker refer to **Chapter 8, "Troubleshooting and problem determination" on page 241**, for assistance with resolving the problem.

Starting the components

When the broker and Configuration Manager have been successfully created they can be started. To check that the components were created successfully, type mqsilist in the Command Console. This displays a list of the components that have been created and the names of queue managers where appropriate. An example of the output from mqsilist is shown in Figure 7-3 on page 214.

Figure 7-3   List of components from mqsilist

Components are started using the `mqsistart` command in the Command Console. Use the following instructions to start the Configuration Manager and the broker.

1. Start the Configuration Manager by typing the `mqsistart` command in the Command Console followed by the Configuration Manager name, for example:

   mqsistart CONFIG_MANAGER

   A BIP8096 success message should be displayed after the command has been run.

   Verify that the Configuration Manager has started successfully and is available for use by checking the WebSphere Broker messages in the Windows Event Viewer generated by the WebSphere Message Broker runtime. The instructions for starting the Windows Event Viewer (Figure 7-4) vary between different versions of Windows, but typically it can be launched from the Administrative Tools section of the Control Panel, or in the Computer Management tool found in the Control Panel.
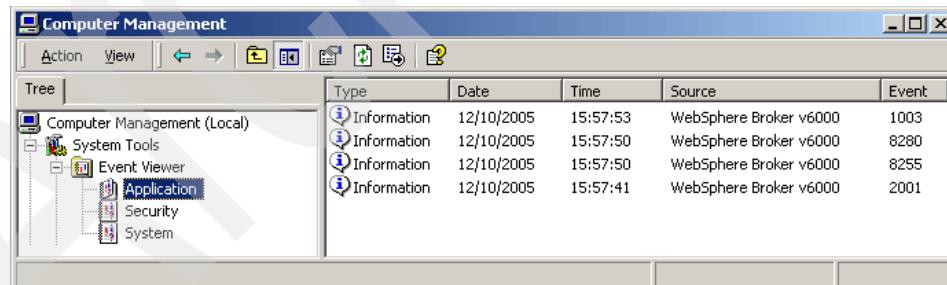
Figure 7-4   Windows Event Viewer

   The contents of each of the messages in the Windows Event Viewer can be viewed by double-clicking the message. A message with an Event ID of 1003 indicates that the Configuration Manager started correctly and is now available for use. An example of this message is shown in Figure 7-5 on page 215.
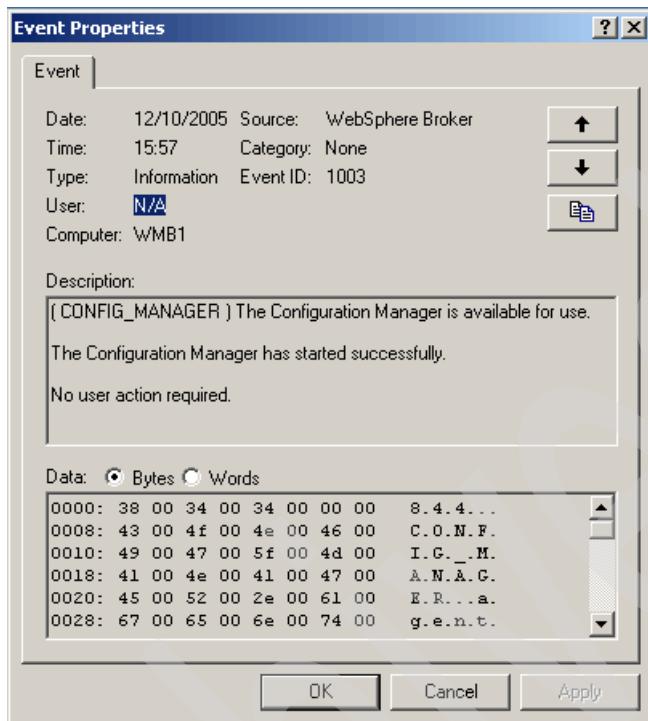
Figure 7-5   Configuration Manager available for use event message

If the message with an Event ID of 1003 is not present, the Configuration Manager is not available for use even if it may have started correctly. The Windows Event Viewer may display error messages that give diagnostic information to help to track down the cause of the problem. For more information about WebSphere Message Broker messages in the Event Viewer and diagnosing problems see "Windows Event Viewer" on page 253.

2. Start the broker by typing the `mqsistart` command into the Command Console followed by the broker name, for example:

```
mqsistart BROKER1
```

A BIP8096 success message is displayed after the command has been run.

3. Check the Windows Event Viewer for a WebSphere Broker error message with an Event ID of 2001 to confirm that the broker started correctly.

The broker and Configuration Manager must be started in order to create a broker domain. If for any reason the Configuration Manager or the broker failed to be created or to start, refer to **Chapter 8, "Troubleshooting and problem determination" on page 241**, for information about how to perform problem determination.

Creating a domain connection

This section assumes that the basic components of a broker domain, a Configuration Manager, and a broker have been created and are started on the system.

The domain connection is the reference to a broker domain in the Message Brokers Toolkit. Multiple broker domains can be defined in the Message Brokers Toolkit with connections to both local and remote Configuration Managers.

The following instructions demonstrate how to create a domain connection for a local Configuration Manager, but the same method can be used to create a domain connection for a remote Configuration Manager.

Use the instructions below to create a domain connection:

1. Launch the Message Brokers Toolkit.

2. Change to the Broker Administration perspective by clicking **Window** → **Open Perspective** → **Broker Administration perspective**.

3. From the File menu, select **New** → **Domain**. This opens the Create a Domain Connection dialog.

4. Enter the name of the Configuration Manager's queue manager, for example, BROKER1_QUEUE_MANAGER, in the Queue Manager Name field.

5. Change the port number to reflect the port that the queue manager's listener is configured on, for example, 4444.

6. Click **Next**.

At this point the Message Brokers Toolkit attempts to establish communication to the Configuration Manager. A status bar is displayed at the bottom of the wizard to show the progress, as shown in Figure 7-6 on page 217.
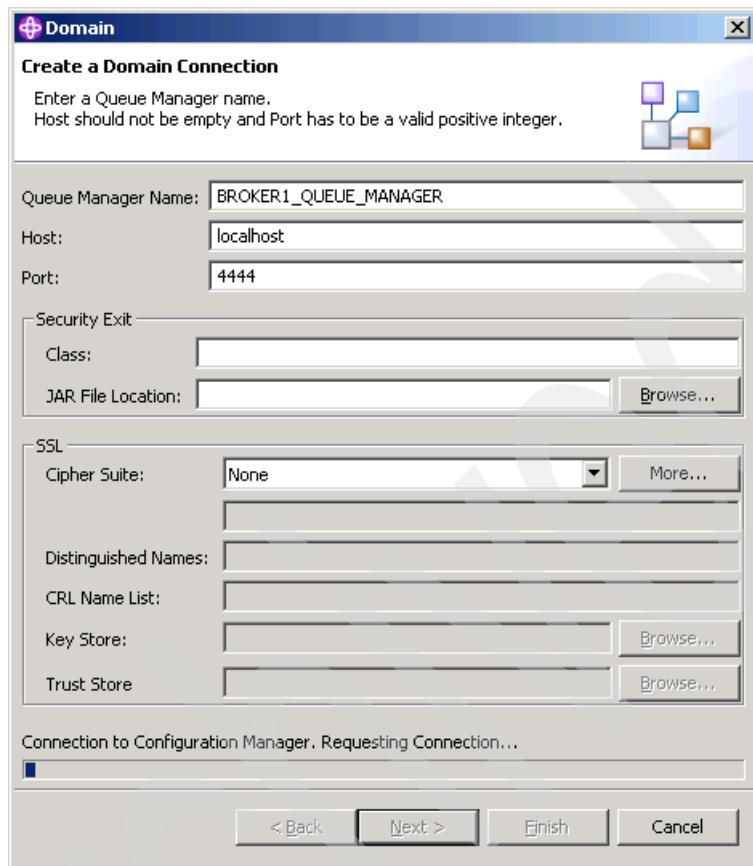
Figure 7-6   Creating a domain connection

7. Once a connection to the Configuration Manager is established enter a name for the Server Project, for example, LocalServerProject.

8. Enter a Connection name, for example, Connection1.

9. Click **Finish** to complete the domain connection.

The connection to the domain is displayed in the Domains view of the Broker Administration perspective, as shown in Figure 7-7 on page 218. In the Broker Administration Navigator view the Server project is displayed. This contains a configuration file that holds the domain connection information that has been specified in the Create a Domain Connection dialog.
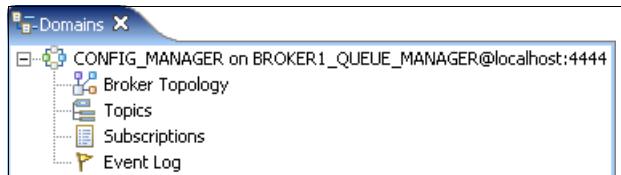
Figure 7-7   New domain connection displayed in Domains view

Adding a broker to the domain

The next step in creating a simple broker domain is to add the broker to the domain. Follow the instructions below to add a broker to a domain:

1. Select **New** → **Broker** from the File menu.

2. Select the domain to add the broker to if more than one exist.

3. Enter the broker name, for example, BROKER1.

4. Enter the queue manager name, for example, BROKER1_QUEUE_MANAGER.

5. Click **Finish**.

   A Topology Configuration Deploy message is displayed. This provides a choice of deployment operations to be performed. Figure 7-8 shows the three options to select from: Delta, Complete, or None.

Figure 7-8   Topology Configuration Deploy message

   A deploy is a configuration message sent to the components in the domain. This can be related to changes in the domain topology, such as adding or deleting brokers; or they can be changes to the configuration of message flow applications and resources.

   – *Delta* deploys only those parts of the configuration that have changed.

   – *Complete* deploys the entire configuration.

   – *None* results in no deploy being made at that time. The changes are not made in the broker domain, but are stored in the Message Brokers Toolkit.

With None, the next time a deploy is made from the Message Brokers Toolkit, these stored changes are deployed.

6. Click **Delta** in the Topology Configuration Deploy dialog box.

After initiating the deployment of a change in the topology to add a broker, a success message is displayed when the deployment is received by the Configuration Manager. The content of the success message can be read by clicking the **Details** button, as shown in Figure 7-9.
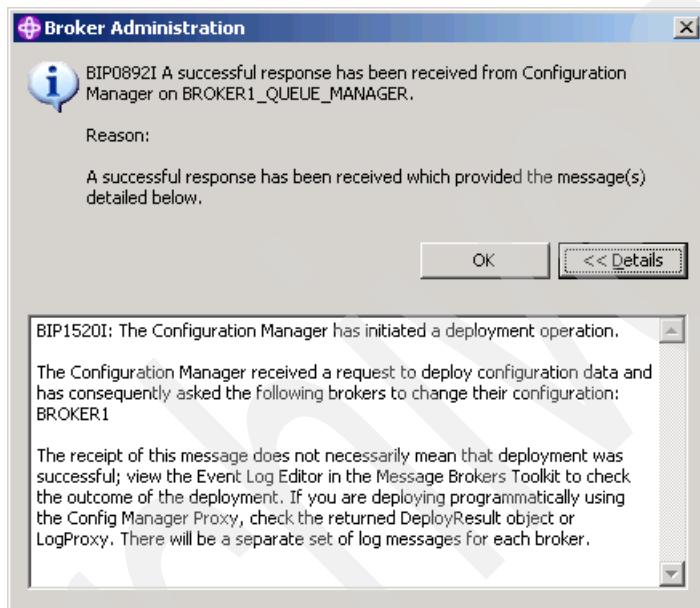
Figure 7-9   Deployment operation initiated message

The Event Log in the Message Brokers Toolkit can be used to check that the broker has been successfully added to the domain. This is different from the Windows Event Log with which it is sometimes confused, as the Event Log only shows messages from the Configuration Manager. The messages in the Event Log are used to determine whether the outcome of deployment operations, either for domain components or the deploy of bar files, is successful.

Open the Event Log by double-clicking the Event Log icon in the Domains view of the Broker Administration perspective, as shown in Figure 7-10 on page 220.
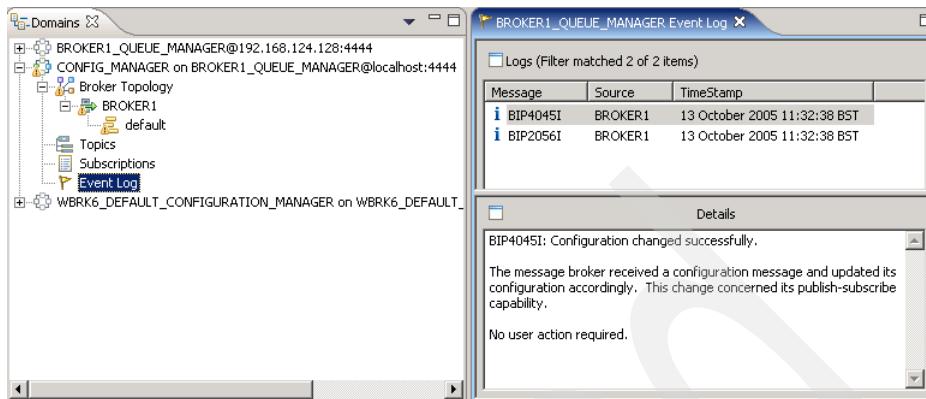
Figure 7-10   Event Log in the Message Brokers Toolkit

Messages indicating that the deployment to add the broker to the domain was successful are BIP4045 and BIP2056. The broker is also visible in the Domains view beneath Broker Topology.

When a broker is first added to the domain, a warning is displayed in the Alerts view indicating that the Execution Group is not running, as shown in Figure 7-11. It is normal to see this message before any message flows or message sets have been deployed to an execution group. The execution group is not started until resources are deployed to it.
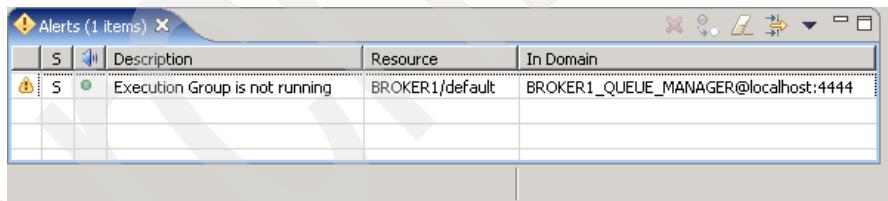
Figure 7-11   Execution Group is not running alert

The simple broker domain is now complete and the broker is ready for message flow applications and resources to be deployed to it. The following sections provide instructions on deploying message flow applications, publish/subscribe, and extending the broker domain by adding multiple brokers.

Administering components in a broker domain

This section provides a short introduction to administering the components in the broker domain. Adding additional components to the simple broker domain is discussed in "Extending a broker domain" on page 222.

Some administration beyond the deployment of message flow application resources can be carried out in the Message Brokers Toolkit. The Broker Topology can be administered from the Message Brokers Toolkit in particular for setup and configuration relating to publish/subscribe, which is discussed later in this chapter. Connections between brokers and some broker properties such as security and settings for multicast can be administered from the Message Brokers Toolkit. These are advanced topics and are beyond the scope of this book.

Administration on the components in a domain is mostly carried out using the WebSphere Message Broker command-line utilities, but the Configuration Manager Proxy API can also be used to automate administration tasks.

The main groups of commands that are used for administering components in the broker domain is listed below:

► Starting and stopping components
► Changing components
► Creating and deleting components
► Setting and retrieving trace on components
► Setting security
► Backing up

One of the most useful commands that is available is to view what components are created on a system. This command is `mqsilist`, and when used on its own displays all the components on the system (for a particular install if multiple versions of the product are installed). This command can also be used with broker names to view the running execution groups and deployed resources.

More information about the commands used to administer WebSphere Message Broker components can be found in **Reference** → **Operations** → **Commands** → **Runtime commands** in the WebSphere Message Broker product documentation.

Connecting the Message Brokers Toolkit to the domain

A final note in this section is how to reconnect to the domain after the Message Brokers Toolkit has been closed. When the Message Brokers Toolkit is closed, any domains that are disconnected are shown as grayed out in the Domains view of the Broker Administration perspective. To reconnect to the domain and view the status of brokers, execution groups, and message flows, right-click the domain connection in the Domains view of the Broker Administration perspective and click **Connect** in the context menu that is displayed.

7.4 Extending a broker domain

A simple broker domain configuration as created in the previous section of this chapter, or by the Default Configuration wizard, is only really useful for the development and testing of message flow applications. For any more advanced activity and in a production system the domain is more complex, for performance and high availability.

This section provides instructions on how to configure a broker domain with multiple brokers including components on a remote computer. This chapter also provides instructions to create a User Name Server, which is used for authentication with publish/subscribe applications.

7.4.1 Adding a remote broker to the domain

This section contains instructions on how to add remote brokers to an existing domain and then deploy a message flow to any broker within that domain. A broker can belong to only one domain, so you must create a new remote broker.

In these instructions, it is assumed that you have an existing broker domain with a single broker, a Configuration Manager, and a queue manager with a listener. The following new components are created:

► A broker, a queue manager, and a listener. These components are created manually on a different computer to the broker domain.

► Sender and receiver channels between the new and existing queue managers.

► Transmission queues for the new and existing queue managers.

Creating the required resources on the remote machine

The following instructions detail how to create the required resources for the broker on the remote machine:

1. Create a new queue manager as follows:

   a. Open the WebSphere MQ Explorer.

   b. Right-click **Queue Managers**, then click **New** → **Queue Manager**.

   c. Enter a queue manager name.

   d. Accept all of the default values until you reach step 4 of the wizard.

   e. In step 4, select **Create listener configured for TCP/IP**.

   f. Enter a port number for the listener. This must be a a port number that is not being used by anything else.

2. On the new queue manager, create a transmission queue.

a. Open WebSphere MQ Explorer.

b. For the queue manager that you have just created, right-click **Queues**, then click **New** → **Local Queue**.

c. Enter a queue name. Use the name of the queue manager in the existing broker domain.

d. In the General properties page, select **Transmission** from the Usage drop-down menu.

e. Click **Finish**.

3. On the new queue manager, create a sender channel to communicate with the queue manager that exists in the broker domain:

a. Open the WebSphere MQ Explorer.

b. For the new queue manager, expand the contents under **Advanced**.

c. Right-click **Channels** then click **New** → **Sender Channel**.

d. Enter a name for the sender channel. A suitable name should describe the relationship between the two queue managers, for example, QM2_to_QM1.

e. Click **Next**. This opens the New Sender Channel window (Figure 7-12 on page 224).

f. Enter a connection name. A connection name comprises the IP address or name of the machine on which the Configuration Manager exists, followed by the port number, in brackets, of the listener for the queue manager for the Configuration Manager, for example, wmbsystem(1414), or 9.1.12.123(1234), where *wmbsystem* is the name of the machine on which the broker domain exists.

g. Enter a transmission queue name. This should be the name of a transmission queue that is to be defined on the broker queue manager. This transmission queue is created in "Preparing the required resources in the broker domain" on page 224.

h. Click **Finish**.

Channel name:	QM2_to_QM1
Type:	Sender
Description:	
Transmission protocol:	TCP
Connection name:	zurich(1414)
Transmission queue:	WBRK6_DEFAULT_QUEUE_MANAGER
Local communication address:	

Figure 7-12   Creating a sender channel in WebSphere MQ Explorer

4. On the new queue manager, create a receiver channel to communicate with the queue manager, QM1:

   a. Open the WebSphere MQ Explorer.

   b. For the new queue manager, expand the **Advanced** folder.

   c. Right-click **Channels**, then click **New** → **Receiver Channel**.

   d. Enter a name for the receiver channel. A suitable name should describe the relationship between the two queue managers, for example, QM1_to_QM2.

   e. Click **Finish**.

5. Create the remote broker by using the `mqsicreate` command, and supply the name of the queue manager that was created in step 1.

Preparing the required resources in the broker domain

Before adding a new remote broker to the existing broker domain, follow the steps below on the same computer as the Configuration Manager:

1. Create a new transmission queue on the queue manager in the broker domain.

   Follow the instructions given in step 2 of the previous section, "Creating the required resources on the remote machine" on page 222. When asked to supply a queue name, provide the name of the transmission queue that was specified for the sender channel for the remote broker's queue manager.

2. Create a new sender channel for the queue manager in the broker domain.

   Follow the instructions given in step 3 of the previous section, "Creating the required resources on the remote machine" on page 222. When asked to

supply a transmission queue name, ensure that you specify the name of the transmission queue that was created on the remote machine.

3. Create a new receiver channel for the queue manager in the broker domain. Follow the instructions given in step 4 of the previous section, "Creating the required resources on the remote machine" on page 222.

Adding a remote broker to an existing broker domain

Follow the steps below to add the remote broker to the existing broker domain:

1. Start each of the sender channels that you created in the previous sections, one on the queue manager in the existing broker domain, and one on the queue manager on the remote machine. After a sender channel has started successfully, the receiver channels start automatically.

   To start a sender channel:

   a. Open the WebSphere MQ Explorer.

   b. Navigate to the sender channel on the appropriate queue manager.

   c. In the Content view, right-click the sender channel and click **Start**.

2. In the Message Brokers Toolkit, navigate to the Broker Administration perspective.

3. In the Domains view, right-click **Broker Topology**, then click **New** → **Broker**.

4. Enter the name of the remote broker.

5. Enter the queue manager that is associated with the remote broker.

A deploy message is sent to the remote broker to assign it to the domain. Check the Event Log in the Message Brokers Toolkit for success messages from the deployment.

The Broker Administration perspective might show the broker as stopped and the execution group not running until a message flow or message set is deployed to the default execution group on the broker. If the Event Log contains messages to indicate the broker was successfully added to the domain then the communication between the Configuration Manager and remote broker is running successfully.

Refer to "Remote broker not responding" on page 301 for help with troubleshooting the problem if success messages are not displayed in the Event Log.

7.4.2 Deploying resources to a remote broker

Deploying resources to a remote broker is exactly the same as for a local broker: Drag and drop a broker archive file onto an execution group, or select **Deploy File**.

Deploy-related event messages in the Event Log in the Message Brokers Toolkit shows the name of the broker that they apply to. To confirm that the resources within the bar file have been successfully deployed to a remote broker, in the Command Console or command line on the remote broker machine and type:

```
mqsilist brokerName -e executionGroupName
```

Using the `mqsilist` command with the name of the broker and the execution group that you have deployed to shows any message flows that are running in that execution group. Message flows are only displayed by `mqsilist` when they are deployed and running.

7.4.3 Creating a User Name Server

A User Name Server is used for authenticating users and groups for permissions in publishing and subscribing to applications through the broker. The User Name Server can be created on the command line using the following command, which is very similar to the syntax of the command to create a Configuration Manager:

```
mqsicreateusernameserver -i userid -a password -q queue_manager
```

If the User Name Server is created on the same machine as the Configuration Manager they can share the same queue manager for simplicity. If they are on different machines, or use different queue managers for performance reasons, channel connections are needed between the User Name Server and the Configuration Manager.

7.5 Deploying message flow applications

Message flow applications such as message flows and message sets are deployed to execution groups within brokers. These message flows and message sets cannot be directly deployed to the broker; they must first be compiled in a message broker archive file. The message broker archive file is then deployed to a specific execution group.

A message broker archive file is a compressed file that contains any number of compiled message flows and message sets, including compiled ESQL, mappings, Java, and other resources as required for the applications. A

message broker archive can also contain source files, enabling it to be used for back-up purposes.

The message broker archive files can be used as a mechanism for moving deployable artefacts from one machine to another. A message broker archive can also be used to configure alternative properties for message flows, so that the same flows can be deployed to different brokers or execution groups with different properties, for example, changing the data source from a test database to a production database, without altering the original flow.

For test and development purposes the main way of deploying the message broker archive to the broker is through the Message Brokers Toolkit. There are two additional methods for deploying a broker archive file to the broker that can be used to automate deploy operations:

► `mqsideploy` command-line utility
► Configuration Manager Proxy API

These two alternative methods are not described in any further detail here, as they are beyond the scope of this book. Further information about both of these tools can be found in the product documentation.

In order to deploy message flow applications successfully, both the Configuration Manager and the broker to which the resources need to be deployed must be running and part of the same domain. If they are using separate queue managers then any channels required for communication between the queue managers must be running.

7.5.1 Creating a message broker archive

A message broker archive is created in the Broker Administration perspective of the Message Brokers Toolkit. To create a new message broker archive:

1. Click **New** → **Message Broker Archive** from the **File** menu.
2. Select a server project from this list, for example, LocalServerProject.
3. Enter a name for the broker archive file in the File name field, for example, BrokerArchive1.
4. In the Broker Archive editor click the Add (🔲) button near the top of the editor.
5. In the Add to the Broker Archive dialog select the resources that you want to deploy. An example is shown in Figure 7-13 on page 228.
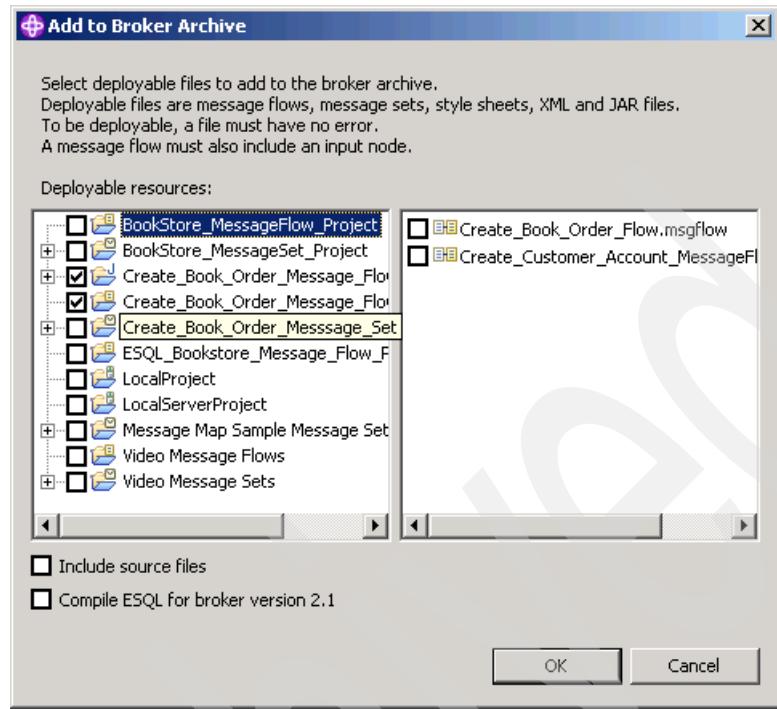
Figure 7-13   Add to Broker Archive dialog

6. Click **OK** to add the selected resources to the message broker archive file. This compiles the resources.

7. A status dialog is displayed to indicate the outcome of adding the resources to the broker archive.

8. Click **Details** to check that the resources were added successfully.

9. Click **OK**.

   If message flows or message sets contain errors, or errors exist in the associated projects, adding the files to the broker archive fails. Figure 7-14 on page 229 shows an example of the message that is displayed in the Details view.
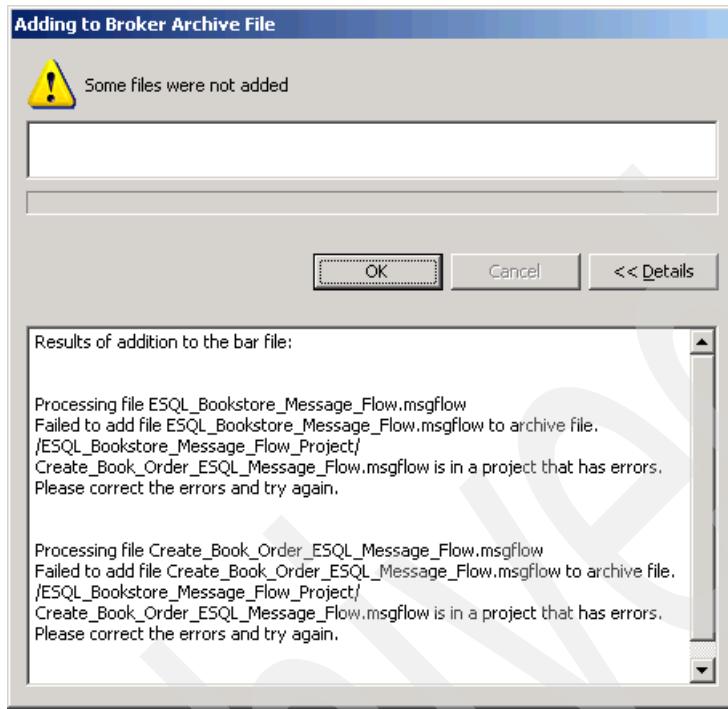
Figure 7-14 Error adding files to broker archive

10.Save the message broker archive file.

Deploying a message broker archive file

There are two ways to deploy a message broker archive file within the Message Brokers Toolkit:

▶ Drag and drop
▶ Deploy File

Both require an established connection to the Configuration Manager.

Deploying a message broker archive using drag and drop

This is the simplest method of deploying a message broker archive, but care must be taken to avoid inadvertently dropping the message broker archive into the wrong execution group. (This can be corrected in the Domains view by right-clicking the deployed resource and clicking **Delete** from the context menu.)

To perform a drag-and-drop deploy:

1. In the Broker Administration perspective, expand the **Broker Topology** in the Domains view to see the available brokers and execution groups.

2. In the Broker Administration Navigator view, click the message broker archive (.bar) file to deploy from the server project.

3. Hold the mouse button down over the message broker archive file and drag the file over the execution group where you wish it to be deployed.

4. Let the mouse button go over the execution group.

When the file is over an execution group the mouse pointer changes shape to a square. If the mouse pointer is shaped like a circle with a diagonal line through it, then the mouse is not positioned in the correct place for the deploy.

When the message broker archive file is dropped onto the execution group, the deploy is initiated and a configuration message is sent to the Configuration Manager. If the configuration message is received and processed successfully by the Configuration Manager a confirmation message is displayed in the Message Brokers Toolkit, as shown in Figure Figure 7-15.
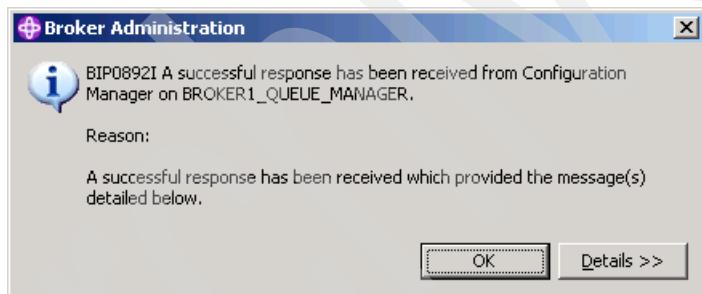
Figure 7-15   Success response from the Configuration Manager

The Configuration Manager passes the configuration message to the appropriate broker, which then responds after it has processed the configuration message and started the new or changed message flow application resources. The broker returns a success message or error message to the Configuration Manager, which is displayed in the Event Log in the Message Brokers Toolkit.

Check the Event Log in the Message Brokers Toolkit for the BIP2056 and BIP4040 success message to verify that the deployment operation was successful. If the deployment was unsuccessful the Event Log may contain error messages to indicate the cause of the failure.

Deploying a message broker archive using Deploy File

This method is slightly more complex than the drag-and-drop method, as it involves initiating the deploy using a menu and a dialog, but it is less prone to accidental deployment to the wrong execution group than the drag-and-drop method.

To deploy a message broker archive using Deploy File:

1. Click the message broker archive (.bar) file to deploy in the Broker Administration Navigator view.

2. Right-click the message broker archive file, then click **Deploy File**.

3. This opens the Deploy a BAR File dialog, which shows the available brokers and their execution groups in the broker domain, as shown in Figure 7-16.
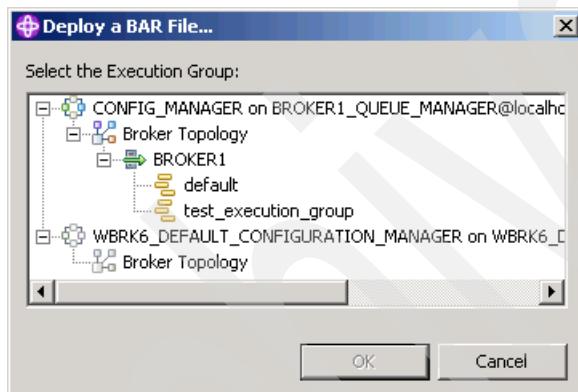
Figure 7-16   Deploy File

4. Select a target execution group and click **OK** to deploy the broker archive.

A confirmation message is received from the Configuration Manager if the deployment is successful. Check the Event Log in the Message Brokers Toolkit for the BIP2056 and BIP4040 success messages to verify that the deploy operation was successful.

Administering deployed resources

Using either the Message Brokers Toolkit or the WebSphere Message Broker command-line utilities, it is possible to administer deployed resources and the execution groups they are deployed to. The types of operation that can be performed on deployed resources and execution groups include stopping, starting, deleting, and removing.

In the Message Brokers Toolkit, right-click an execution group or deployed resource, such as message flows and message sets, in the Domains view. This displays a context menu with the application options. An example of this for an execution group is shown in Figure 7-17.
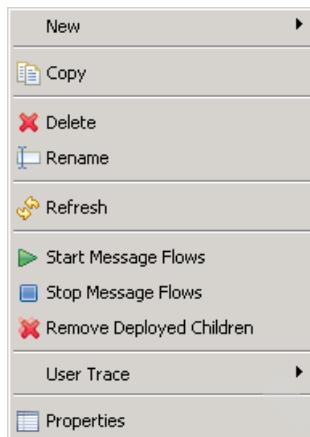
Figure 7-17   Context menu for an execution group

Information about the commands used to administer WebSphere Message Broker execution groups and deployed resources can be found in **Reference** → **Operations** → **Commands** → **Runtime** commands in the WebSphere Message Broker product documentation.

7.5.2 Message flow application resource versioning

When a message set or message flow is deployed to an execution group, the name of the message set or message flow is displayed either in the Message Brokers Toolkit or on the command line if `mqsilist` is used. If there is an update to the message set or message flow and it is redeployed it may not be obvious whether the update has been made by using the resource names alone.

When developing resources and running a production environment it is a good idea to have different broker archive files containing different versions of message flow application resources. This allows you to return to a working version of a message set or message flow if the latest version is discovered to have a problem or defect. It is useful within broker archive files to know which version of a message set or message flow was used to create the broker archive file.

There are a number of ways that the version of a message set and message flow can be determined in a broker archive file and when it is deployed to an

execution group. The versioning information that is gathered at deployment time is as follows:

▶ Date and time that the resource was deployed
▶ Date and time that the resource was compiled in the broker archive file
▶ Name of the broker archive file from which the resource was deployed
▶ The contents of a user-defined Version number
▶ Any other user defined keywords

The different methods for adding and viewing information about deployed message flow application resources are covered in the following sections.

Adding a version number to a message flow

To add a version number to a message flow:

1. Open a message flow in the message flow editor.

2. Right-click a blank area of the message flow canvas, then click **Properties**.

3. Enter a version number for the message flow next to Version in the Default Values for Message Flow Properties dialog, as shown in Figure 7-18.

4. Click **OK**.

5. Save the message flow.
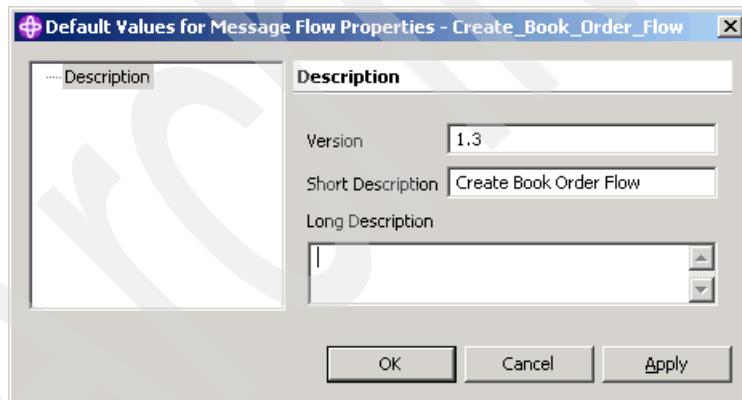
Figure 7-18   Adding a version number for a message flow

The version number must be manually updated when changes have been made to the message flow.

Adding a version number to a message set

A version number can also be added to a message set using the following instructions:

1. Open a message set by double-clicking the **.messageSet** file in the Resource Navigator view.

2. In the Message Set editor click **Documentation**.

3. Enter a version number in the Version field, as shown in Figure 7-19.

4. Save the message set.
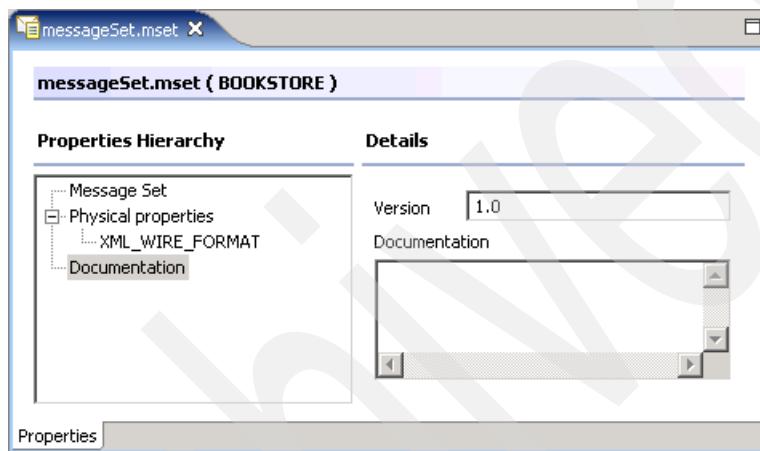
Figure 7-19   Adding a version number to a message set

The version number must be manually changed if updates are made to the message set.

Adding extra version information

It is also possible to add extra version information into message flows by adding keywords to resource files such as ESQL and Java. Keywords can also be added into message sets using the Documentation section of the message set's Properties dialog.

The way to do this is to add a string to these files with the format:

```
$MQSI keyword = value MQSI$
```

These keywords can be used to identify the author of the resource, tested by, machine name, or any other information that may be useful. The WebSphere Message Broker product documentation provides more information about how to set these keywords within the message flow application source files.

Viewing deployment and version information

The version information can be viewed for each compiled file in the broker archive file. Add message flow application resources with Version information contained in them to a message broker archive file.

To view version and keyword information for deployed resources, first deploy any message sets or message flows that contain version information. In the Broker Administration perspective, select one of these message sets or message flows from the Domains view. In the Properties view the deployment and version information are visible. This is useful to check the version number and also the time that the message sets or message flows were deployed. An example for a message set can be seen in Figure 7-20.
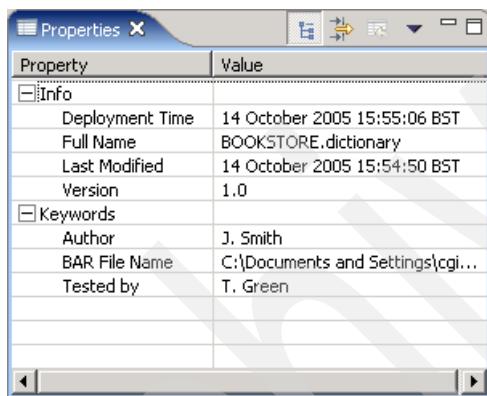
Figure 7-20   Deployment information, version, and keyword for a message set

7.6 Publish/subscribe

Publish/subscribe is a style of messaging in which messages that are produced by a single application can be received and utilized by many other applications. Publish/subscribe is a broad subject, and an in-depth discussion of its uses and configuration is beyond the scope of this book. The Broker Administration perspective in the Message Brokers Toolkit includes a number of tools to assist in the configuration and administration of a publish/subscribe broker network, so a brief description of the function of each tool is given here with a short overview of publish/subscribe concepts.

As an additional source of information, and to try out some examples of publish/subscribe using the Message Brokers Toolkit, try the Soccer and Scribble samples that are provided in the Samples Gallery in the help system, as well as the Pager samples. Note that the Pager samples and Scribble sample

are not supplied with WebSphere Event Broker, as they contain nodes that are not available, plus the Pager sample uses a message set.

7.6.1 Publish/subscribe basic concepts

An application known as a *publisher* generates a message that can be used by other receiving applications known as *subscribers*. The published message contains publish/subscribe commands and information in the message header. A published message is called a *publication*. There are two basic types of publish/subscribe commands contained in the header: One to register a subscription and the other to publish a message. Both messages show a message topic. A subscriber subscribes to messages of a particular topic using a subscription message, and a publisher publishes a message about a given topic.

This is an example from the SurfWatch sample of a message for registering a subscription:

```
<psc><Command>RegSub</Command><Topic>SouthShore</Topic><Topic>SunsetBeach</Topic><Topic>Rockpile</Topic><Topic>Kahaluu</Topic><RegOpt>PersAsPub</RegOpt><QName>PAGER</QName></psc>
```

This is an example from the SurfWatch sample of a publication:

```
<psc><Command>Publish</Command><Topic>Laniakea</Topic></psc><mcd><Msd>mrm</Msd><Set>GettingStartedMessageSets</Set><Type>Pager</Type><Fmt>XML</Fmt></mcd><Pager><Text>SurfWatch12.02.0420:57:18Laniakea: Onshore, waves 0m.</Text></Pager>
```

The various tools in the Broker Administration perspective can be used to set up collections of brokers for use in publish/subscribe networks and to manage topics and subscriptions as described in the following sections.

7.6.2 Broker topology

Many brokers can be added to a broker domain and can be configured for publish/subscribe through the Broker Topology editor in the Message Brokers Toolkit. Brokers that are added to the broker domain are visible in the Broker Topology editor. Properties for individual brokers can be edited here; these are advanced properties relating to publish/subscribe security, multicast, and so on.

Also in the Broker Topology editor, brokers can be connected in a *collective*, in which brokers are connected to enable publish/subscribe messages to flow between them. In order to combine two brokers into a collective, the queue managers must be connected for communication using channel sender and receiver pairs. This is in addition to any channels that connect the queue managers for the brokers to the Configuration Manager queue manager.

Setting up communication for multiple brokers is explained in "Adding a remote broker to the domain" on page 222, but remember the following rules:

► Create a sender channel on each queue manager with the same name as the receiver channel on the other queue manager.

► In the sender channel properties, make the Connection name the same as the machine to connect to. Name the listener on the queue manager in this format: MachineName(listener port), for example, BRK001(1728).

► Create a transmission queue with the same name as the queue manager to be connected, and set this as the Transmission queue value in the sender channel.

► Confirm that the channel communication starts successfully.

Brokers can be added to the broker domain through the Broker Administration perspective using the Domains view or the Broker Topology editor. Instructions are given below for adding a broker through the Broker Topology editor:

To add a broker to the Broker Topology:

1. Double-click **Broker Topology** in the Domains view to open the Broker Topology editor.

2. In the Broker Topology editor under Entity, click **Broker**.

3. Click the canvas of the Broker Topology editor to add a broker to the Broker Topology.

4. Right-click the new broker and click **Properties** to set the broker name and the Queue Manager name. Other properties for publish/subscribe and multicast can be set up here if required.

To connect two brokers in a collective:

1. In the palette of the Broker Topology editor, under Entity, click **Collective**.

2. Click the canvas of the Broker Topology editor to add a collective to the Broker Topology.

3. Click the Connection tool in the palette part of the Broker Topology editor. Use the Connection tool to connect two brokers to the collective by creating two connections in the same way that connections between message flow nodes are made in the Message Flow editor.

4. Save the editor contents (**File → Save**).

5. Select either **Delta** or **Complete** to deploy the configuration changes that have been made to the brokers when the Topology Configuration Deploy message is displayed.

Using a collective enables publish/subscribe messages to be communicated across brokers.

7.6.3 Topics

Topics that are defined in publication messages and in subscription messages control the routing of publish/subscribe messages. When a publication is created by an application, a message that passes through a publish/subscribe message flow is routed to applications that have subscribed to receive messages on that topic. You can set up definitions of topics and subtopics within the Message Brokers Toolkit, and these can be used to configure topic-based security to limit which applications can access which messages passing through the system.

In order to view users and groups for setting security on topics, the following tasks must have been performed:

► A User Name Server must be been set up (for simplicity using the same queue manager as the Configuration Manager in the broker domain).

► Channels between the User Name Server queue manager and any broker queue managers must be set up.

► Use the `mqsichangebroker` command to set the -j parameter. This indicates that publish/subscribe access control is to be used for the broker, and to set the queue manager for the User Name Server onto any brokers in the domain:

```
mqsichangebroker BROKER1 -j -s BROKER1_QUEUE_MANAGER
```

► Use the `mqsichangeconfigmgr` command to set the User Name Server queue manager on the Configuration Manager using the following format:

```
mqsichangeconfigmgr -s BROKER1_QUEUE_MANAGER
```

Create topics and assign permissions for the publishers and subscribers on those topics using the following instructions:

1. In the Broker Administration perspective, double-click **Topics** in the Domains view.

2. In the Topics Hierarchy editor that is displayed, right-click in the Topics section and select **Create Topic** from the context menu.

3. This opens the Topic wizard. Enter a name for the topic.

4. Click **Next**.

5. On the Principal Definition page define the security settings for users and groups using the options to **Deny** or **Allow** subscription or publication on this new topic. These can be added to and edited later.

6. Click **Finish** to complete the task.

Other topics and subtopics can be created in the same way. To create a subtopic, select the named topic rather than the Topic root on the first page of the Topic wizard.
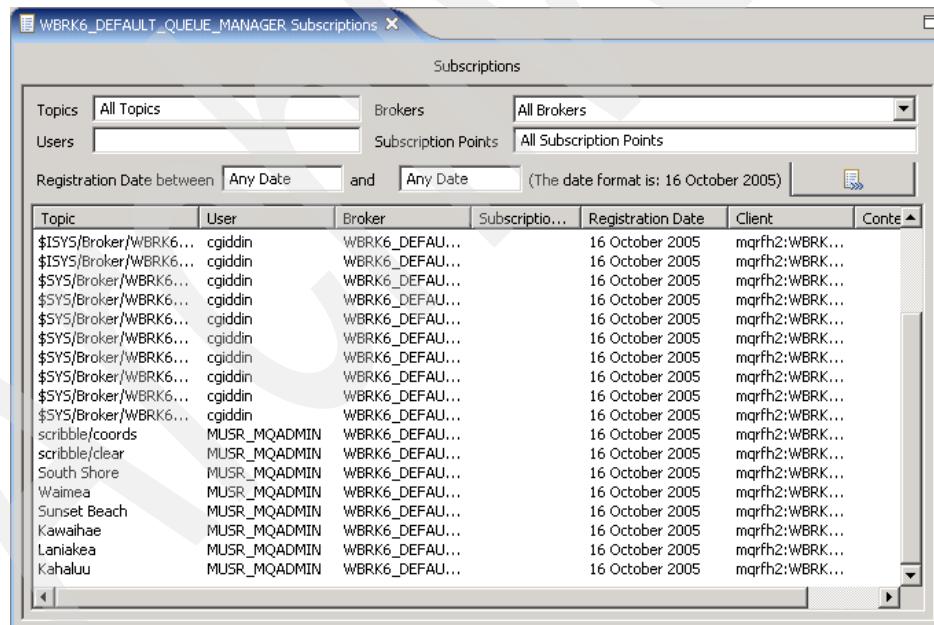
7.6.4 Subscriptions

When a subscriber application sends a subscription request to a broker on a particular topic, the broker stores the subscription data in a subscription table. The data in this subscription table can be accessed and searched using the Message Brokers Toolkit in the Subscriptions Query editor.

To show all subscriptions that are registered in the subscription table:

1. In the Broker Administration perspective, double-click **Subscriptions** in the Domains view.

2. In the Subscriptions Query editor, click the Query button (   ).

This displays all of the subscriptions that are registered with all of the brokers that are set up for publish/subscribe, as shown in Figure 7-21.

Figure 7-21   Subscription Query editor

These registered subscriptions can be queried on broker, topic, users, dates, and subscription points. Subscriptions do not remain in the subscription table indefinitely, because they can be de-registered by the subscriber, expired, or

deleted. They are also removed from the table when a subscriber application stops.

The Subscription Query editor can be used to delete subscriptions from brokers:

1. Right-click the subscription that you want to delete and click **Delete**.
2. Click **OK** in the confirmation dialog.

A configuration change is sent to the broker via the Configuration Manager to remove the subscription from the subscription table. Check the Event Log in the Broker Administration perspective for success or failure messages.

Those subscriptions with a topic name prefixed with a dollar symbol are internal subscriptions created and used by brokers. The Message Brokers Toolkit allows you to delete these subscriptions as with any user-created subscription. However, it is not advisable to delete these subscriptions because unpredictable results can occur.

Refer to the WebSphere Message Broker documentation in the product documentation for more information about publish/subscribe topics and subscriptions.

8

Troubleshooting and problem determination

This chapter provides assistance with determining the cause and resolution of problems when using WebSphere Message Brokers. The following topics are discussed in this chapter:

► Locating error information
► Using the Flow Debugger
► Using trace
► Troubleshooting common problems

8.1 Locating error information

This section details the location where errors generated by WebSphere Message Broker may be found. There are many places that useful information can be found to help with problem determination; some information can be found within the Message Brokers Toolkit, but other information can be found using operating system tools and within the directory structure of the system that is running the product.

8.1.1 Event messages

Event messages showing errors are usually the first indication that a problem may exist and must be resolved in some part of the WebSphere Message Broker's configuration or deployed applications. These messages are displayed in the Message Brokers Toolkit, in the operating system logs, or on the command line in response to a command or a command-line utility. Information about the different types of event messages that may be seen in the Message Brokers Toolkit, in the local system logs, and on the command line are detailed here with guidance for interpreting them.

Event Message structure

Event messages generated by the WebSphere Message Broker have the following structure:

```
BIP8081E
```

► `BIP` is the three-character product family identity, the same as in previous versions of the product.

► The four-character number following BIP is the unique identifier of the message. This indicates the condition that generated the message in the product and can be looked up in the product help or message catalog for further information. (Details of how to find more information about messages is given later in this section.)

► The final character indicates the event message type: `I` for an Information message, `W` for a Warning message, or `E` for an Error message.

An event message produced by WebSphere Message Broker having this structure is known as a BIP message. The BIP message code is usually accompanied by a description of the condition that generated the message, and sometimes a reason for the condition or a suggestion on how to fix it.

Event message types

Event messages generated by WebSphere Message Broker may be classified as one of three types:

- ► Information
- ► Warning
- ► Error

Information messages

Information messages simply provide information to the user and do not represent any problem of concern. Success conditions are classed as information messages; for example, BIP8096I is a message indicating successful command initiation for a `mqsistart` on the command line, and BIP0892I is the message for a successful response being received from the Configuration Manager in the Message Brokers Toolkit, as in Figure 8-1.
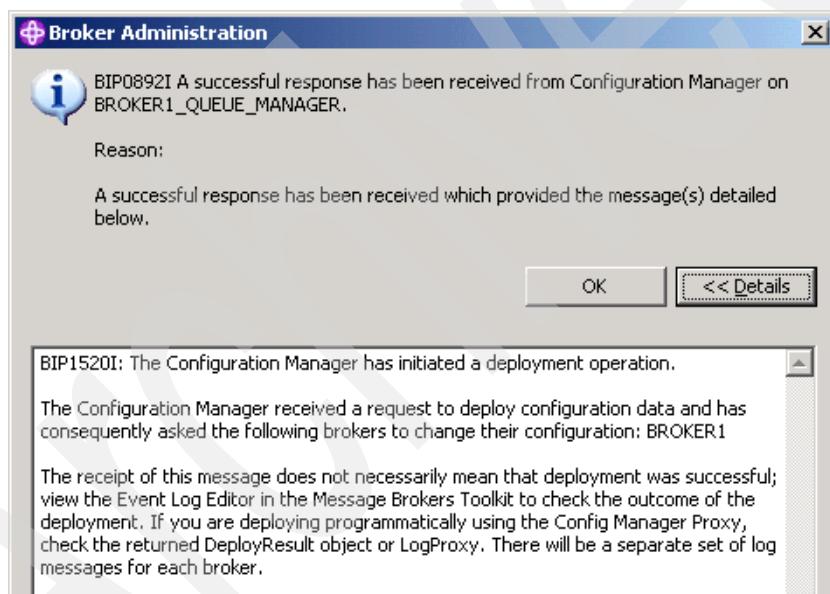
Figure 8-1   Pop-up message from the Message Brokers Toolkit

Warning messages

Warning messages are less severe than error messages and do not represent an immediate problem, but they indicate situations that may need investigation. These messages are less likely to be seen on the command line or as pop-ups in the Message Brokers Toolkit, and are more likely seen in the system log of the machine where the warning occurred.

Warning messages without a BIP code are generated by the Message Brokers Toolkit and can be seen in the Problems view and in the Alerts view in the Broker Administration perspective of the Message Brokers Toolkit. These views are described later in this chapter.

Error messages

The most severe and important messages are the error messages that are generated by WebSphere Message Broker. These can be seen on the command line, in the Message Brokers Toolkit, and in the system log of the machine where the product is running. These messages are either generated immediately in response to a failed action, such as trying to start a broker that does not exist, or as a response to a failure during the running of the product. Messages seen on the command line or as pop-ups in the Message Brokers Toolkit are failure responses to an action from the user.

Messages that are found in the system log or in the Message Brokers Toolkit Event Log usually occur report events that occur in running components in the WebSphere Message Broker runtime. An example would be an error message produced when a message flow attempts to access a WebSphere MQ queue that does not exist. These errors do not occur as a result of a user action.

8.1.2 Messages within the Message Brokers Toolkit

Messages displayed within the Message Brokers Toolkit are of two basic types: Pop-up messages that are generated as instant feedback to an action; and messages that result from the running of the product and are displayed in a log or a view, such as the results of a deploy operation in the Event Log or an error produced when a message flow containing an error is saved.

The following locations where messages are displayed in the Message Brokers Toolkit are discussed in this section:

► Pop-up messages
► Problems view
► Alerts view
► Message Brokers Toolkit Event Log

The most important of these is the Message Brokers Toolkit Event Log, as this is where the results of administration operations on the broker domain are displayed.

Pop-up messages

An example of a pop-up message is shown in Figure 8-1 on page 243, which shows a success message that was received from the Configuration Manager after a deploy action. This figure also shows the typical structure of a BIP

message with the BIP code followed by a description and then a Reason section that details why the message is being displayed. There is also a Details section that can be made visible or hidden by clicking the **Details** button. This gives more detailed information about the message, and in an error message this contains suggestions about how to correct the problem.

Pop-up messages are also displayed in the Message Brokers Toolkit that are not BIP messages, and frequently are some kind of progress or status indicator, such as when messages have been added to a broker archive file. Figure 8-2 shows a progress message that is displayed when the Message Brokers Toolkit connects to the Configuration Manager.
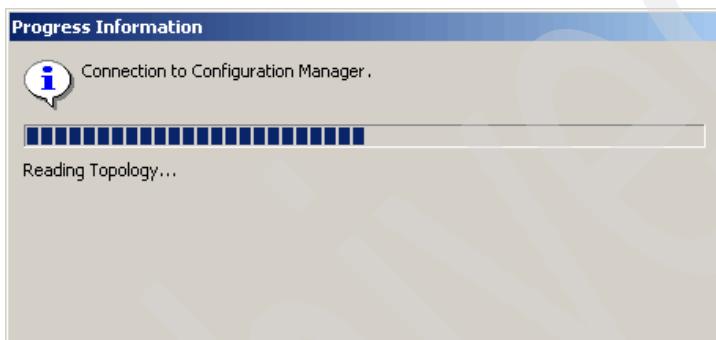
Figure 8-2   Progress message connecting to a Configuration Manager

Figure 8-3 on page 246 shows an example of a status message displayed after adding files to a broker archive file. The Details section shows the status after the addition of some message flows and message sets to a broker archive file. If any problems occurred while adding the resources, those errors or warnings are displayed here.
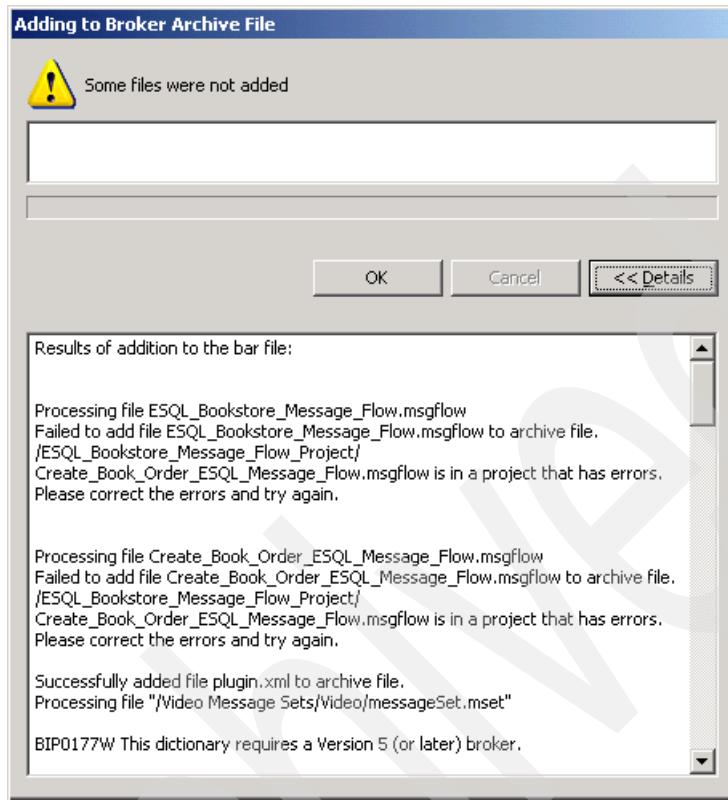
Figure 8-3   Warning adding resources to the broker archive file
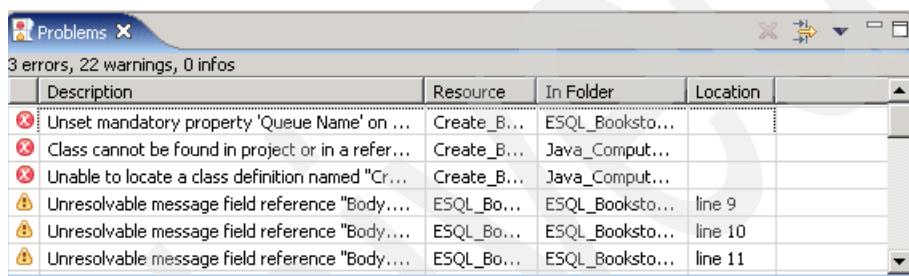
In Figure 8-3, two messages are displayed that indicate potential problems with the files chosen to add to the broker archive file. One of the message flows that has been chosen to be added to the broker archive file is in a project that contains errors. The message indicates that the message flow was not added to the broker archive file, and advises the user to correct the errors in the project before trying again.

The other problem displayed in this status message is with a message set that was added to the broker archive file containing namespaces. In this situation the warning message BIP0177W is displayed to indicate that the message set is incompatible with brokers that are running on previous versions of the product. This is a good example of the difference between errors and warnings. In this case the message set is valid so there is no error, but if the message set is deployed to a WebSphere MQ Integrator V2.1 broker, then errors occur.

Problems view

The Problems view in the Broker Application Development perspective shows information, warning, and error messages relating to the message flow application development resources such as message definition files, ESQL files, mapping files, Java, and message flows. Whenever one of these resources is saved, the resource is compiled and the contents are checked for errors. Any errors or warnings that are found are displayed in the Problems view.

The information that is associated with these problems is displayed with an icon indicating the severity level: Information, Warning, and Error. There are also priority icons so that the most significant errors can be highlighted. Figure 8-4 shows an example of messages displayed in the Problems view.

Figure 8-4   Errors and warnings in the Problems view

The top three messages that are displayed in Figure 8-4 are error messages. One of the message flows in the workspace is missing a queue name on the MQInput node; the other two relate to Java in a message flow. The rest of the visible messages are warning messages that relate to unresolvable message references. These do not cause any problems with deploying the message flows they belong to, as the message used for the message flow is self-defining. If a message is put to the message flow that does not match the structure expected by the ESQL then errors are generated later by the runtime.

The messages in the Problems view can be sorted by clicking the column headings. For example, to sort these messages by severity, click the column heading where the severity icons are displayed (left-most column). To sort by file name, click the column heading labeled Resource.

The messages can be filtered using the Filters dialog, which is launched by clicking the Filters button in the top right of the Problems view (the button with three arrows, shown in Figure 8-4) to launch this dialog. The example in Figure 8-5 on page 248 shows how to hide all messages containing the word *unresolvable* to get rid of the problem seen in Figure 8-4. This shows how the Filters dialog can be especially useful when a lot of similar messages are displayed in the Problems view.
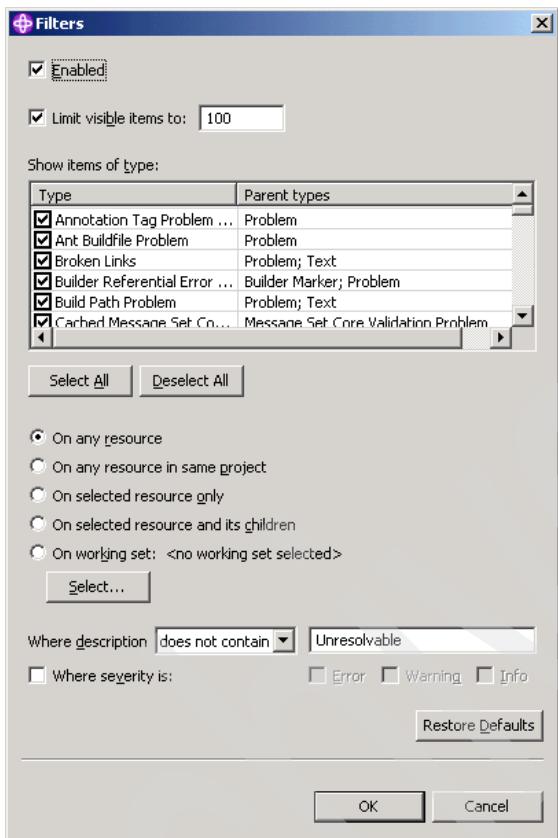
Figure 8-5   Filter for the Problems view

Alerts view

The Alerts view in the Broker Administration perspective looks similar to the Problems view in the way that it displays messages. However, the messages that it displays are the statuses of runtime components within a broker domain. This information is refreshed regularly so that if connections to the brokers in the domain are running correctly, then any changes to the brokers or execution group states are displayed. Figure 8-6 on page 249 shows an example of the messages that are displayed in the Alerts view.

Figure 8-6   Messages in the Alerts view

The Alerts view shows that a deploy operation is in progress (waiting for an answer from Configuration Manager), BROKER1 is not running, and neither are the two execution groups and four message flows that are deployed to it. These messages are displayed because the broker is stopped. If a component is stopped unexpectedly then this can alert you to a problem.

Running components are not displayed here, as they are displayed in the Domains view. Messages in the Alerts view cannot be sorted in the same way as the messages in the Problem view, but they can be filtered. The Alerts Filter dialog enables the Alert sources to be selected. The Alerts Filter dialog can be displayed by clicking the Filter button, which is the button with the three arrows that can be seen in Figure 8-6.

Alert sources such as individual brokers and execution groups can be deselected so that no alerts relating to these are displayed in the Alerts view. This is useful if the Message Brokers Toolkit is connected to more than one broker domain, the Filter Alerts can be used to view alerts from just one domain at a time, as shown in Figure 8-7 on page 250.
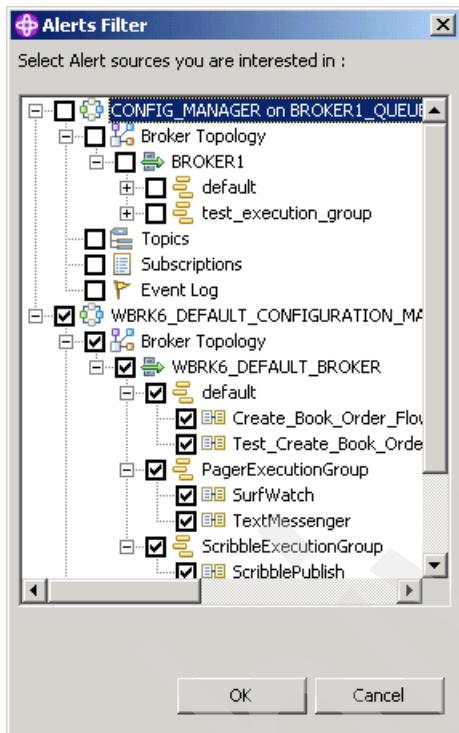
Figure 8-7   Hidings alerts from a broker domain

8.1.3 Message Brokers Toolkit Event Log

The Event Log editor in the Message Brokers Toolkit is the primary source of messages relating to deployment actions from the Message Brokers Toolkit to the Configuration Manager and the Broker Topology. The messages that are displayed in the Event Log are not recorded in the Windows Event Viewer, and the two logs must not be confused.

Whenever a deployment action is initiated through the Message Brokers Toolkit, the responses that are received are stored in the Configuration Manager's internal repository. These messages are then displayed in the Event Log editor in the Message Brokers Toolkit.

To access the Event Log, double-click **Event Log** (marked with a flag icon) in the Domains view of the Broker Administration perspective in the Message Brokers Toolkit.

When a message displayed in the event log is selected, its contents are displayed in the Details section beneath. These are BIP messages with the same

structure as those seen in the Windows Application log and on the command line.

A deploy response message is seen for each broker that is involved in a deploy action in the Event Log, and the name of the broker is seen in the Source section under Logs. Figure 8-8 shows successful deploy messages for the broker BROKER1.
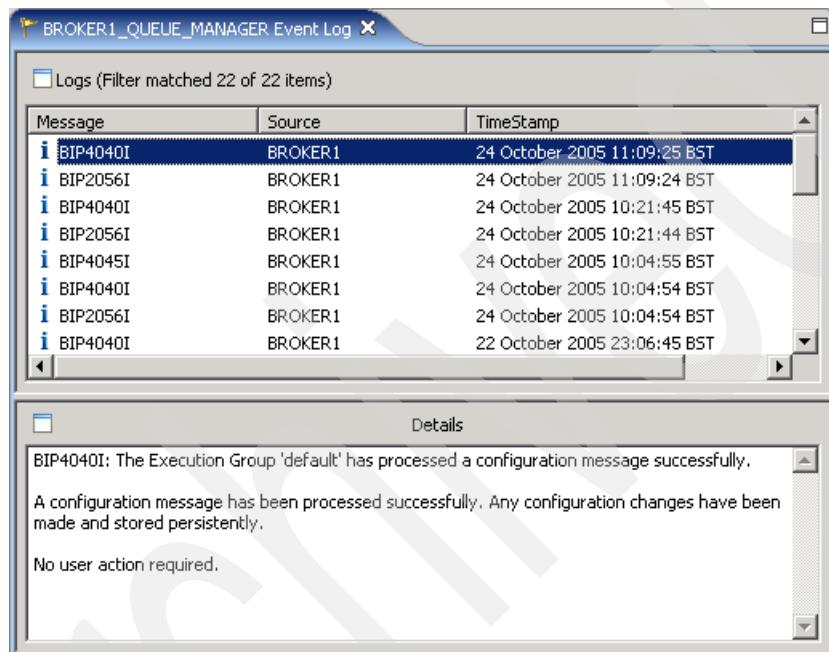
Figure 8-8   Successful deploy message

When an error message is displayed in the Message Brokers Toolkit Event Log, it may be necessary to look for other, related error messages in the Windows Event Viewer or a remote systems local log to determine the cause of the failure.

The Event Log updates itself automatically as messages, but right-clicking in the Event Log and selecting Revert refreshes the information in the log. Using this context menu, you also can filter and clear the log. Using Clear removes the messages from the Configuration Manager's repository, so this must be used only if the messages do not have to be kept. When messages are present in the log, then the log can also be saved from the context menu as text files.

8.1.4 Messages on the command line

Messages are displayed on the command line in the same format as the BIP messages are displayed in the Message Broker Toolkit when commands have been run. Figure 8-9 shows an example of a typical command line message. This is a BIP8018 generated by attempting to start a broker that is already started. The BIP code is displayed, followed by the description in the first sentence. The rest of the message gives the reason and a suggestion for solving the problem.

Figure 8-9   A BIP message displayed on the command line

For many of the WebSphere Message Broker commands, syntax assistance is displayed in addition to the message if the parameters that are used in the command are incorrect. This is a useful way to determine the syntax that must be used with these commands. For example, typing `mqsistop` on a command line results in the response seen in Figure 8-10 on page 253. This shows the syntax of the command and specific details as to the meanings or requirements of the parameters, and these are followed by the BIP8007E error message to indicate that a mandatory argument is missing.

```
IBM WebSphere Message Brokers 6.0                            _|□|×|

D:\MQSI\6.0>mqsistop
BIP8117W: Stops a component.

Syntax:
 mqsistop componentName [-q] [-i]

Command Options:
 'componentName' broker name, Configuration Manager name or
'UserNameServer'
 '-q' stops the WebSphere MQ queue manager used by the compo
nent
 '-i' forces the broker to stop immediately (use with cautio
n).

BIP8007E: Mandatory argument missing.
When using this command interface the user should supply the
 mandatory argument.
Correct and reissue the command.

D:\MQSI\6.0>
```

Figure 8-10   Syntax help for the mqsistop command

8.1.5 Windows Event Viewer
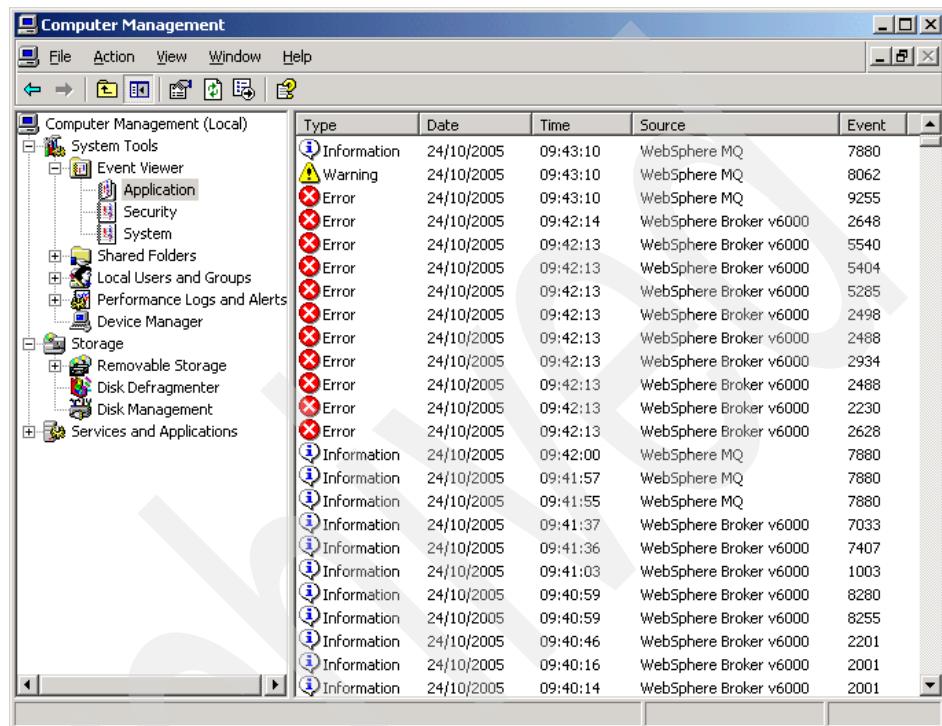
Messages that are generated by the WebSphere Message Brokers runtime are recorded in the local error log. On Windows this is the displayed in the Windows Event Viewer, on Unix it is the syslog, and on z/OS it is the operators console. These messages are all BIP messages that include information and warning messages as well as error messages. Where an error condition has occurred there may be multiple messages to describe the problem, generated by different components or parts of the runtime. In this section, information is provided on locating and viewing messages using the Windows Event Log.

The exact location of the Windows Event Log may depend on the version of Windows that is installed, but typically it is found under Administrative Tools in the Windows Control Panel. It can also be accessed through the Computer Management option (right-click **My Computer** → **Manage** → **System Tools** → **Event Viewer**).

Windows Application log

In the Windows Event Viewer, the BIP runtime messages from WebSphere Message Broker are recorded in the Application log. Figure 8-11 on page 254 shows the Application log through the Computer Management Tool in Windows. Messages that are produced by the runtime have a source of WebSphere Broker v6000, but messages generated by other programs such as WebSphere MQ and DB2 Universal Database can also be found in the Application log.

When the version of WebSphere Message Broker is updated, a corresponding change is seen in the source name to indicate which version the message came from. This is because multiple versions can be installed on the same machine, and therefore both versions may generate messages in the Application log.

Figure 8-11   Computer Management: Application log

The BIP code is shown in the Event column of the Application log (for example, the BIP2648E message, which is an error type).

To view more details about any message in the Windows Event Viewer, double-click the message to open it in a window called Event Properties, where message details can be seen. An example of the properties of a message from the Application log is shown in Figure 8-12 on page 255. This message shows that Configuration Manager has become available for use after starting.
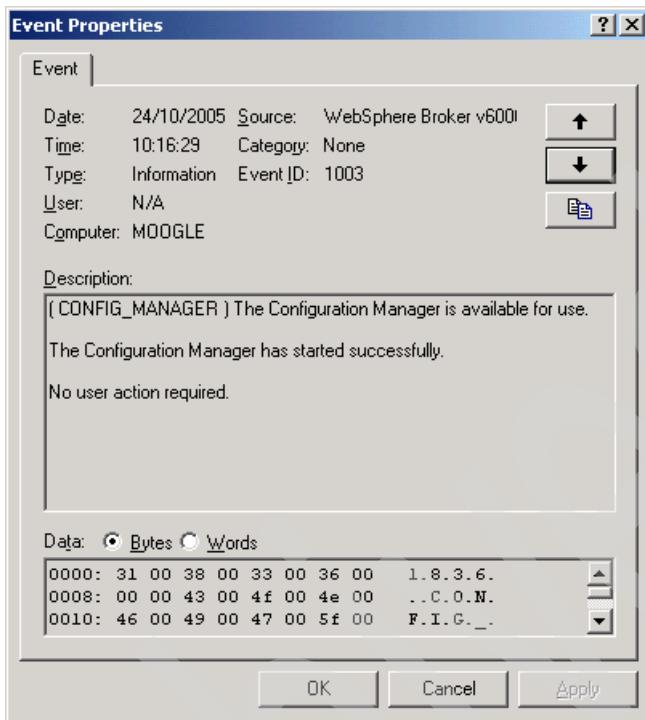
Figure 8-12   Example Application log message properties

When the WebSphere Message Broker runtime sends error messages to the Application log, there is usually a set of messages to indicate what problems have occurred for the messages to be thrown. Figure 8-11 on page 254 shows a set of error messages generated when a problem has occurred with the format of a message passed through a message flow.

There is often a sequence of messages that can be followed to determine the cause of the error. The first error generated is often a BIP2628 message indicating that an exception has been generated on an input node; an example of this type of message is shown in Figure 8-13 on page 256.

The reason why the first message is often an exception condition on the input node is that if an error occurs in a message flow, the input message is passed back up the flow, node by node, until it comes across a node with either a Failure or a Catch terminal connected. If a Failure or a Catch terminal is connected then the flow itself handles the error, and the same level of error message is not seen in the Application log or system log.

> **Tip:** When developing message flows, disconnect nodes connected to Failure and Catch terminals. When an error occurs the message flows back up the message flow, and errors are output in the Application log. If the Failure and Catch terminals are connected, then the flow handles the errors, and therefore does not output error messages to the Application log.
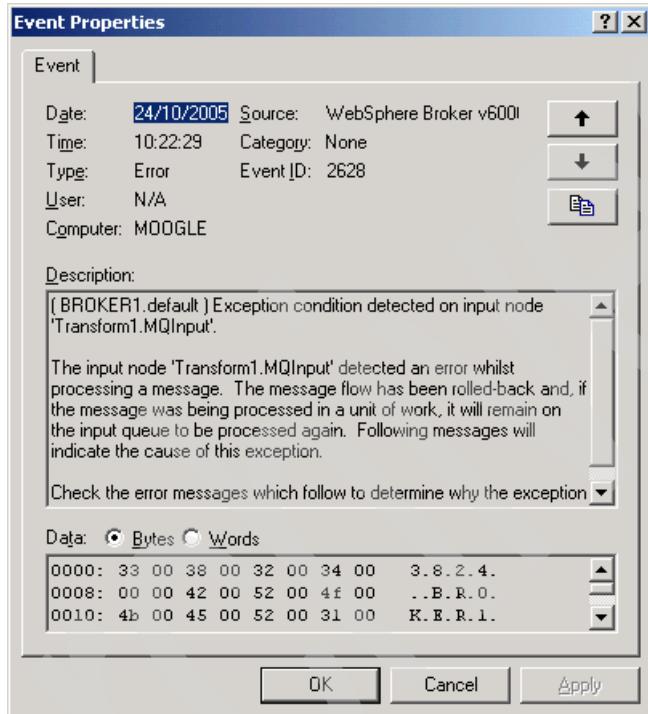
Figure 8-13   Example of an error of the input node

When the message in the message flow gets passed back to the input node, there is nowhere left for the message to go so it is passed back to the input source such as a message queue. Messages passed back to the input source are known as *backed out* messages.

> **Important:** If the message source is a WebSphere MQ queue it is blocked if no backout requeue queue is defined. "Configuring the ESQL_Simple message flow" on page 58 contains instructions for assigning a backout queue.

Further messages in the sequence give information about the node that the exception occurred in, and what the cause of the error may be. If, for example, a message has been received in a node that fails to match the format expected by the node, then the error shown in Figure 8-14 might be generated.
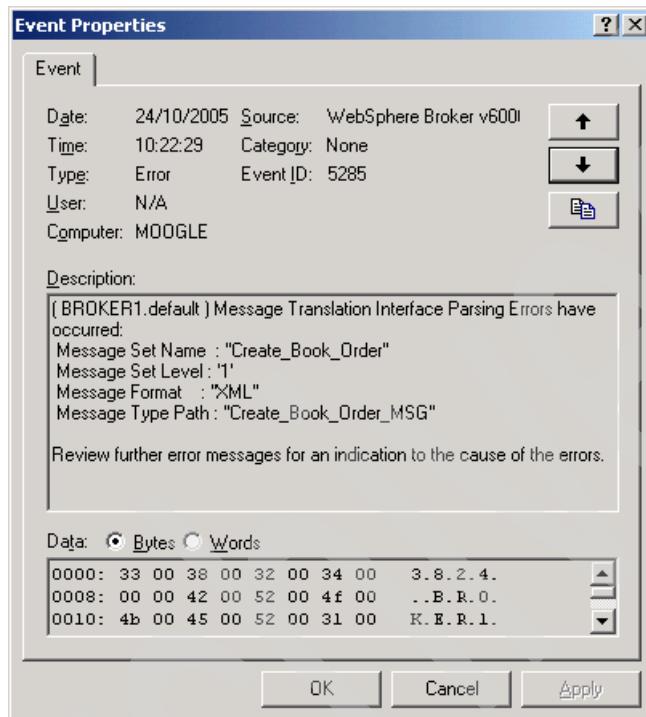
Figure 8-14   Parsing error message from the Application log

The errors that follow in the Application log give specific details as to which fields in the message did not match the expected format. Often the most useful message to indicate a problem is the second-to-last error message generated in a sequence of errors. Figure 8-15 on page 258 provides an example of this. The error here shows that the date and time in the input message do not match the expected format for the field. The expected format is yyyy-MM-dd HH:mm:ss, while the input message contains data for this field as 12:55:12 2005-09-27, which does not match this format. The action to resolve this problem is to alter the input message to supply the date and time in the expected format, or to alter the message set to match the format of the input message.
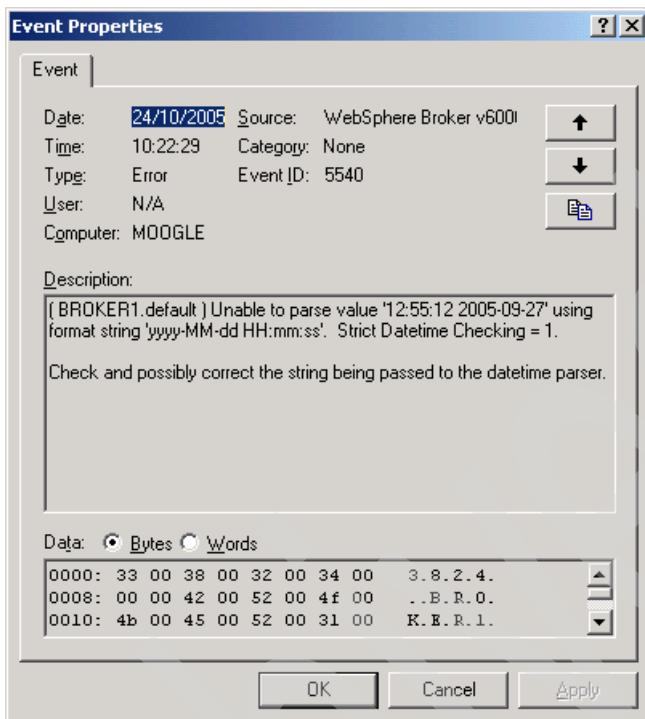
Figure 8-15 Example error message when message format is unexpected

Windows System Log

Information about the starting, running, and stopping of WebSphere Message
Broker components is displayed in the System Log in the Windows Event
Viewer. This is because the components such as brokers, the Configuration
Manager, and User Name Server are created as Windows Services. When
commands are sent to these components to start or stop them, then messages
are written to the System Log to indicate the status of the command. Figure 8-16
on page 259 shows an example of a message from the System Log that
indicates that a start message has been sent to a broker called BROKER1.
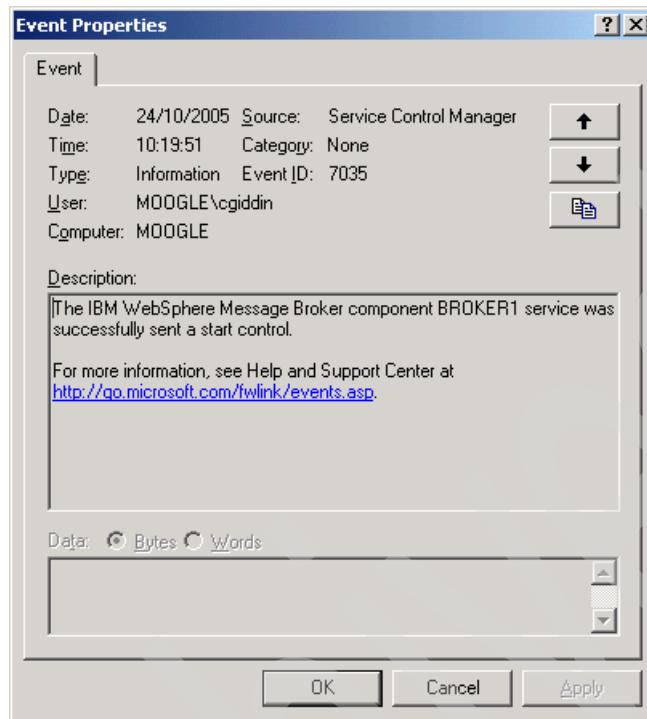
Figure 8-16   System Log message response to mqsistart broker command

Windows Event Viewer Log Properties

It is important to ensure that the properties for the Application log and System log have been set up correctly in the Windows Event Viewer; otherwise, messages could fail to be recorded, which may lead to difficulties in troubleshooting problems in the runtime. The Application log in particular can fill up very quickly if there are problems in the system, and either become full (in which case no new messages are written) or begin overwriting earlier messages. Either way, the original causes of a problem can be difficult to determine.

Use the instructions below to check the properties of the Application log:

1. In the Windows Event Viewer, right-click the **Application** log in the left-side pane and select **Properties** from the context menu.

2. Increase the Maximum log size. This determines the number of messages that can be retained by the log. The size of log that is required depends on the amount of activity that is likely to be recorded over time. In this example (Figure 8-17 on page 260) the log size is 960 KB.
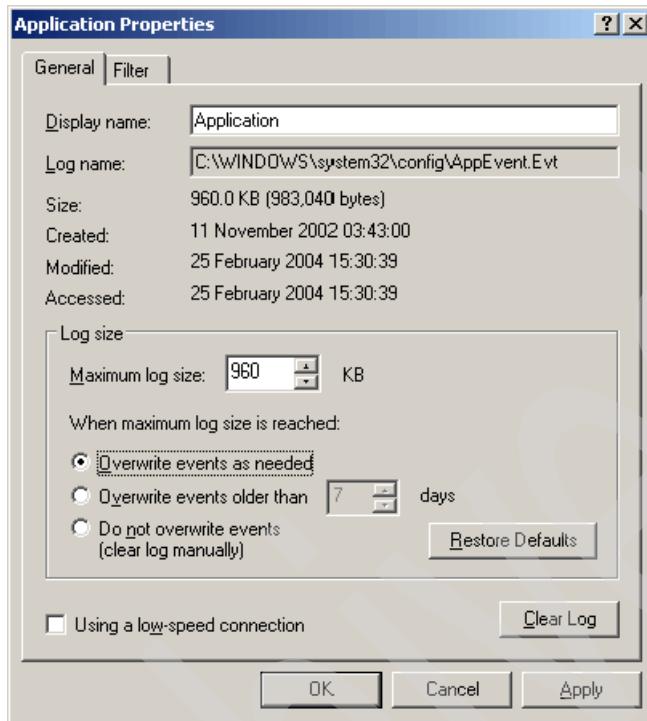
Figure 8-17   Application log properties

3. Select an action under "When maximum log sized is reached." Selecting the first option, Overwrite events as needed, is recommended, as this prevents the log from becoming full and the loss of recent messages. However, the option that is selected must take into account local working practices and how long event messages may have be retained. Clearing the log manually means that no old messages are lost, but it does run the risk of the log becoming full and new messages not being recorded.

The log can be manually cleared at any time by clicking the **Clear Log** button. The Application log can also be saved by right-clicking the Application log and selecting **Save log file as** from the context menu.

8.1.6 Locating more information about event messages

In addition to the information that accompanies event messages, there are two other locations that can be used to look up more information:

► Information Center
► Messages.html

Finding diagnostic messages in the Information Center

The first of these is in the Information Center in the Message Brokers Toolkit (or other source of product help). Use the following instructions to locate information about a BIP message:

1. Open the Message Brokers Toolkit Help by selecting **Help** → **Help Contents**.

2. Click **WebSphere Message Broker V6.0** in the list of contents.

3. Select **Diagnostic messages** from the bottom of the WebSphere Message Broker V6.0 list of contents.

4. In the Diagnostic messages page on the right side, enter a four-character number in the box labelled *Enter message number* and click the **Search** button to locate the BIP message description with that number. An example for BIP5285 is shown in Figure 8-18.

Figure 8-18   Searching for diagnostic messages in the Information Center

5. For each message, the following information is returned:

   – The BIP code

- Severity
- Message
- Explanation
- Response

Finding diagnostic messages in the messages.html file

The second place to find more information about BIP messages is in an HTML file called messages.html. This file is found in the messages directory in the install path. In a typical Windows install this can be found in C:\Program Files\mqsi\6.0\messages. Several directories exist here for translated messages, and in each is a messages.html file.

The messages.html file contains a list of BIP messages in order, in a table, with links to further information at the top of the file. Locate the required BIP message and click the link to view the information about it. This information is the same as the Help system except for the addition of an Inserts table for each message, which details the variable information for that message. For example, Table 8-1 shows the Inserts information for message BIP2623. The message displayed with a BIP2623 is as follows:

```
Unable to open queue '&2' on WebSphere MQ queue manager '&1': completion code
&3; reason code &4.
```

Using the table provided in the messages.html file enables the source of variable information in the error message to be determined. Replace the ampersand (&) variables with the descriptions from the table to understand the message. If the error message in this example was received, then the next logical step is to look up the WebSphere MQ reason code to help determine the cause of the error.

Table 8-1   Example inserts for a BIP message

Insert number	Description	Datatype
&1	MQ queue manager name	CHARACTER
&2	MQ queue name	CHARACTER
&3	MQ return code	CHRACTER
&4	MQ reason code	CHARACTER

8.1.7 Other useful logs

A variety of other logs that record information are created when using WebSphere Message Broker. This section gives a brief description of some of the other logs that are generated.

Install logs

Each product produces a log during installation. If any errors occur during installation, the details are recorded in the installation logs.

WebSphere Message Broker runtime

Installation of WebSphere Message Broker on Windows creates these logs in the home directory:

► mqsi6_install.log
► mqsi6_envvar.log

In Windows the home directory is usually C:\Documents and Settings\UserID, where UserID is the login ID of the user that installed the product. The main log for the runtime installation is the mqsi6_install.log.

Message Brokers Toolkit

The Message Brokers Toolkit install log can be found in the installation directory of the Message Brokers Toolkit, for example:

```
C:\MessageBrokers\logs\wmbt_install.log
```

The log for the uninstall of the Message Brokers Toolkit can be found in the home directory, for example:

```
C:\Documents and Settings\UserID\wmbt_uninstall.txt
```

WebSphere MQ

The installation log for WebSphere MQ is created in the Temp directory. This varies from machine to machine. To find the location of Temp type the following on a command line:

```
set temp
```

This displays the location of the temp directory. The WebSphere MQ install log file also includes a time and date stamp in the name. An example WebSphere MQ install log location and name are shown below for WebSphere MQ V6.0:

```
C:\Documents and Settings\wmbuser\Local
Settings\Temp\MQv6_Install_2005-10-06T12-48-54.log
```

ODBC Drivers for Cloudscape or DB2 Universal Database

The installation log for any DB2 Universal Database component including the ODBC Drivers for Cloudscape can also be found in the home directory. The following examples show the name and an example location for the two logs that are produced by these DB2 products:

► C:\Documents and Settings\UserID\My Documents\DB2LOG\db2.log
► C:\Documents and Settings\UserID\My Documents\DB2LOG\db2wi.log

The db2.log contains all of the install log information for every DB2 Universal Database component installed, while the db2wi.log contains information for just the last DB2 Universal Database component installed on the machine.

MRM logs

A number of logs are generated in Message Set Projects during the import or export of message definitions such as XML schema. A log is also created by the runtime if deployment of a message set with a TDS layer fails. This type of error is indicated by a BIP1836 message in the Message Brokers Toolkit Event Log.

8.2 Using the message Flow Debugger

In order to use the message Flow Debugger, the Rational Agent Controller must be installed and running on the system where the flow is deployed. Instructions for installing the Rational Agent Controller can be found in "Rational Agent Controller" on page 20.

Breakpoints must be added to message flows in order to track the progress and status of any messages passing through the flow.

The message Flow Debugger is useful for determining what actually happens in a flow, and for tracking down problems or unexpected behavior. It is especially useful in complex flows, where the route of messages can be tracked, and input and output messages can be verified through the flows.

Using the Flow Debugger ESQL code, Java code and mappings can be stepped into, so that the effect of each line of code on the message in the message flow can be seen. In this section, a number of examples are used to demonstrate some of the capabilities of the message Flow Debugger.

The message Flow Debugger has its own perspective in the Message Brokers Toolkit, which can be accessed by choosing **Debug** from the Open Perspective command on the Window menu.

Some of the values in messages or variables created by code in the node can be interactively altered while a message is passing through a message flow. This is useful for testing or debugging how a message flow handles expected and unexpected values for fields in a message.

Important: There are some known problems with the message Flow Debugger at release of WebSphere Message Broker V6.0. Contact your IBM Support Center if difficulties occur when invoking the Flow Debugger for information about available fixes.

8.2.1 Adding breakpoints to a message flow

A breakpoint is a point defined between nodes in a message flow at which point progress of a message is stopped by the message Flow Debugger so that the message can be examined and altered. If no breakpoints are created in a flow, then a message passes through the flow without being stopped, so it cannot be examined.

There are several ways to add breakpoints to connections between nodes:

1. Right-click a connection between two nodes and select **Add Breakpoint** from the context menu.

2. Right-click a node and select **Add Breakpoints Before Node** from the context menu to add a breakpoint on all connections that enter the node.

3. Right-click a node and select **Add Breakpoints After Node** to add a breakpoint on all connections that leave the node.

If it is known that the problem that is under investigation occurs between a particular set of nodes, then breakpoints can be added only in those locations. For tracking the entire progress of a test message, all of the connections must be given breakpoints. A subflow can also be given breakpoints, but If no breakpoints are created in a subflow, then the message Flow Debugger does not stop progress at a subflow, and the message is only tracked going into and out of the subflow, but not through it.

Figure 8-19 on page 266 shows a message flow containing breakpoints between the nodes. This figure also shows a message stopped at a breakpoint, when the debugger is attached to the execution group. In the figure the breakpoints are displayed as circles on the connections between the nodes. The message is stopped between the XML_CANCELRESERVATION_IN node and the DeleteReservation node, and is represented by a highlight around a breakpoint.

Attaching the debugger is discussed in more detail in the next section.
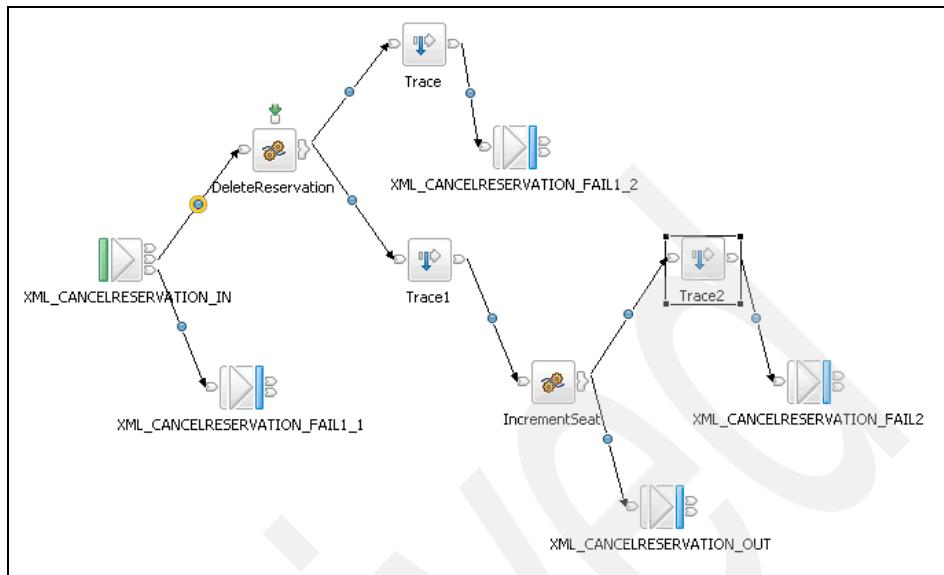
Figure 8-19   Message flow showing breakpoints on the connections

A deployed message flow can have breakpoints added and removed from it without the need to redeploy.

8.2.2 Attaching the Flow Debugger

The message Flow Debugger must be attached to a specific execution group in order to start debugging message flows. This means that any flows that are running in that execution group, including subflows, can be debugged at the same time. To attach the message Flow Debugger a debug configuration must be created. Use the following instructions to create a debug configuration:

1. Change to the Debug perspective by selecting **Open Perspective** → **Debug** from the Window menu.

2. In the Debug perspective, click the Debug button (🐞 ▾) on the toolbar.

3. On the Debug dialog click **Message Broker Debug**, as shown in Figure 8-20 on page 267.

4. Click **New**.

5. Enter a name for the flow debug configuration.

6. Select the Flow Project that contains the flows that need to be debugged by clicking **Browse** next to Flow Project.

7. Click **Browse** to select a Flow Engine (execution group). This displays a list of the running execution groups, as shown in Figure 8-21 on page 267.
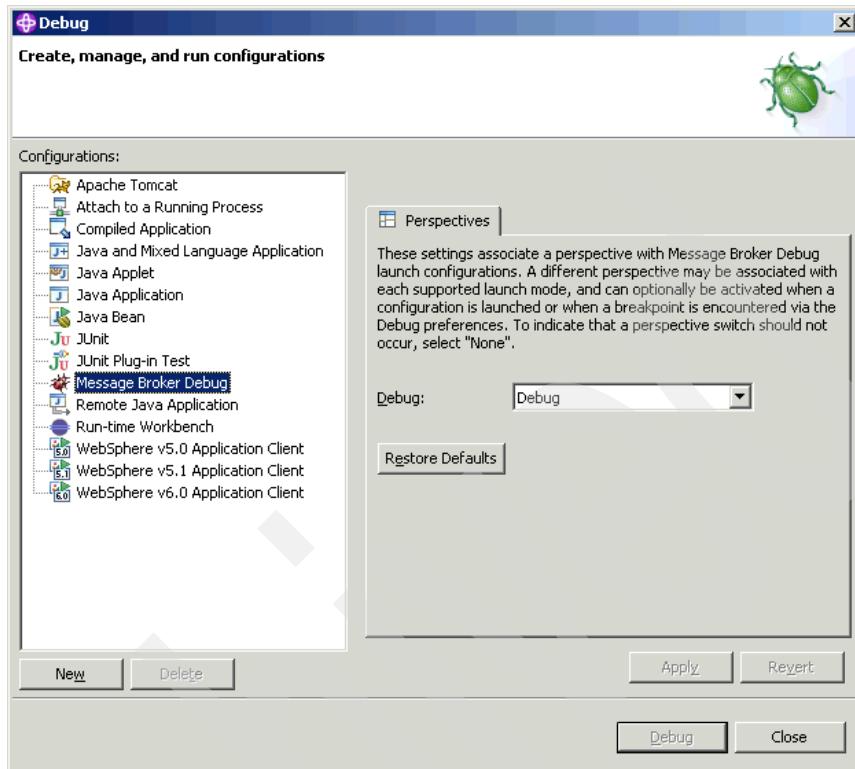
Figure 8-20   Selecting a debug configuration type

8. Select the execution group to debug from the Flow Engine List and click **OK**.

9. If the execution group contains message flows with Java code to debug, further setup is required; see "Debugging Java code" on page 273.

10.Click the Debug button to connect the Flow Debugger.
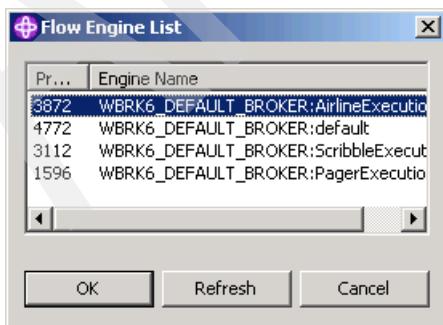
Figure 8-21   Available execution groups in the Flow Engine List

> **Note:** To debug more than one execution group at a time a separate debug configuration must be created for each one. As a warning debug is processor intensive, so debugging multiple execution groups at the same time can use large amounts of virtual memory.

The execution groups connected to the debugger are displayed in the top left of the Debug perspective. Figure 8-22 shows two debug configurations each connected to an execution group.
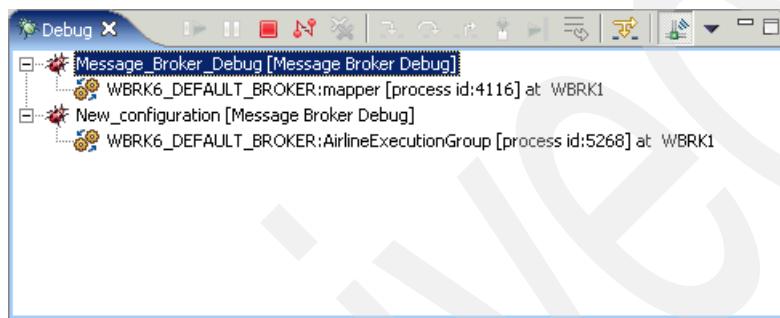
Figure 8-22   Execution groups connected to the Flow Debugger

Once the Flow Debugger is connected to an execution group, any of the flows in the execution group can be debugged.

8.2.3 Tracking a message through a flow

The next stage in using the Flow Debugger is to put a test message through a message flow. This task can be performed by using any tool that can put a message to an input node, for example, using an enqueue file. Breakpoints must have been added to the message flows that the messages are put to, so that the message stops at those points in the flow. If you want to try this out, use the WebSphere Message Broker samples deployed to the default configuration, or use the message flows and message sets created in Chapter 4, "Developing applications with ESQL" on page 47.

The following instructions can be used to track a test message through a message flow.

1. Add breakpoints to the message flow.

2. Attach the Flow Debugger to the execution group to which the message flow is deployed.

3. Use an existing message enqueue file, or create a new file to put a message to the message flow.

The Flow Debugger opens the message flow to which the message has been put and highlights the first connection with a breakpoint to indicate the progress of the message. Figure 8-19 on page 266 shows this for the Cancel Reservation message flow from the airline sample.

The content of the message at this point can also be viewed by expanding the message in the Variables view on the right side of the Debug perspective, as shown in Figure 8-23.
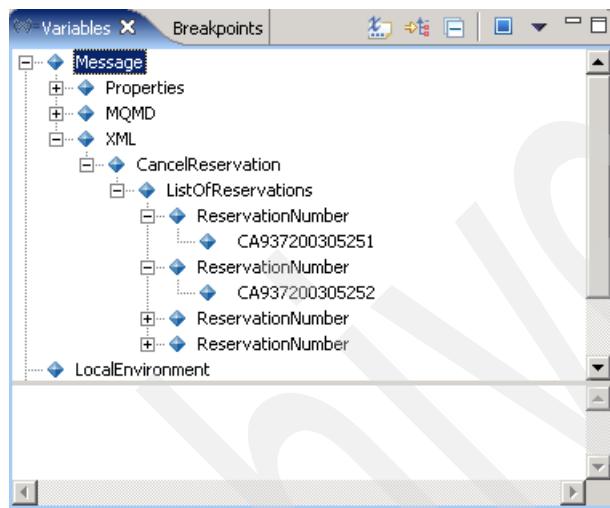
Figure 8-23   Input message in the Variables view

In order to control the movement of a message through a message flow using the Flow Debugger, there are a number of buttons on the toolbar. These buttons are shown in Figure 8-24.

Figure 8-24   Debug toolbar

The first button the debug toolbar is the Resume button. This allows the message to progress through the flow. It allows the message to move to the next breakpoint. The message is processed by any nodes that are present between this breakpoint and the next one.

The Step Over button (fourth from right) performs a similar action to resume, except that the node between this breakpoint and the next does not process the message in the message flow.

The Run to Completion button, the final button on the far right, passes the message through the flow without stopping at any breakpoints. The message continues normal processing until it is passed out of the flow.

In order to debug ESQL code, Java code, or mappings in a message flow, the Step into Source Code button must be clicked to step into the code. This button (second from right) is only highlighted when the message is stopped at a breakpoint before a node for which the source can be debugged, such as a Filter node, JavaCompute node, or a DataInsert node.

When a message is stopped at a breakpoint on a connection before a node containing code or a mapping that can be debugged, an arrow with a square appears over the top of the node. This arrow icon matches the icon used for the Step Into Source Code button. This can be seen in Figure 8-25, which shows a breakpoint before a Mapping node. The presence of this arrow means that the Step into Source button can be pressed to step into the code or mapping at this point.
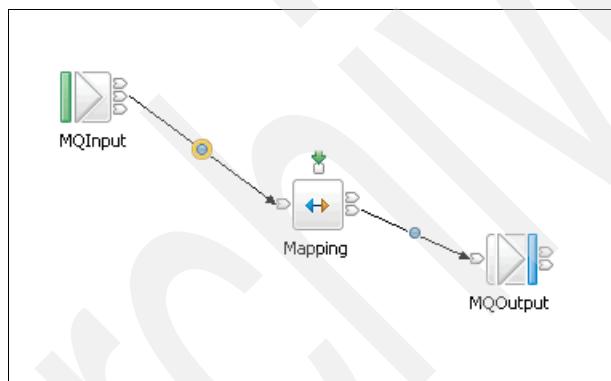
Figure 8-25   Step into Source indicator

8.2.4 Stepping through ESQL

To debug ESQL code in a node, click the Step Into Source Code button on the toolbar when a message travelling through a flow is stopped at a breakpoint before the node containing ESQL. This opens the ESQL for the node in the bottom half of the Debug perspective. The arrows on the toolbar of the debug view can be used to step line-by-line through the ESQL.

The Variables view displays a number of items when used with ESQL. The DebugMessage is a copy of the message received by the node, plus information that may have been generated by other nodes in the flow, such as exception and environment information.

The OutputRoot is the output message that the ESQL is creating to pass to the next node in the flow. This is where the ESQL is constructing an output message from information in an input message, a database, or other logic in the ESQL itself. Stepping through the ESQL demonstrates how the output message is built up line by line.

Temporary variables that are created by the ESQL are also displayed in the Variables view. Figure 8-26 shows an example of the contents of the Variables view, using the cancel reservation message flow from the Airline sample. This shows the output message being built, plus the variables I and J, which are used in the ESQL functions for the node.
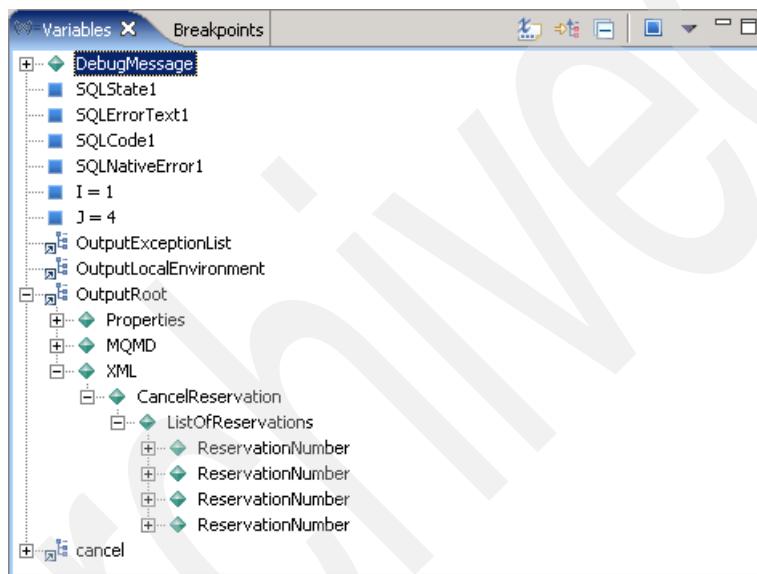
Figure 8-26   Example Variables view for ESQL

The buttons on the debug toolbar are used to either step line by line through the code, or to skip to the end onto the next node. After the ESQL code is completed, the ESQL code closes and the message flow is displayed with the message stopped at the next breakpoint after the node with ESQL. The message can then be passed through the message flow or the next node stepped into.

8.2.5 Stepping through mappings

To debug a node with mappings, click the Step Into Source Code button on the toolbar when a message travelling through a flow is stopped at a breakpoint before the node with mappings. This opens the message map for the node in the

bottom half of the Debug perspective. The arrows on the toolbar of the debug view can be used to step line by line through the mapping.

Message maps can have breakpoints added directly to rows in the spreadsheet view in order to debug specific parts of a map. Right-click the grey bar on the left of the spreadsheet view and select **Add Breakpoint**. This makes a breakpoint appear next to the current position in the map in the Message Mapping editor. Three breakpoints can be seen in Figure 8-27.
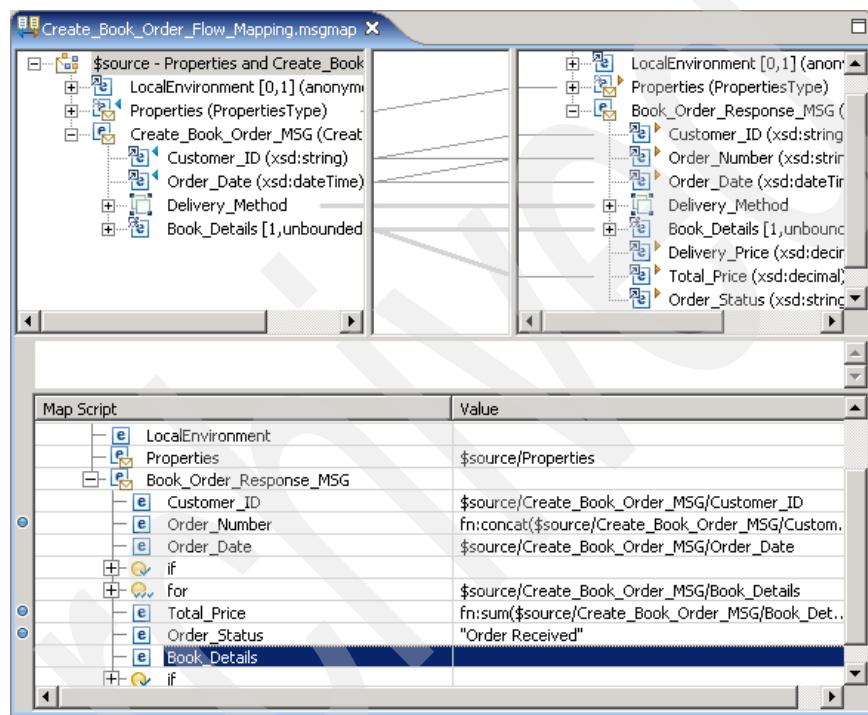
Figure 8-27   Mapping editor with breakpoints set

The Variables view for mappings is more complex than it is with ESQL. The same types of information are displayed, but various extra objects are also present that relate to how the mappings are internally constructed in the message flow. Figure 8-28 on page 273 shows an example Variables view for a message map.
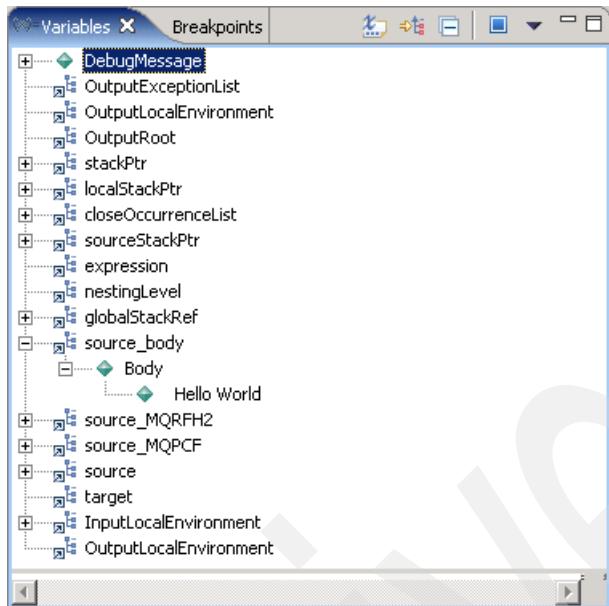
Figure 8-28   Example Variables view for a mapping

After the message map code is completed, the message map closes and the message flow is displayed with the message stopped at the next breakpoint after the node with mapping. The message can then be passed through the message flow or the next node stepped into.

8.2.6 Debugging Java code

In order to debug Java code, in a JavaCompute node, or a user-defined node, a change must be made to the broker to provide a debug Java port number for the broker JVM. This change must be made before a debug configuration can be created for a project containing Java to be debugged.

Specifying a Java port

The following command must be entered on the Command Console to specify the Java port:

```
mqsichangeproperties BROKER1 -e javaComputeEG -o ComIbmJVMManager -n
jvmDebugPort -v 4455
```

In this example, BROKER1 is the name of the broker, javaComputeEG is the execution group name, and 4455 is a port number. Check that the port you specify is free before running this command. See "WebSphere MQ resources" on page 207 for instructions on how to choose a free port.

Once the `mqsichangeproperties` command has been run, the broker must be restarted for the changes to take affect. Use an `mqsistop` and `mqsistart` with the broker name to stop and restart the broker.

Creating a debug configuration for Java

A new debug configuration must be created that contains the debug Java port for the execution group. Use the following instructions to set up a debug configuration:

1. Create a new debug configuration using the instructions above.

2. Click **Java debug setting** to set the Java port.

3. Check the **Debug Java Source Code** box.

4. Enter the port assigned to the execution group using the `mqsichangeproperties` command run previously. (Figure 8-29 on page 275 shows an example where the Java port is 4455.)

5. Click **Debug** to connect the debugger to the execution group containing the Java code to be debugged.

6. Click the **Step Into Source Code** button on the toolbar when a message travelling through a flow is stopped at a breakpoint before the node containing Java code.

This opens the Java for the node in the bottom half of the Debug perspective. The arrows on the toolbar of the debug view can be used to step line by line through the Java code.
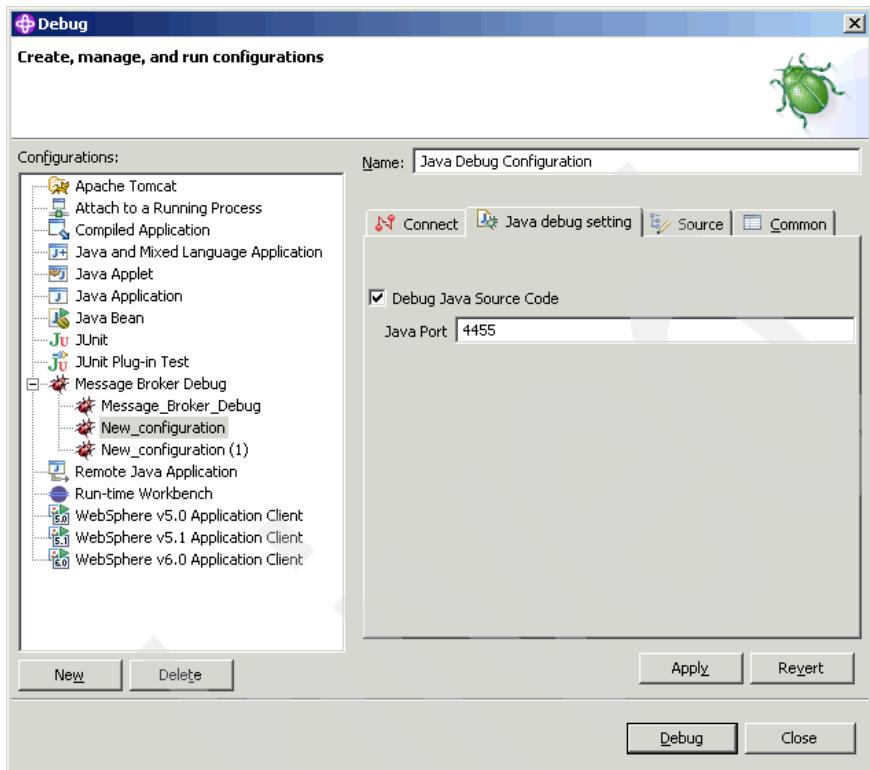
Figure 8-29   Setting the Java port

The Variables view displays items that are different from those seen in mappings or ESQL and instead represent the objects seen in the Java code. If you are familiar with the Rational Application Development platform, the Flow Debugger when used with the JavaCompute node or user-defined nodes is the same as the normal Java debugger. An example of the Variables view when a JavaCompute node is being debugged is shown in Figure 8-30 on page 276.

After the Java code is completed, the Java code closes and the message flow is displayed with the message stopped at the next breakpoint after the JavaCompute node. The message can then be passed through the message flow or the next node stepped into.
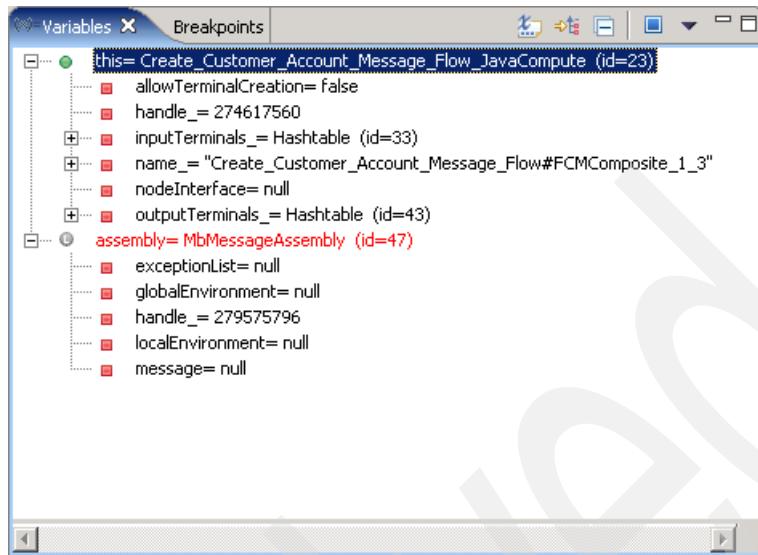
Figure 8-30   Example Variables view when debugging Java code

8.2.7 Flow of errors in a message flow

In an ordinary message flow, the progress of the message is from the Input node through the message flow in a downstream direction. However, when an error occurs, the flow changes and messages start to move upstream as they begin to roll back. If error handling capabilities are not added to a message flow, then when errors occur the message rolls back until it reaches the input node and is backed out onto the input queue.

It is helpful to look at message flows by using the debugger, as this shows how messages are handled and rolled back when an error occurs. To demonstrate this, put an invalid message to the a message flow. A number of the samples intentionally supply messages with errors to demonstrate the effects of putting an invalid message through a message flow. These can be used to demonstrate the flow of an error. As an alternative, the examples created in Chapter 4, "Developing applications with ESQL" on page 47, can be used by altering the structure or data in the input messages to make them invalid.

If an error occurs in the message flow while the debugger is attached, the following happens:

► An entry is created in the ExceptionList for the error that occurred.
► The message begins to flow back up the message flow.

As the message begins to flow back up the message flow towards the input node, exceptions are added to the ExceptionList. The message continues to flow up the message flow until the message reaches a node with either the Failure terminal or the Catch terminal connected. If these are connected then the message begins to flow down that part of the message flow unless another error occurs. Figure 8-31 shows an example of errors in the ExceptionList, after a database action has failed.
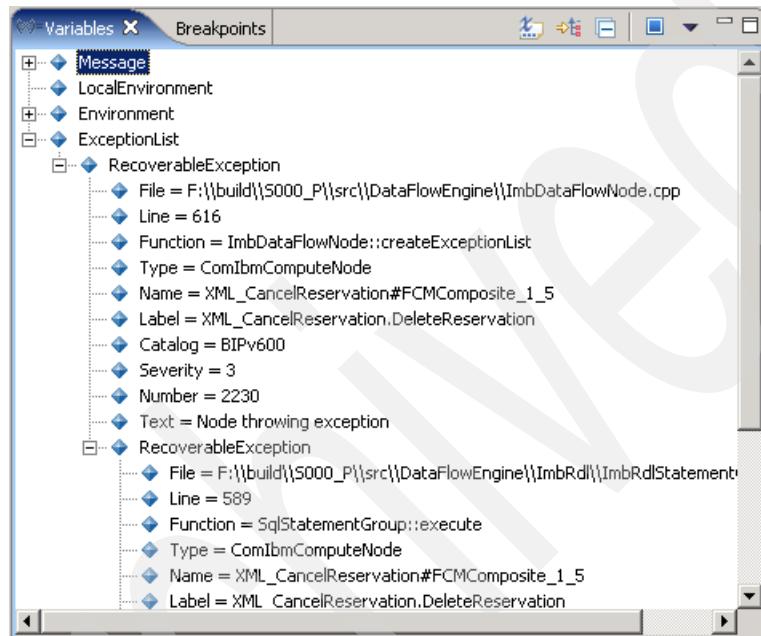
Figure 8-31   Errors in the ExceptionList

The Error Handler sample demonstrates how different error handling techniques affect an invalid message in the flow, and is an excellent flow to observe under the debugger in order to understand the flow of errors in message flows, and how to implement error handling techniques for your own flows.

8.2.8 Disconnecting the debugger

It is recommended when using the message Flow Debugger that all messages should be passed out of the flow before disconnecting the Flow Debugger. It is also not possible to deploy to an execution group when the debugger is connected to it.

In the Debug perspective, right-click the execution group connection in the top left of the perspective and select **Disconnect** from the context menu.

8.3 Using trace

This section gives an overview of using trace with WebSphere Message Broker.
Trace is used when a problem occurs that cannot be solved using any of the
methods that have been described so far in this chapter. Trace is used most
often on execution groups, where it can be used to work out exactly what is
happening when a message passes through a message flow. Message flows
can even be designed to output user-specified trace, such as the content of the
message tree using a Trace node.

Trace can also be used for helping to debug problems with WebSphere Message
Broker commands or components, although this is usually only used for the
purpose of collecting trace for service if a potential product defect is occurring. In
addition, you also can start and collect trace from components of the Message
Brokers Toolkit.

The types of trace that are described here are set up and collected only when
necessary, as tracing results in a decrease in performance due to the extra
processing that is required in a system.

For tracing on execution groups, components, and commands, there are
different types and levels of trace and amounts of detail: User trace, service
trace, and normal and debug levels of trace. Of these combinations, user trace at
normal level has the least information, and service trace at debug level has the
greatest amount of information. However, most of the information that is
contained in the service trace is not of value to most users.

8.3.1 Tracing execution groups

Tracing of execution groups to obtain detailed information about what happens to
messages that pass through a flow is straightforward and very useful. The Error
Handler sample has been used here to demonstrate how to perform trace on an
execution group, and to show how to read the output from a trace file. These
same instructions can be used for any deployed message flows.

User trace level normal

This section traces messages through the Error Handler message flow using
user trace at normal level. This example uses these parameters BROKER1 for
the broker and ErrorHandler for the execution group with the Error Handler
sample deployed to it.

To set user level trace for an execution group enter the following command in the Command Console, substituting the broker name and execution group with the appropriate names:

```
mqsichangetrace BROKER1 -u -e ErrorHandler -l normal -r
```

mqsichangetrace is the command that changes the trace settings on the broker name that follows it (in this case BROKER1). The parameter -u indicates that the type of trace required is user trace; -e and the name following it are for the execution group to trace (in this case the execution group is called ErrorHandler); -l is the level of the trace, in this case normal; and -r is to reset the trace log. The -r parameter is optional, but it is useful to reset the log (empty it); otherwise, the trace file can become very large and difficult to read.

The result of the command that has been run is to change the trace on the broker and execution group to which the Error Handler sample is deployed to user trace at normal level. This means that trace is now collected for certain actions that occur within this execution group, for example, messages moving within a message flow. The next step is to put a message through the message flow to generate some of these actions that are recorded in the trace.

Type the following command into the Command Console to read the contents of the trace log into an XML file, substituting the names of the broker and execution group:

```
mqsireadlog BROKER1 -u -e ErrorHandler -o Trace1.xml
```

The **mqsireadlog** command reads the contents of the trace log into an XML file. The other parameters that are given with the command determine which trace log the information is being read from, as many components, each with its own trace log, may be being traced. In this example, the name that follows the **mqsireadlog** command is the broker name, the -u takes the user trace information from the trace log, -e and the name following it are the execution group being traced, and the -o parameter is the name of the output file to put the trace information to. This does not require a file extension, but it is useful to give it an xml file extension so that it is recognizable as an XML file rather than a fully formatted trace file.

At this stage it is possible to locate and view the trace file, but it is in XML format and is difficult to read. Therefore, a further step must be carried out to get the file into a formatted state for interpretation. This can be performed using the **mqsiformatlog** command.

Type the following command onto the Command Console:

```
mqsiformatlog -i Trace1.xml -o Trace1.txt
```

The **mqsiformatlog** command converts an XML file that was generated by the **mqsireadlog** command into a formatted file that can be read by a text editor. It has two parameters: -i is the name of the XML input file, and -o is the name to give to the output file.

On Windows, it is useful to give the file a txt file extension so that the file can be opened from the command line into the Notepad program simply by typing the name of the file on the command line. If Notepad is used as the text editor, ensure that Word Wrap is *not* on, as this makes reading the trace easier.

The output in the trace file consists of several lines of user trace, each with a timestamp at the beginning, followed by a four-character code, the type of trace (in this example all messages are UserTrace), a BIP code, and a description of the event with details such as node names, queue names, and parser types. This gives details of all events that occur as the message passes through the message flow.

Example 8-1 shows an example of the content of a user trace file. The timestamps and source information have been removed.

Example 8-1   Excerpt from a usertrace file

```
BIP2632I: Message received and propagated to 'out' terminal of MQ input node
'Main_Flow.STAFF_IN'.
BIP6060I: Parser type 'Properties' created on behalf of node
'Main_Flow.STAFF_IN' to handle portion of incoming message of length 0 bytes
beginning at offset '0'.
BIP6061I: Parser type 'MQMD' created on behalf of node 'Main_Flow.STAFF_IN' to
handle portion of incoming message of length '364' bytes beginning at offset
'0'. Parser type selected based on value 'MQHMD' from previous parser.
BIP6061I: Parser type 'XML' created on behalf of node 'Main_Flow.STAFF_IN' to
handle portion of incoming message of length '133' bytes beginning at offset
'364'. Parser type selected based on value 'XML' from previous parser.
BIP4004I: Message propagated to 'true' terminal of filter node 'Main_Flow.Check
Backout Count'.
BIP4080I: Message propagated to try terminal from try-catch node
'Main_Flow.Error_Handler.TryCatch'. The try-catch node
'Main_Flow.Error_Handler.TryCatch' has received a message and is propagating it
to any nodes connected to its try terminal. No user action required.
BIP4004I: Message propagated to 'true' terminal of filter node 'Main_Flow.Check
Valid Staff Number'.
BIP4184I: Message propagated to 'out' terminal of database node
'Main_Flow.Update Staff Database'.
BIP2638I: The MQ output node 'Main_Flow.STAFF_OUT' attempted to write a message
to queue 'STAFF_OUT' connected to queue manager ''. The MQCC was '0' and the
MQRC was '0'.
```

```
BIP2622I: Message successfully output by output node 'Main_Flow.STAFF_OUT' to
queue 'STAFF_OUT' on queue manager ''.
```

Example 8-1 on page 280 does not show any errors in the usertrace; the BIP
messages all end in *I*, indicating that they are for information only. If an error
occurs while trace is switched on, error messages are recorded in the trace,
together with information that can help indicate the cause of the problem.

Example 8-2 shows an example of user trace generated when an error has
occurred in a message flow. In this case an invalid message has been put to the
message flow. A different chain of events is shown here than the previous trace
file. Instead of UserTrace, the word `Error` appears in front of the BIP2232 error
message. This message is in fact displayed in the Application log in the Windows
Event Viewer, and it is the only BIP message recorded in either of these trace
files that is displayed in the Event Viewer. The exceptions produced by the
invalid message are clearly seen.

There is also an example of a UserException shown by the BIP3002 message
where a user exception has been generated as part of the error handling in the
flow itself.

Example 8-2   Excerpt of user trace showing error conditions

```
BIP4005I: Message propagated to 'false' terminal of filter node
'Main_Flow.Check Valid Staff Number'.
BIP4101I: Exception thrown by throw node 'Main_Flow.Throw'. The throw node
'Main_Flow.Throw' has received a message and will throw an exception as this is
its normal behavior. No user action required.
BIP4081I: Message propagated to catch terminal from try-catch node
'Main_Flow.Error_Handler.TryCatch'. The try-catch node
'Main_Flow.Error_Handler.TryCatch' has caught an exception which occurred in a
node connected to its try terminal. The message has been augmented with an
exception list and is propagating it to any nodes connected to its catch
terminal for further processing. See the following messages for details of the
exception list. No user action required.
BIP3001I: Exception thrown by throw node 'Main_Flow.Throw'; text is 'Invalid
staff number'. The throw node 'Main_Flow.Throw' has received a message and thus
has thrown an exception as this is its normal behavior. The message text
associated with this exception is 'Invalid staff number'. Since this is
application generated (by message flow behavior), the user action is determined
by the message flow and the type of exception generated.
BIP4184I: Message propagated to 'out' terminal of database node
'Main_Flow.Update Error Database'.
BIP4101I: Exception thrown by throw node 'Main_Flow.Error_Handler.Throw To
Complete Rollback'. The throw node 'Main_Flow.Error_Handler.Throw To Complete
Rollback' has received a message and will throw an exception as this is its
normal behavior. No user action required.
```

```
BIP2232E: Error detected whilst handling a previous error in node
'Main_Flow.Error_Handler.Throw To Complete Rollback'. The message broker has
detected an error in node 'Main_Flow.Error_Handler.Throw To Complete Rollback'
whilst handling a previous error. See the following messages for details of the
exception list associated with the original error. Thereafter messages will be
associated with the new error.
BIP3001I: Exception thrown by throw node 'Main_Flow.Throw'; text is 'Invalid
staff number'. The throw node 'Main_Flow.Throw' has received a message and thus
has thrown an exception as this is its normal behavior. The message text
associated with this exception is 'Invalid staff number'. Since this is
application generated (by message flow behavior), the user action is determined
by the message flow and the type of exception generated.
BIP2231E: Error detected whilst processing a message 'Main_Flow.STAFF_IN'. The
message broker detected an error whilst processing a message in node
'Main_Flow.STAFF_IN'. The message has been augmented with an exception list and
has been propagated to the node's failure terminal for further processing. See
the following messages for details of the error.
BIP3002I: Exception thrown by throw node 'Main_Flow.Error_Handler.Throw To
Complete Rollback'; text is 'From Error_Handler message flow. See ERRORDB for
details.'. The throw node 'Main_Flow.Error_Handler.Throw To Complete Rollback'
has received a message and thus has thrown an exception as this is its normal
behavior. The message text associated with this exception is 'From
Error_Handler message flow. See ERRORDB for details.'. Since this is
application generated (by message flow behavior), the user action is determined
by the message flow and the type of exception generated.
BIP2638I: The MQ output node 'Main_Flow.STAFF_FAIL' attempted to write a
message to queue 'STAFF_FAIL' connected to queue manager ''. The MQCC was '0'
and the MQRC was '0'.
BIP2622I: Message successfully output by output node 'Main_Flow.STAFF_FAIL' to
queue 'STAFF_FAIL' on queue manager ''.
```

This level of trace is sufficient for the majority of purposes, but in order to get even more information, for example, down to the level of actions performed on a message by ESQL, then a higher level of trace is required.

User trace level debug

To demonstrate a user trace at debug level, the same message flow and invalid message is used as in the section above.

To set a user trace level of debug, the following command must be entered in the Command Console, substituting the broker and execution group name:

```
mqsichangetrace BROKER1 -u -e ErrorHandler -l debug -r
```

In this example, BROKER1 is the name of the broker and ErrorHandler is the name of the execution group. The following command can be used to read the trace log when a message has been put to the message flow.

```
mqsireadlog BROKER1 -u -e ErrorHandler -o Trace2.xml
```

This command on the Command Console converts the XML file from Trace2.xml to a text file called Trace2.txt:

```
mqsiformatlog -i Trace2.xml -o Trace2.txt
```

In comparison to a trace file at normal level, it is obvious that a lot more information is recorded in the debug trace file, making it harder to read but much clearer to work out exactly what is happening to a message in a message flow. A line is included for the evaluation of each line of ESQL that is run in the message flow. Example 8-3 shows an example of a debug level user trace. This example demonstrates how errors that occur in a message flow can be seen in the trace file.

The BIP2539 message indicates that the result of an ESQL evaluation was false. This passes the message to the False terminal of the Filter node, which the message is being evaluated by. A Throw node is connected to the Filter node in this message flow, causing an exception to be thrown, as seen in the following BIP messages.

Example 8-3   Excerpt of user trace at debug level

```
BIP2537I: Node 'Main_Flow.Check Valid Staff Number': Executing statement 'IF
Body.Staff.StaffNumber <= '10' THEN... ELSE... END IF;' at
(.Main_Flow_Filter.Main, 3.2).
BIP2538I: Node 'Main_Flow.Check Valid Staff Number': Evaluating expression
'Body.Staff.StaffNumber <= '10'' at (.Main_Flow_Filter.Main, 3.27).
BIP2538I: Node 'Main_Flow.Check Valid Staff Number': Evaluating expression
'Body.Staff.StaffNumber' at (.Main_Flow_Filter.Main, 3.5).
BIP2539I: Node 'Main_Flow.Check Valid Staff Number': Finished evaluating
expression 'Body.Staff.StaffNumber <= '10'' at (.Main_Flow_Filter.Main, 3.27).
This resolved to ''99' <= '10''. The result was 'FALSE'.
BIP2537I: Node 'Main_Flow.Check Valid Staff Number': Executing statement
'RETURN FALSE;' at (.Main_Flow_Filter.Main, 6.3).
BIP4005I: Message propagated to 'false' terminal of filter node
'Main_Flow.Check Valid Staff Number'.
BIP4101I: Exception thrown by throw node 'Main_Flow.Throw'. The throw node
'Main_Flow.Throw' has received a message and will throw an exception as this is
its normal behavior. No user action required.
BIP4081I: Message propagated to catch terminal from try-catch node
'Main_Flow.Error_Handler.TryCatch'. The try-catch node
'Main_Flow.Error_Handler.TryCatch' has caught an exception which occurred in a
node connected to its try terminal. The message has been augmented with an
exception list and is propagating it to any nodes connected to its catch
terminal for further processing. See the following messages for details of the
exception list. No user action required.
BIP3001I: Exception thrown by throw node 'Main_Flow.Throw'; text is 'Invalid
staff number'. The throw node 'Main_Flow.Throw' has received a message and thus
```

> has thrown an exception as this is its normal behavior. The message text
> associated with this exception is 'Invalid staff number'. Since this is
> application generated (by message flow behavior), the user action is determined
> by the message flow and the type of exception generated.

Debug level user trace is very useful for tracking exactly what happens to a message in a message flow and why. It can be used as an alternative to the Flow Debugger for debugging problems, and shows greater detail on the events in a message flow. Unlike the Flow Debugger, the values of items in a message or variables can not be altered. It is less easy to follow than the visual representation of the message travelling through the message flow.

Service trace

Service trace activates more comprehensive tracing than the user trace, and can additionally be used to trace the Message Brokers Toolkit, Configuration Manager, User Name Server, and any of the WebSphere Message Broker commands. Service trace is usually only activated when an error message or an IBM Support Center requests that you collect this level of trace. Instructions for using service trace for components and commands are given in the following sections.

Service trace for message flows can be collected using the same commands as user trace. The only difference is that the -u parameter in the **mqsichangetrace** and **mqsireadlog** commands is replaced by a -t parameter. An example is:

```
mqsichangetrace BROKER1 -t -e ErrorHandler -l debug -r
```

In this example BROKER1 is the name of the broker and ErrorHandler is the name of the execution group.

Display trace settings

The trace settings on an execution group or other WebSphere Message Broker components can be displayed using the **mqsireporttrace** command.

To display the service trace settings for an execution group, enter the following on the Command Console, substituting the broker name and execution group name as necessary:

```
mqsireporttrace BROKER1 -t -e ErrorHandler
```

To display the user trace settings for an execution group that was used in the previous examples, enter the following on the Command Console, substituting the broker name and execution group name as necessary:

```
mqsireporttrace BROKER1 -u -e ErrorHandler
```

Reset trace settings

When trace is set on an execution group, additional processing is generated in order to record each activity in the message flows. There is an impact on performance, and therefore tracing should be limited to only those execution groups where tracing is required, and tracing should only be performed for a limited amount of time.

After trace has been collected, reset the trace settings to *none* to stop trace from being written. For example, enter the following command in the Command Console:

```
mqsichangetrace BROKER1 -t -e ErrorHandler -l none
```

This command resets the service trace to none for the execution group ErrorHandler on BROKER1. To reset the user trace on an execution group, enter a command in the Command Console using the following syntax:

```
mqsichangetrace BROKER1 -u -e ErrorHandler -l none.
```

8.3.2 Tracing components

On occasion it may be necessary to collect trace for WebSphere Message Broker components, such as the Configuration Manager, the User Name Server, or brokers. These use the **mqsichangetrace** and **mqsireadlog** commands, similar to trace on an execution group. Only service trace can be used for commands, so the -t parameter must be used with the commands, not the -u parameter.

To collect trace for a Configuration Manager use the instructions below, substituting *configmgr* for the name of your Configuration Manager:

1. Enter this command in the Command Console:

   ```
   mqsichangetrace configmgr -t -b -l normal
   ```

2. When trace is ready to be read, enter this command in the Command Console:

   ```
   mqsireadlog configmgr -t -b agent -f -o ConfigmgrTrace.xml
   ```

3. To format the log, enter the **mqsiformatlog** command as before, for example:

   ```
   mqsiformatlog -i ConfigmgrTrace.xml -o ConfigmgrTrace.txt
   ```

4. View the trace file that is generated.

The other runtime components of WebSphere Message Broker can be traced in the same way. This tracing is generally used for collecting information for service.

8.3.3 Tracing commands

Only service trace can be used to trace WebSphere Message Broker commands. Trace for commands does not use the `mqsichangetrace` command; instead, only the `mqsireadlog` and `mqsiformatlog` commands are used, as with the trace on components. However, an extra step must be performed to start trace for the commands by setting the MQSI_UTILITY_TRACE environment variable. The following instructions give an example of collecting trace for two commands, `mqsilist` and the `mqsistop` command on a broker.

To perform debug level service trace on the `mqsilist` command:

1. Open the Command Console and enter:

       set MQSI_UTILITY_TRACE=DEBUG

2. Enter the following and press Enter:

       mqsilist

3. Enter the following and press Enter:

       mqsireadlog utility -t -b mqsilist -f -o mqsilistTrace.xml

4. Enter a `mqsiformatlog` command to format the trace for output to a text file; for example:

       mqsiformatlog -i mqsilistTrace.xml -o mqsilistTrace.txt

5. View the trace file.

To perform normal level service trace on the `mqsistop` command for the broker WBRK_BROKER:

1. Open the Command Console and enter:

       set MQSI_UTILITY_TRACE=DEBUG

2. Enter:

       mqsistop WBRK_BROKER

3. Enter:

       mqsireadlog WBRK_BROKER -t -b mqsistop -f -o mqsistopTrace.xml

4. Enter a `mqsiformatlog` command to format the trace for output to a text file; for example:

       mqsiformatlog -i mqsistopTrace.xml -o mqsistopTrace.txt

5. View the trace file.

The advantage of setting the MQSI_UTILITY_TRACE environment variable on the Command Console rather than in the system is that as soon as the

Command Console window is closed, the tracing is stopped. To manually stop the tracing of commands, type this in the Command Console:

```
set MQSI_UTILITY_TRACE=none
```

It is important to reset these variables like this when the command that you are tracing has completed. If MQSI_UTILITY_TRACE is left as it is, all subsequent commands are also traced. This leads to a degradation in performance of these commands.

8.3.4 Tracing the Message Brokers Toolkit

Various components of the Message Brokers Toolkit for WebSphere Message Broker can have trace activated on them. This is not described in detail here, as trace on these components is only likely to be required if the IBM Service Center requests it. However, trace in the Message Brokers Toolkit and other advanced settings can be accessed through the Preferences window (in the Message Brokers Toolkit, select **Windows** → **Preferences**).

There are a variety of options for recording trace for Broker Administration in Preferences. Figure 8-32 on page 288 shows the Preferences window with some of the settings that can be altered for tracing connections for broker administration.
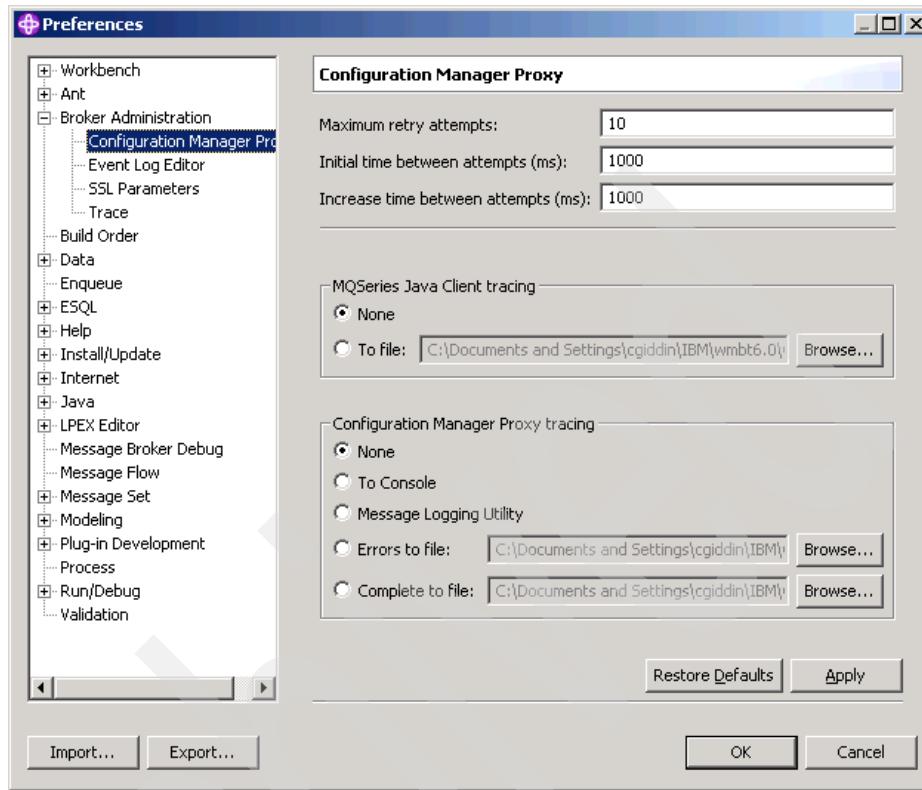
Figure 8-32   Preferences in the Message Brokers Toolkit

8.3.5 WebSphere MQ trace

Trace can be switched on within WebSphere MQ; however, this must be used with caution because it records trace for every executable running in WebSphere MQ and can lead to the production of very large log files and affect performance. Error logs are produced for WebSphere MQ without trace being set; check these first if problems are occurring in WebSphere MQ before switching on trace. These error logs can be located in the trace directory under the install path for WebSphere MQ (for example, C:\Program Files\IBM\WebSphere MQ\trace).

To switch on trace for WebSphere MQ:

1. Start the WebSphere MQ Explorer program.

2. Right-click **IBM WebSphere MQ** and select **Trace** from the context menu.

3. Click **Start** in the Trace window (Figure 8-33 on page 289), and click **OK**.
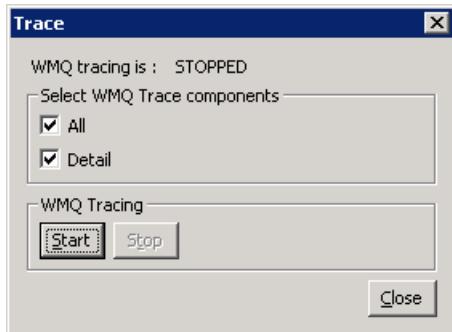
Figure 8-33   Starting trace on WebSphere MQ

To view the trace files that have been created, open the TRC files that are located in the trace directory under the install path for WebSphere MQ (for example, C:\Program Files\IBM\WebSphere MQ\trace).

Select **Stop** in the Trace dialog to stop trace from being recorded on WebSphere MQ components and programs after the required trace has been generated.

8.3.6 ODBC trace

ODBC trace can be useful if problems related to database connections occur when using message flows that access databases to query information, or update, delete, or insert data from tables. To start ODBC tracing on Windows:

1. Open the **ODBC Data Source Administrator** program from the Windows Control Panel. (It is usually located under Administrative Tools, but the location depends on the version of Windows that is installed on the system.)

2. Select the **Tracing** tab in the ODBC Data Source Administrator (Figure 8-34 on page 290).

3. Check the **Log file Path**. This can be changed to a different name and directory, if necessary.

4. Click **Start Tracing Now** to start tracing ODBC connections on the system.
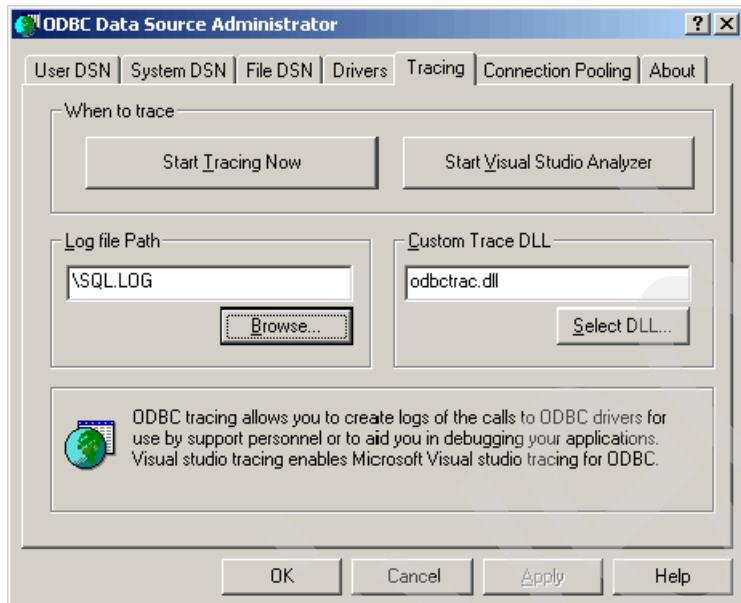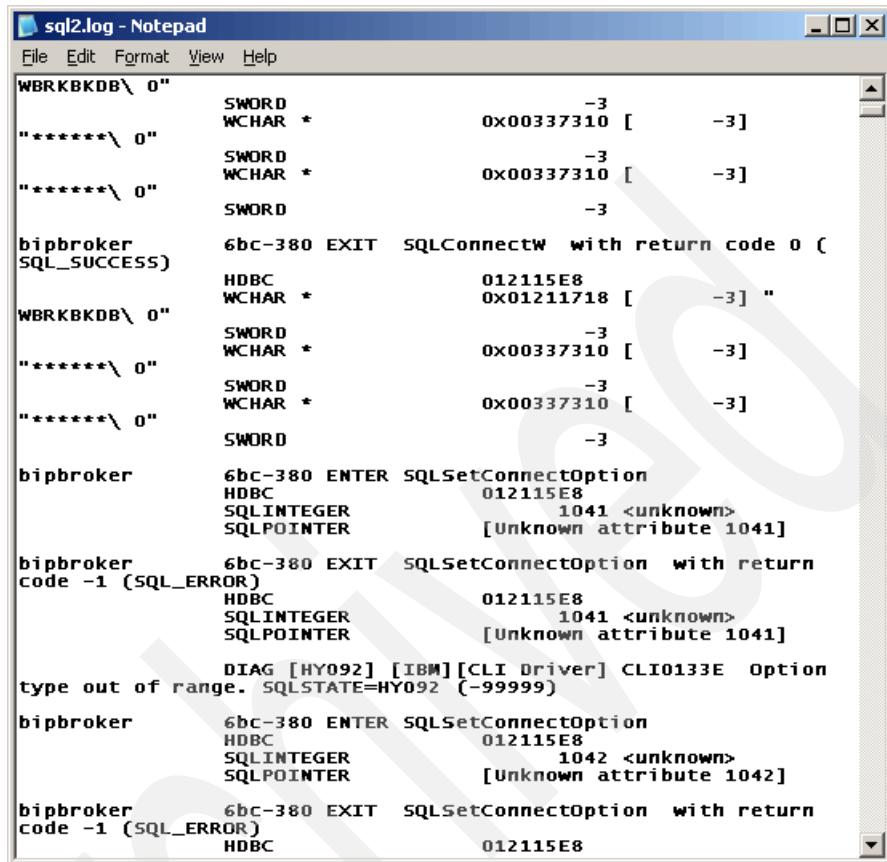
Figure 8-34   Starting ODBC trace

Click **Stop Tracing Now** in the ODBC Data Source Administrator to stop collecting ODBC trace.

Figure 8-35 on page 291 shows an example of the output that is produced in the ODBC trace file. The success of an ODBC/SQL action is indicated by the return code, where 0 (zero) is successful and the value -1 is an error condition.

Figure 8-35   Example contents of an ODBC trace log

8.4 Troubleshooting common problems

This section describes some basic, common problems that may be experienced
with WebSphere Message Broker, and suggestions for overcoming these issues.

8.4.1 Default Configuration wizard problems

The Default Configuration wizard may fail for a variety of reasons, but typically
because of an underlying problem with the WebSphere Message Broker
configuration that prevents the creation or starting of one of the components that
the Default Configuration wizard creates.

If the Default Configuration wizard fails, the message shown in Figure 8-36 is displayed. This message indicates which task the Default Configuration wizard was attempting when it failed. The Default Configuration wizard provides a Yes and a No button. Clicking the Yes button attempts to complete the task that failed again. Clicking No roll backs all of the tasks that the Default Configuration wizard has completed up to that point.
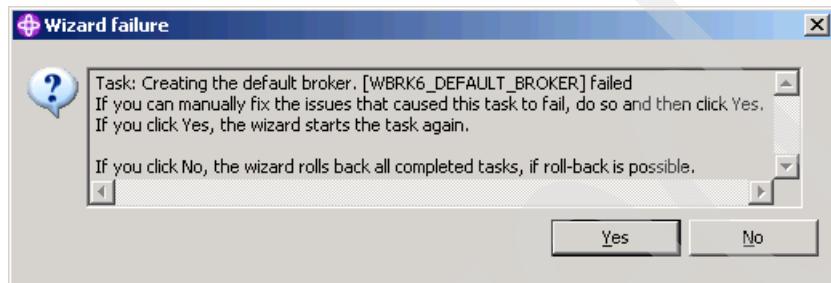
Figure 8-36   Error produced on Default Configuration wizard failure

The only way of finding out why the particular task has failed at this point is to consult the log that the Default Configuration wizard creates. This log can be found in the .metadata directory in the Message Brokers Toolkit workspace. The workspace directory is located by default here in Windows:

```
C:\Documents and Settings\wmbuser\IBM\wmbt6.0\workspace
```

Where *wmbuser* is the user ID for the user logged on to the machine. The log created by the Default Configuration wizard is called:

```
DefaultConfigurationWizard.log
```

This log can be consulted for errors generated while the wizard was run. An example of an error that may occur and its format in the DefaultConfigurationWizard.log is provided in Example 8-4. This shows that the cause of the error is that user name and password that the Default Configuration wizard is trying to use to create the broker is wrong.

Example 8-4   Excerpt of DefaultConfigurationWizard.log

```
D:\MessageBrokers>mqsicreatebroker WBRK6_DEFAULT_BROKER -i wmbuser -a
********** -q WBRK6_DEFAULT_QUEUE_MANAGER -n DEFBKDB6
BIP2321E: Database error: ODBC return code '-1'.
The message broker encountered an error whilst executing a database operation.
The ODBC return code was '-1'. See the following messages for information
obtained from the database pertaining to this error.
Use the following messages to determine the cause of the error. This is likely
to be such things as incorrect datasource or table names. Then correct either
the database or message broker configuration.
```

```
BIP2322E: Database error: SQL State ''08001''; Native Error Code '-30082';
Error Text ''[IBM][CLI Driver] SQL30082N  Attempt to establish connection
failed with security reason "24" ("USERNAME AND/OR PASSWORD INVALID").
SQLSTATE=08001  ''.
The error has the following diagnostic information:    SQL State
''08001''     SQL Native Error Code '-30082'      SQL Error Text
''[IBM][CLI Driver] SQL30082N  Attempt to establish connection failed with
security reason "24" ("USERNAME AND/OR PASSWORD INVALID").  SQLSTATE=08001  ''
This message may be accompanied by other messages describing the effect on the
message broker itself.  Use the reason identified in this message with the
accompanying messages to determine the cause of the error.

BIP8040E: Unable to connect to the database.
The database cannot be accessed with the userid and password that were
specified when the broker was created.
Check that the database is running, that an ODBC connection has been created
and that the userid and password pair specified for ODBC connect on the
mqsicreate command are capable of being used to connect to the database using
an ODBC connection. Also ensure that the database has a adequate number of
database connections available for use.
```

The conclusion from the errors shown in Example 8-4 on page 292 is that the password supplied to the Default Configuration wizard has been entered incorrectly.

Sometimes an error may be displayed in the DefaultConfigurationWizard.log that can be fixed by the user; for example, if an error was displayed indicating that no start database command was received, this would mean that the DB2 Universal Database service was stopped. This could be started and the Default Configuration wizard could continue from where it left of. In such a circumstance where you can fix the problem, click **Yes** after the problem is resolved to try re-running the failed task.

If a problem occurs that cannot be fixed by the user, then the **No** button must be clicked to roll back the tasks that the wizard has performed to continue. In the problem shown above, where the password has been entered incorrectly, we must click **No** to roll back all the tasks, so that the Default Configuration wizard can be restarted and the password re-entered.

8.4.2 Errors with the Message Brokers Toolkit

A variety of errors can occur in the Message Brokers Toolkit. Some problems caused by an internal problem with the underlying code can cause a message to be displayed on the screen, indicating that more information can be found in the Error Log. Typically these are Java or NullPointer errors and can occur in the

PDE Runtime Error Log. This can be accessed by using the following instructions:

1. In the Message Brokers Toolkit, select **Preferences** from the Window menu.

2. Check the Eclipse Developer box in the Capabilities section, as shown in Figure 8-37.

3. Click **OK**.
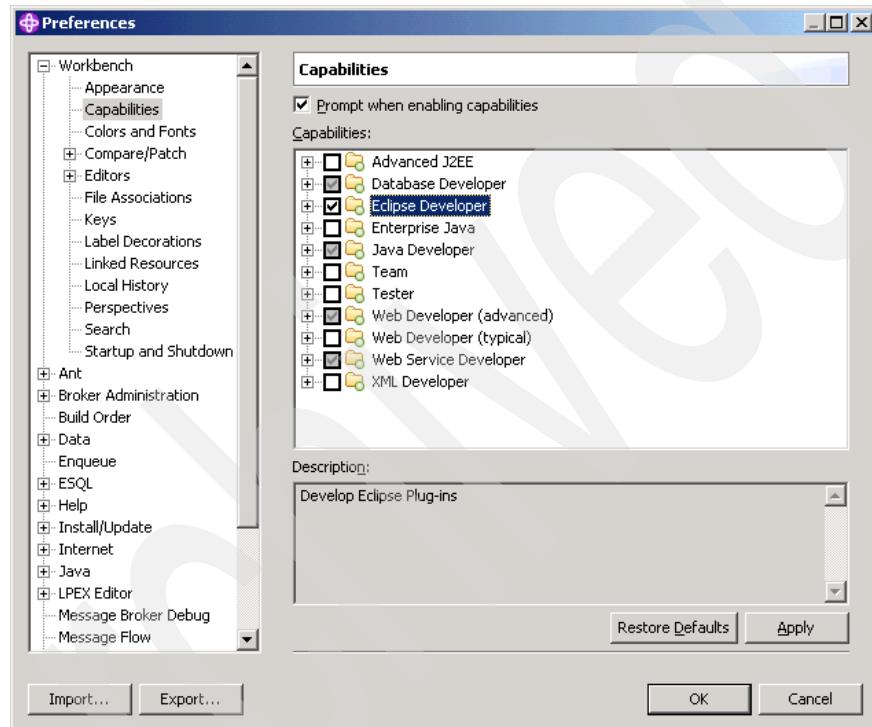
Figure 8-37   Adding the Eclipse Developer capability

4. Select **Show View** → **Other** from the Window menu.

5. Select **Error Log** from PDE Runtime in the choice of views, as shown in Figure 8-38 on page 295.

Information in this log may be useful to service to help track down the cause of a problem, or where more than one product is using the Rational Application Development platform, which product may be producing errors.
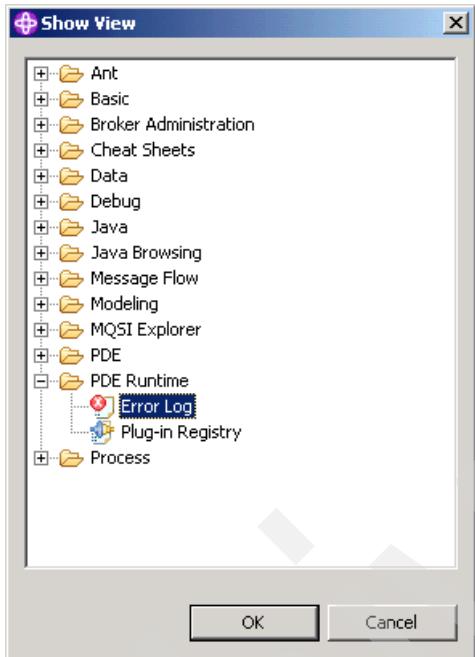
Figure 8-38   Selecting the PDE Runtime Error Log

Other problems seem to cause strange behavior in the Message Brokers Toolkit, such as an editor no longer opening correctly. Other problems that may occur are that the perspectives and user settings in the Message Brokers Toolkit cannot be saved on shutdown or be restored during startup of the Eclipse workbench. Errors may also occur if the Eclipse Workbench is terminated unexpectedly. For any such errors that are not resolved through normal use of the Message Brokers Toolkit, the following method is useful to try to resolve the problem.

1. Close the Message Brokers Toolkit.
2. Open a command window.
3. Navigate to the install directory of the Message Brokers Toolkit, for example, C:\MessageBrokers.
4. Type the following onto the command line:

   ```
   wmbt.exe -clean
   ```

The Message Brokers Toolkit restarts, and often this solves the problem. If this does not resolve the problem then check for errors in the PDE Runtime Error Log.

Unexplained errors in message flows and message sets

Some unexplained errors can occur where a message flow or message set that was working correctly suddenly shows unexplained errors. This may be due to a change in a referenced project or for no clear reason. An example error message is `Unable to find element reference`. A potential way to solve this problem is to clean the projects in the Message Brokers Toolkit's workspace. Use the instructions below to clean the projects in the Message Brokers Toolkit's workspace.

1. Select **Clean** from the Project menu.

2. Select to either clean all projects, or select the project to clean by clicking **Browse** (an example is shown in Figure 8-39).

3. Click **OK**.
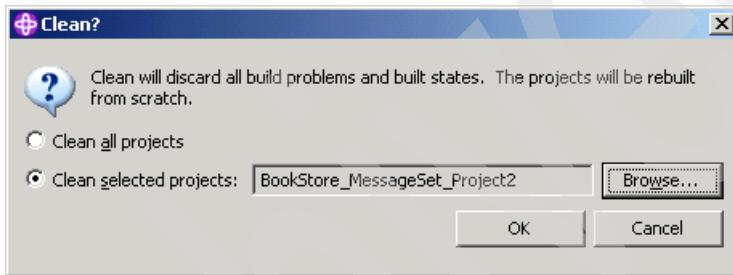
Figure 8-39   Clean projects

This rebuilds the projects, often removes unexplained error messages, and rebuilds the links with referenced projects. Cleaning all projects is useful where there are only small numbers, but the clean takes some time with many projects. If individual projects are selected then any projects referenced to the project showing the problems should also be cleaned.

If these steps do not solve a particular problem that is occurring in the Message Brokers Toolkit, you should seek further assistance.

8.4.3 Problems connecting to the Configuration Manager

The Message Brokers Toolkit must communicate with the Configuration Manager in order to perform any types of deploy task. The properties for the connection to the Configuration Manager are set up in the Domain Connection wizard. The properties are stored in a file with a .configmgr extension and, if the properties change, they can be edited by double-clicking the Domain Connection file.

Figure 8-40 shows the domain connection when the connection between the Message Brokers Toolkit and the Configuration Manager has not been established. The domain connection and the editor options beneath it are grayed out.
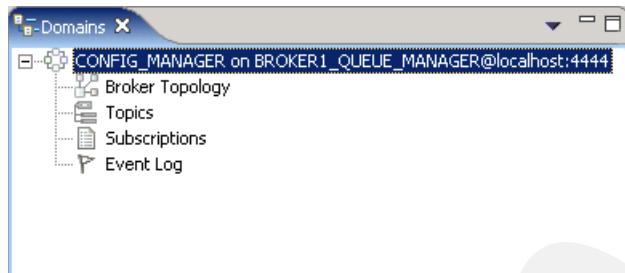
Figure 8-40   Disconnected broker domain

To establish the connection, right-click the domain connection and select **Connect** from the context menu. This makes the Message Brokers Toolkit attempt to connect to the Configuration Manager, and a progress bar is displayed. If the progress bar seems to stop or take a long time, there is likely to be a problem with the connection to the Configuration Manager.

If a failure occurs during connection to the Configuration Manager, then a BIP error message is displayed with the cause or possible causes of the problem, as seen in Figure 8-41.

Figure 8-41   Error message: Communication problem with Configuration Manager

Use the error information in the Details section of the BIP error message to help track down the cause of the problem. Some common solutions to problems with the Message Brokers Toolkit's connection to the Configuration Manager are:

► Confirm that the Configuration Manager is created.

► Confirm that the Configuration Manager is started successfully, as indicated by a BIP1003 message in the Application log in the Windows Event Viewer.

► Confirm that the WebSphere MQ queue manager is available.

► Confirm that the WebSphere MQ queue manager that is specified in the Domain Connection configuration is correct.

► Confirm that the WebSphere MQ Listener is running.

► Confirm that nothing else is running on the same port as the WebSphere MQ Listener.

► Confirm that the WebSphere MQ Listener port that is specified in the Domain Connection configuration is correct.

► Ensure that the user that is running the Message Brokers Toolkit has the appropriate security authorities to access and run the Configuration Manager.

► If the Configuration Manager is on a remote machine there may be a time delay for the communication between the Configuration Manager and the Message Brokers Toolkit. Use the instructions in Figure 8-41 on page 297 to increase the number of retry attempts and increase the time between attempts. The preferences options are shown in Figure 8-42 on page 299.

► If the Configuration Manager is on a remote machine check the Application log or syslog for error messages.
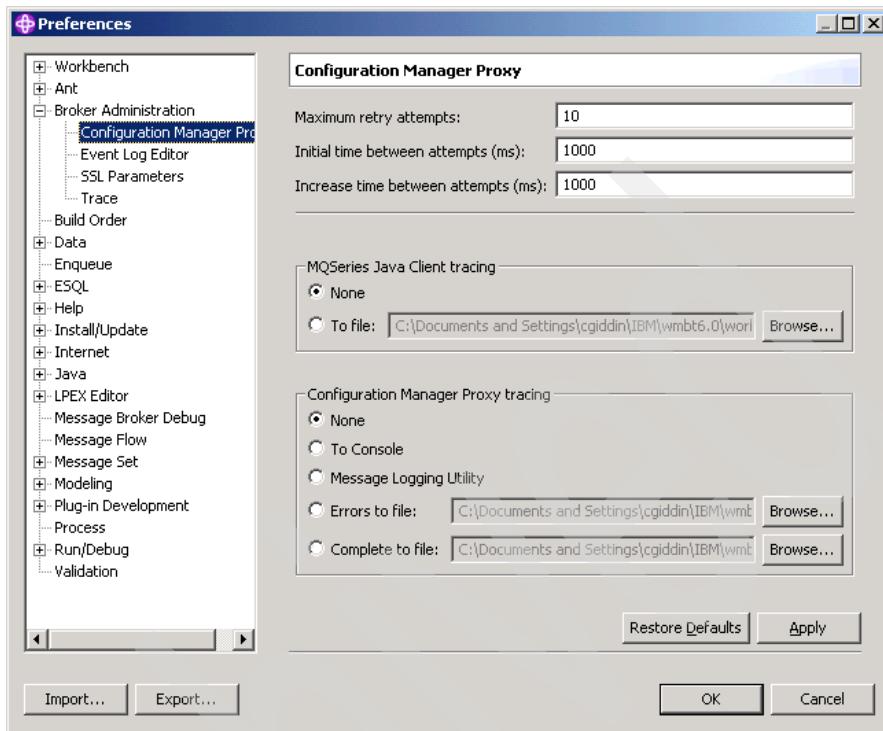
Figure 8-42   Preferences for communication with the Configuration Manager

8.4.4 Problems with deployment

A variety of problems can occur with deployment actions, and this section details a few of the common problems and solutions.

Deleted brokers

If a broker is deleted on a system and then re-created with the same name, the broker is not recognized by the Configuration Manager on the next deploy operation to it. This is because each broker and execution group on its first deploy is allocated a unique ID. This ID is referred to in subsequent deploys, and if the broker is re-created, the ID is no longer present.

If the broker is deleted and re-created without being deleted from the Configuration Manager, then BIP2062 and BIP2087 error messages are seen in the Message Brokers Toolkit Event Log in the next deploy operation.

To remedy this situation:

1. In the Broker Administration perspective in the Message Brokers Toolkit, right-click the re-created broker and select **Remove Deployed Children** from the context menu.

2. This displays successful configuration change messages in the Event Log (for example, BIP2056 and BIP4045). An error message from the Configuration Manager may also be present (BIP1536).

3. Deploy to the default execution group on the broker.

The execution group and broker are then assigned an ID and can be deployed to as normal.

If the Configuration Manager and its database have been re-created, then you must re-create any brokers in order to be able to deploy to them.

Using the Configuration Manager Proxy API Exerciser

If difficulties still occur with a deleted broker, such as the Configuration Manager continuing to attempt to deploy resources to the broker, the Configuration Manager Proxy API Exerciser sample can be used to help. This sample allows you to view and manage a broker domain by using the Configuration Manager Proxy API.

To start the Configuration Manager Proxy API Exerciser sample:

1. Click **Start → IBM WebSphere Message Broker V6.0 → Java Programming APIs → Configuration Manager Proxy API Exerciser**.

2. Connect to a running Configuration Manager by clicking **File → Connect to Configuration Manager**.

3. Enter the connection parameters to the Configuration Manager.

4. Click **Submit.**

Information about the broker domain is retrieved and displayed in the Configuration Manager Proxy API Exerciser window.

To execute a Configuration Manager Proxy API method against a broker object, right-click a broker in the navigation tree view. A list of the available methods is displayed. Selecting one of these methods executes the method.

> **Attention:** The Configuration Manager Proxy API is very powerful. As well as performing useful administration tasks, there are options to alter areas of the broker domain configuration that are not usually available. These should not be used in an environment except for a test environment before an understanding of the methods and the effect that has on the broker domain has been developed by using the WebSphere Message Broker documentation. Inappropriate use of the methods can lead to components that can no longer be administered, requiring deletion and recreation.

For the problem in this example useful methods are Cancel all outstanding deploys to this broker, Delete broker, and Delete broker configuration.

Remote broker not responding

If a deploy to a remote broker fails or no response is received, these actions may help to solve or determine the cause of the problem.

- ► Confirm that the sender and receiver channels on the remote broker's queue manager are running.

- ► Confirm that the sender and receiver channels on the Configuration Manager's queue manager are running.

- ► Check the Event Log in the Message Brokers Toolkit for messages from the Configuration Manager or the broker.

- ► Check the Windows Event Viewer for error messages from the Configuration Manager or from WebSphere MQ.

- ► Check the local log on the broker system for error messages from the broker or from WebSphere MQ.

Outstanding deploys

On occasion, an unprocessed deploy message can block other deploys from the Configuration Manager to a broker. This situation is resolved easily:

1. In the Broker Administration perspective in the Message Brokers Toolkit, right-click the **Domain Connection**.

2. Select **Cancel Deployment** from the context menu.

This clears any deployment messages from the Configuration Manager on the broker queues. WebSphere MQ channels must be running on any broker queue managers for any broker not on the same queue manager as the Configuration Manager.

The Configuration Manager API Exerciser can be used to cancel all outstanding deploys, or the outstanding deploys for a single broker.

8.4.5 Messages stuck on the input queue

Messages may remain on the input queue of a message flow, usually for any of these reasons:

► Message flow is not running, perhaps because the message flow or broker is stopped or the deploy of the message flow failed. In this case, check the status of the message flow in the Message Brokers Toolkit, and the results of any deploys in the Event Log.

► There could also be issues such as a spelling mistake in the input node of the message flow, so the message flow is attempting to pick the messages up from a different queue. Any error or warning messages in the Application log help to determine this kind of error.

► An error has occurred in the message flow and a message has been rolled back onto the input node. This message blocks any other messages that are put to the queue. See "Configuring the ESQL_Simple message flow" on page 58 for how to resolve this problem by defining a backout queue.

8.4.6 Common DB2 Universal Database Errors

In depth discussion of database problems is beyond the scope of this book, but a couple of the more basic problems and their solutions are provided here to assist new users.

Database not available

A broker may fail to become available after being started if a problem exists with the database. A sequence of error messages is generated in the Application log indicating the cause of the problem. This sequence of messages is repeated every thirty seconds or so while the broker repeatedly continues to connect to the database. There are two common causes of this problem, either that the database is not started or that the user ID and password are no longer valid for the database.

Figure 8-43 on page 303 shows an example of a BIP2322 message indicating that no start database manager command was issued. This means that the database is not running. To solve this problem enter db2start on a command line to check that the database started correctly. If any errors are displayed, then seek assistance from a database administrator or the database documentation.
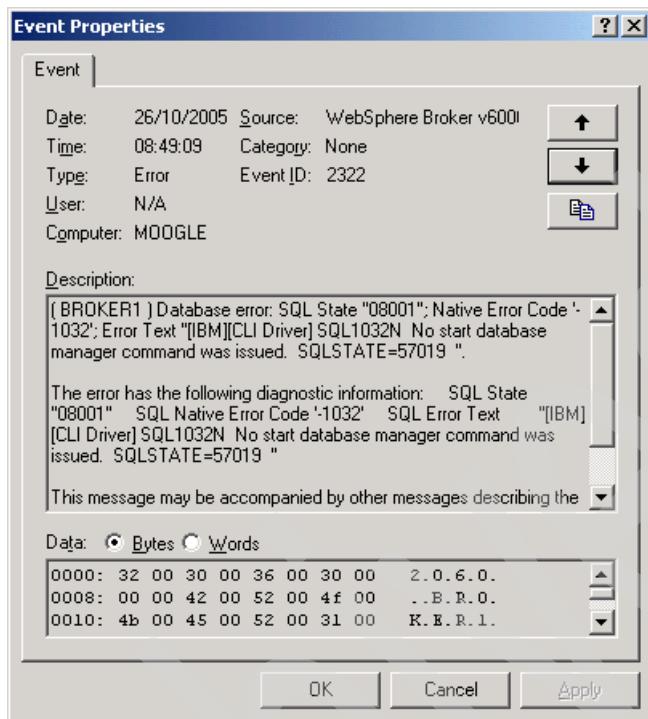
Figure 8-43   Database connection error message

Monitor heap size

Problems may be seen with message flows or WebSphere Message Broker brokers when they attempt to connect to a database. This can be in a message flow that is performing a database action, or in a broker at start up or deploy. With DB2 Universal Database for the broker database, these problems are often associated with the monitor heap size. Errors indicate that the memory for the Monitor heap is running low or has run out.

The amount of memory allocated for the monitor heap can be altered, and if this becomes a recurring problem then the memory allocation should be increased for the monitor heap. To do this use the instructions below:

1. Select **Start** → **Programs** → **IBM DB2** → **Set-up Tools** → **Configuration Assistant**.

2. Select **DBM Configuration** from the Configure menu.

3. Locate the Performance section and select the **MON_HEAP_SZ** parameter.

4. Select the Value column for the MON_HEAP_SZ and click the '**...**' button.

5. Enter a new value for the amount of memory to allocate for database system monitor data, for example, 6000.

6. Click **OK**.

7. Click **OK** to accept the change.

Figure 8-44 shows the DBM Configuration with a pending change to the MON_HEAP_SZ parameter. The change to the configuration does not occur until the database is rebooted. Use the `db2stop` and `db2start` commands on the command line to restart DB2 Universal Database.
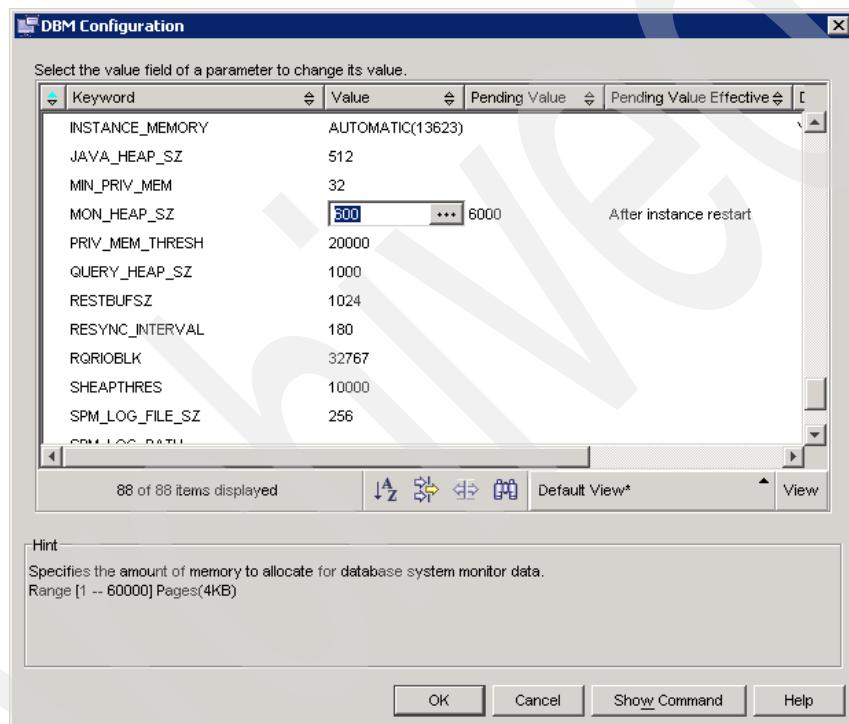
Figure 8-44   DBM configuration

8.4.7 Further information for troubleshooting

The WebSphere Message Broker provides a number of sections in the WebSphere Message Broker Information Center that are devoted to troubleshooting and support topics. Check here for further common problems and solutions and how to recover from failure, and contact service.

The general troubleshooting and support section of the WebSphere Message Broker Information Center can be found here:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp?topic=/com.ib m.etools.mft.doc/au03830_.htm

Information about contacting the IBM Support Center and the information to have ready can be found here:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp?topic=/com.ib m.etools.mft.doc/an01330_.htm

Searching for information about WebSphere Message Broker and common problems using the WebSphere Message Broker Information Centre can be found here:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp?topic=/com.ib m.support.wsbi.doc/html/search.html

Reference material for troubleshooting from the WebSphere Message Broker Information Center can be found here:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp?topic=/com.ib m.etools.mft.doc/au09090_.htm

A

Getting help

This appendix describes:

- ► Getting context-sensitive help
- ► Using the product documentation
- ► Finding information in the product documentation
- ► Updating the information center
- ► Getting information from other sources
- ► Serving the help system from a central location
- ► Useful links

307

Message Brokers Toolkit help

In addition to the product documentation that is discussed in the next section, there is built-in help in the user interface in the form of context-sensitive help.

Getting context-sensitive help

To get context-sensitive help from within the Message Brokers Toolkit:

1. Bring focus to any part of the user interface. For example, to view context-sensitive help in the Message Brokers Toolkit, open the Message Flow editor, then click the MQOutput node in the node palette.

2. Press F1. A yellow box, an infopop, is displayed. Each infopop gives some high-level help, followed by a short list of links.

Figure A-1   Infopop for MQOutput node in the Node Palette

3. If you need more information, you can click one of the links that are displayed below the context-sensitive help details. Clicking one of these links opens the Message Brokers Toolkit help system at the relevant topic.

Using the product documentation

The product documentation exists within the Message Brokers Toolkit help system. This section describes the ways in which you can obtain and view the product documentation and how the documentation is structured.

Viewing the product documentation

All of the product documentation for WebSphere Message Broker V6.0 is in the Message Brokers Toolkit help system. To open the help system from within the Message Brokers Toolkit you can do either of the following:

► Click a topic link in an infopop. This opens the help system at that particular topic.

► Click **Help** → **Help Contents**.

In addition to launching the product documentation through the Message Brokers Toolkit, you can use the resources provided on the documentation CD (included with the product package). This CD contains a standalone information center.

To open the standalone information center, follow the instructions that are in the installing_and_managing.html file, which is located in the root directory of the documentation CD.

Structure and content of the product documentation

When you open the help system, the available documentation is displayed in the Contents pane on the left. The Contents pane lists all of the documentation that is included with WebSphere Message Broker V6.0 (Figure A-2 on page 310).

The WebSphere Message Broker documentation is organized by the high-level tasks that you are likely to want to perform, such as installing, configuring, and administering a broker domain, or developing applications. There are ten main sections, where each section corresponds to one of these high-level tasks.

In the ten main sections, product information is organized according to information type; there are concept topics, which provide an overview of the subject area, and task topics, which offer specific guidance on how to complete various tasks. In each main section and subsequent subsection, concept topics are listed before the task topics. If you are new to one of the sections, you will find it useful to read the concept topics first.

In addition to the ten main sections, the following sections also exist: Product overview, Samples, Reference, Glossary, Feedback, Index, and Diagnostic messages.

Figure A-2   Contents pane

If you are new to WebSphere Message Broker, start by reading the Product overview and the Samples sections. This will give you a good introduction to the product, and introduce you to some of the product's capabilities.

You might also find it useful to read the information about `Navigating and customizing the workbench,` which describes how to work with the basic framework in which the Message Brokers Toolkit is built; for example, you can rearrange the views within perspectives.

Finding information in the product documentation

There are several different ways to find information within the product documentation; for example, you can navigate through the documentation from the Contents pane and follow related links from individual topics. You can also use the Search functionality, use the diagnostic messages search tool, and use the Index.

Searching for information

An alternative to navigating the help system in the Contents pane or by following related links is to use the Search facility. The Search field is located near the top left of the help system window.

Two different levels of search are available:

► Search all the contents. This searches the contents of all the documentation that is installed with the Message Brokers Toolkit, not just the WebSphere Message Broker product documentation.

► Specify a search scope. This narrows the scope of the search to specific sets of documentation.

Using the search scope

The search scope allows you to narrow your search field to specific sets of documentation or individual sections within the documentation. Initially, the search scope is set to `All topics`. However, you can create and save new search scopes.

To create a new search scope:

1. Click **Search scope**. The Select Search Scope dialog opens (Figure A-3).
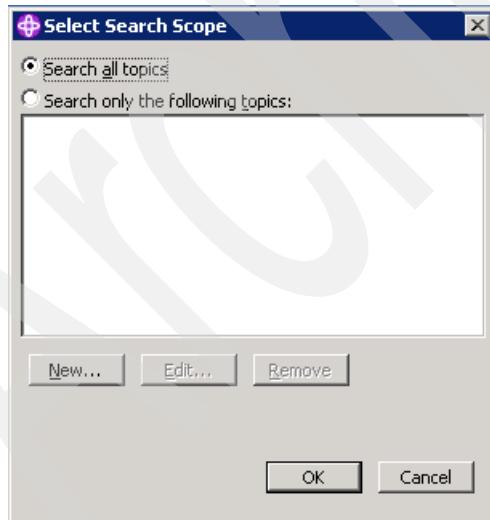
Figure A-3   The Select Search Scope dialog

2. Click **New** to create a new search scope. The New Search List dialog opens.

3. In the New Search List dialog, select the check boxes that correspond to information that you want to include in the search. Expand a section if you want to select specific subsections (Figure A-4).
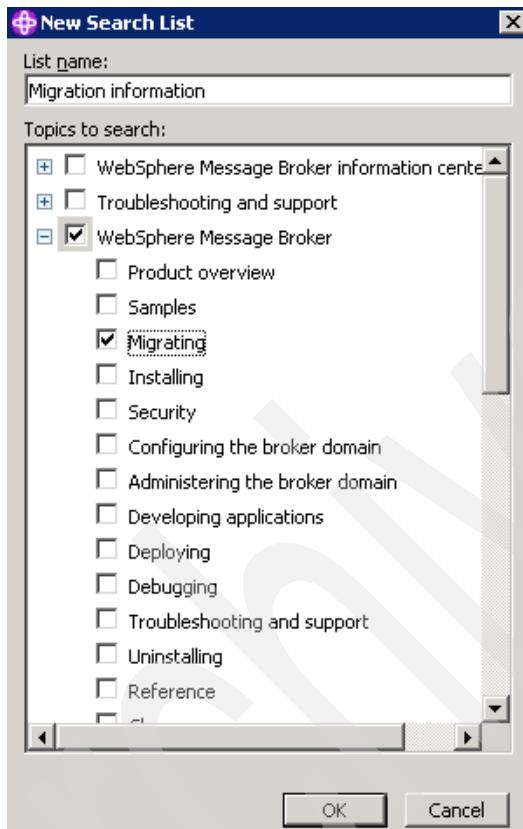
Figure A-4   The New Search List dialog

4. In the List name field, type a name for the search scope and click **OK**. The new search scope is now displayed as an option in the Select Search Scope dialog.

5. Select the search scope that you require, and click **OK**. The name of the search scope that you have selected is displayed after the Search field (Figure A-5).

Figure A-5   The Search field, with the search scope set to Migration information

You can create multiple search scopes but you can only search under one search scope at a time.

Searching on more than one word

If more than one word is typed into the Search field, the Results pane lists topics that contain all of the words. For example, if you enter the words *configure* and *broker*, the search results list only those topics that contain both configure and broker in their content. The Boolean AND operator between the words is implied unless you use another operator, such as OR.

More information about searching is available in the section labelled "WebSphere Message Broker information center home."

Diagnostic messages

The Diagnostic messages tool allows you to search for information about specific BIP messages. Enter the BIP message number in the search field (see Figure A-6) and press Enter. Information about the BIP message is displayed below the diagnostic messages search field.

Diagnostic messages

Enter message number:
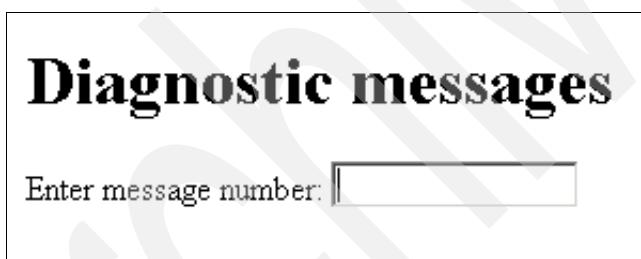
Figure A-6   The Diagnostic messages search utility

Using the Index

The Index contains a set of alphabetically organized links, where each link takes you directly to a topic in the product documentation. This index is designed to be used in a manner similar to the index of a book.

Entries in the Index that are underlined are links to specific topics. Entries that are not underlined do not link to individual topics but are included to provide a reference point for sub-entries. For example, in Figure A-7 on page 314, `administration` is not a link, but is included to group together links that relate to administration.
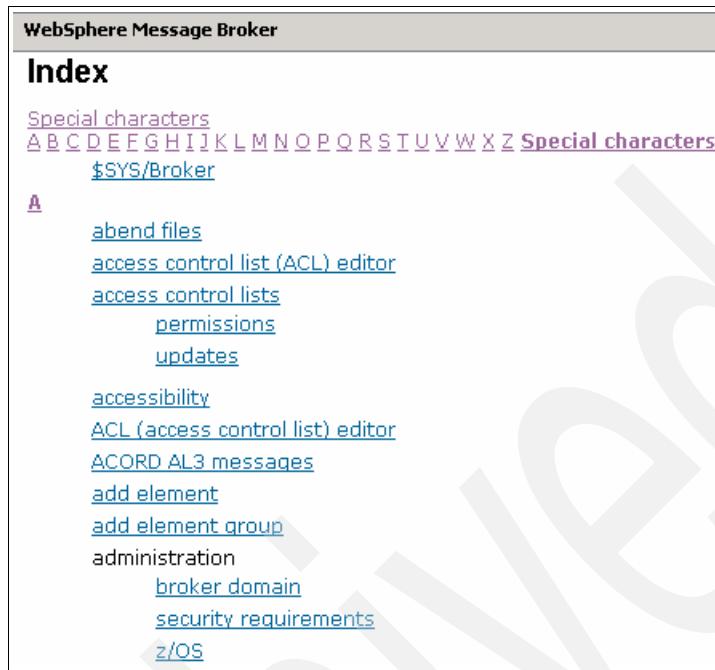
Figure A-7   The Index for WebSphere Message Broker product documentation

Orienting yourself in the help system

If you arrive at a topic by following a link, such as clicking one of the links that is displayed in the infopops, the Contents pane does not automatically display the topic location.

To view the location of a topic in the Contents pane, click the **Show in Table of Contents** button at the top-right of the help system window (the fourth item from the right in Figure A-8). This figure also shows these Help system navigation buttons: Go Back, Go Forward, Show in Table of Contents, Bookmark Document, Print Page, and Maximize.
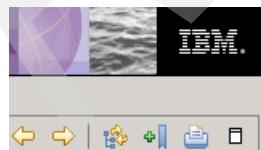
Figure A-8   Help system navigation buttons

Updating the product documentation

You can configure the Message Brokers Toolkit to automatically check for updates to WebSphere Message Broker, including the product documentation. Alternatively, you can manually check for updates from the Message Brokers Toolkit.

Receiving automatic updates

To receive automatic updates to the product documentation through the Message Brokers Toolkit:

1. In the Message Brokers Toolkit, click **Window → Preferences**. The Preferences dialog opens.

2. In the Preferences dialog, expand **Install/Update**, then click **Automatic Updates**.

3. On the Automatic Updates page, select the **Automatically find me new updates and notify me** check box, then select the frequency with which you want the Message Brokers Toolkit to check for new updates.

4. Click **OK**.

The Message Brokers Toolkit will automatically check for updates to the Message Brokers Toolkit and to the WebSphere Message Broker product documentation.

Receiving manual updates

To manually check for updates to the Message Brokers Toolkit and product documentation:

1. In the Message Brokers Toolkit, click **Help → Software Updates → Find and Install**. The Install/Update wizard opens.

2. In the Install/Update wizard, ensure that **Search for updates of the currently installed features** is selected, then click **Next**.

3. The wizard contacts its configured update sites, including the Message Brokers Toolkit Update Site. If there are any updates available, they are displayed in the wizard.

4. When documentation updates are available, select the updates and follow the instructions in the wizard to install them and restart the Message Brokers Toolkit.

The new documentation updates are installed in the Message Brokers Toolkit help system. Next time you perform a search in the help system, the

documentation is re-indexed so that the updated documentation is included in the search.

Updating the documentation in information centers

When the product documentation is available in either a standalone information center (such as the one on the Documentation CD) or centrally served information center (such as the one available online; see the links at the end of this appendix), you must manually download the updates as documentation plug-ins or as the updated information center, depending on the updates.

See the links at the end of this appendix.

Getting help from other sources

The Troubleshooting and support section within the WebSphere Message Broker documentation, contains an overview of the resources that are available to help solve problems that you might have when using WebSphere Message Broker. It also suggests the type of information that you should collect to help IBM Service to diagnose and fix problems.

In addition to the troubleshooting information, there are links to newsgroups, Web sites, and a search interface to help you find information in the WebSphere Message Broker support documents on the Web.

Serving an information center from a single location

You might want to consider serving the WebSphere Message Broker V6.0 help system from a central location within your organization if, for example:

► A large number of people within the organization need regular access to the product documentation and to updates for the documentation.

► Many of the people within the organization who regularly use the product documentation do not have Internet access and so cannot easily obtain and install updates.

By installing the WebSphere Message Broker information center on a central server, anyone with access to that server can view the information by using an ordinary Web browser. The main benefit is that updates to the product documentation can be applied to a single instance of the information center so that everyone in the organization is viewing up-to-date information.

Be aware that if users are accessing the documentation from a central sever, links in the documentation that launch graphical interface actions (such as starting a wizard) no longer work.

For instructions on installing and configuring the help system as a centralized information center, in the Message Brokers Toolkit, open the **Help**, then click **Navigating and customizing the workbench** → **Extending the workbench** → **Reference** → **Other reference information** → **Installing the help system as an infocenter**.

Useful links

When available, you can download fix packs from the WebSphere Message Broker support pages. For previous versions of WebSphere Message Broker these are located at:

http://www-306.ibm.com/software/integration/mqfamily/support/summary/wbib.html

View the latest readme file for WebSphere Message Broker V6.0:

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

View the latest product documentation for WebSphere Message Broker V6.0, including the Installation Guide:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp

View the latest list of required software for WebSphere Message Broker:

http://www.ibm.com/software/integration/wbimessagebroker/requirements/

View the latest list of required software for WebSphere Event Broker:

http://www.ibm.com/software/integration/wbieventbroker/requirements/

Download fix packs for WebSphere MQ Family products:

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

Download Service Packs for Microsoft Windows:

http://www.windowsupdate.com

You can obtain DB2 fix packs on CDROM or by downloading from the Web.

http://www.ibm.com/software/data/db2/udb/support.html

Note, however, that fix packs might be large. To avoid lengthy downloads, request CD-ROM versions. If you have a current support contract, you can order

DB2 fix packs on CD-ROM by calling DB2 support. Contact details are provided at:
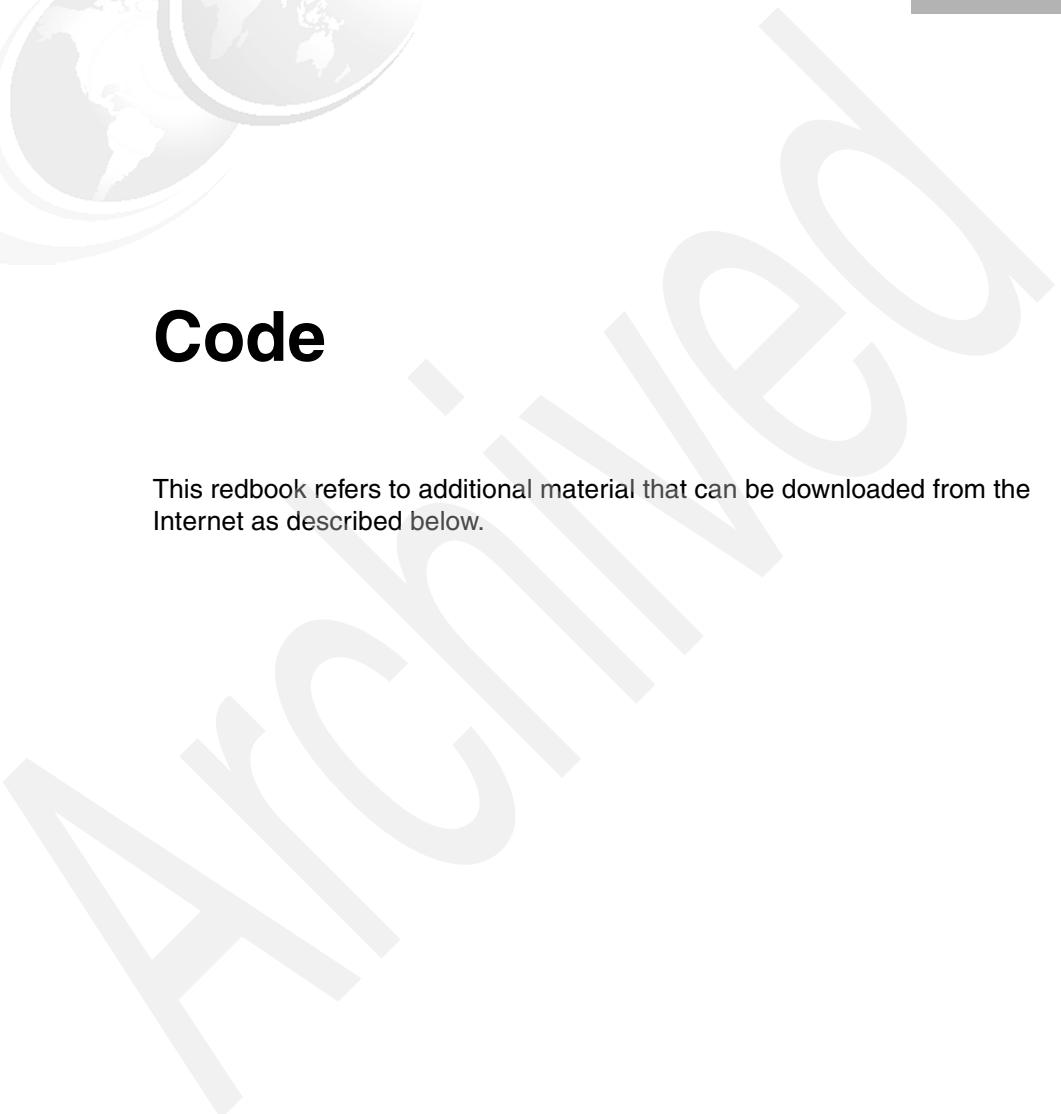
► Download a standalone version of the help system.

  – For Linux:

    `ftp://ftp.software.ibm.com/software/integration/wbibrokers/docs/V6.0/wmb_help_lin.zip`

  – For Windows:

    `ftp://ftp.software.ibm.com/software/integration/wbibrokers/docs/V6.0/wmb_help_win.zip`

► Download the WebSphere MQ Family manuals in PDF format:

  `http://www-306.ibm.com/software/integration/mqfamily/library/manualsa/manuals/crosslatest.html`

► Download platform-specific WebSphere MQ Quick Beginnings guides:

  `http://www-306.ibm.com/software/integration/wmq/library/`

B

Code

This redbook refers to additional material that can be downloaded from the Internet as described below.

319

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/SG247137`

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, SG247137.

Using the Web material

The additional Web material that accompanies this book includes the files listed in Table B-1.

Table B-1   The Web material for Chapter 4, Chapter 5, and Chapter 6

File name	Description
SG247137 all.zip	Contains the pre-built message flow applications, as well as all the code and messages for developing the applications from scratch
SG247137 projects.zip	Contains the pre-built message flow applications
SG247137 resources.zip	Contains the code and messages for developing the message flow applications in Chapters 4–6
SG247137 ch4 ESQL Bookstore Message Flow Project.zip	Contains the pre-built ESQL_Bookstore Message Flow project for Chapter 4
SG247137 ch4 ESQL Simple Message Flow Project.zip	Contains the pre-built ESQL_Simple Message Flow project for Chapter 4
SG247137 ch5 Java Bookstore Message Flow Project.zip	Contains the pre-built Java_Bookstore Message Flow project for Chapter 5
SG247137 ch5 Java Bookstore Message Flow ProjectJava.zip	Contains the pre-built Java_Bookstore Message Flow project for Chapter 5
SG247137 ch5 Java Simple Message Flow Project.zip	Contains the pre-built Java_Simple Message Flow project for Chapter 5

File name	Description
SG247137 ch5 Java Simple Message Flow ProjectJava.zip	Contains the pre-built Java_Simple Message Flow projectJava for Chapter 5
SG247137 ch6 Mapping Bookstore Message Flow Project.zip	Contains the pre-built Mapping_Bookstore Message Flow project for Chapter 6
SG247137 ch6 Mapping Bookstore Message Set Project.zip	Contains the pre-built Mapping_Bookstore Message Set Project for Chapter 6
SG247137 ch6 Mapping Simple Message Flow Project.zip	Contains the pre-built Mapping_Simple Message Flow project for Chapter 6
SG247137 ch6 Mapping Simple Message Set Project.zip	Contains the pre-built Mapping_Simple Message Set Project for Chapter 6

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the SG247137 all.zip file into this folder. The folder now contains two zip files: SG247137 projects.zip and SG247137 resources.zip.

Developing the applications in Chapters 4–6

The SG247137 resources.zip file contains the following resources:

► The ESQL and Java code that you need to develop the applications in Chapters 4 and 5

► The BookStoreDB.sql file that you need to create and populate the DB2 database and tables for developing the Bookstore scenarios in Chapters 4–6

► The input messages that you need to test the message flow applications that you develop in Chapters 4–6.

Unzip the SG247137 resources.zip file to a directory on your workstation so that you can use the files when prompted in Chapters 4–6.

Exploring the pre-built message flow applications

The SG247137 projects.zip file contains the complete projects for the message flow applications that are referenced in Chapters 4–6.

To import and deploy the supplied projects in the Message Brokers Toolkit:

1. Unzip the SG247137 projects.zip file to a folder on your workstation. The folder now contains the 10 message flow application projects to import into your workspace.

2. Unzip each project.

3. Import each project into the Message Brokers Toolkit. For each project:

    a. In the Message Brokers Toolkit, click **File** → **Import**.

    b. Select **Existing Project into Workspace**, then click **Next**.

    c. Browse to the workspace directory and select one of the project folders to import, then click **Finish**.

    The project is imported into the Message Brokers Toolkit and displayed in the Resource Navigator view.

If you want to deploy and run the message flow applications, follow the instructions in the relevant chapter of this book to create the WebSphere MQ local queues and deploy the message flow application.

Glossary

Application log. A log viewable in the Windows Event Viewer that displays event messages from software on a machine.

Bar file. Abbreviation for message broker archive file, used to store compiled message flows, message sets and other code to be deployed to an execution group.

BIP message. An event message produced by WebSphere Message Broker, identifiable by its BIP number, for example BIP1003.

Breakpoint. Used as a point to stop the flow of a message in a message flow when the Flow Debugger is attached.

Broker. A broker is a set of execution processes that host and run message flows.

Broker Administration perspective. This is the perspective in the Message Brokers Toolkit, which is used for administering and monitoring objects in the broker domain. This perspective is also used for changing the configuration and message flow deploy operations.

Broker Application Development perspective. This is the perspective in the Message Brokers Toolkit in which message flows and message sets can be developed.

Broker database. A database that stores configuration for a broker. Multiple brokers can share the same database.

Broker domain. A group of brokers that share a common configuration and managed by a single Configuration Manager.

Broker Topology editor. .An editor in the Message Brokers Toolkit for configuring the properties of brokers in the domain.

Command console. This is a command-line interface, that sets up a suitable environment for running WebSphere Message Broker commands.

Compute node. A node in a message flow for processing messages using ESQL. Usually used for message transformation.

Configuration Manager. The Configuration Manager stores the configuration data for the broker domain that it manages, and performs the deployment operations between the Message Broker Toolkit and the brokers in the domain.

Configuration Manager Proxy API. TA programming interface for performing administration operations on WebSphere Message Broker components.

Database node. A node in a message flow for performing database operations using ESQL.

DataDelete node. A node in a message flow that uses message mappings to delete data in a database based on the contents of an input message.

DataInsert node. A node in a message flow that uses message mappings to insert data in a database from the contents of an input message.

DataUpdate node. A node in a message flow that uses message mappings to update data in a database from the contents of an input message.

DB2 Enterprise Server. A database which is supported for use as a broker database and is supplied with WebSphere Message Broker.

Dead letter queue. A WebSphere MQ queue that holds messages that were put back onto an input queue by a message flow.

Debug perspective. This is the perspective in the Message Brokers Toolkit used for debugging message flows and the Java, ESQL or mapping code associated with them.

Default Configuration Wizard. A wizard which creates a simple broker domain for verifying a WebSphere Message Broker installation. This configuration can be used for test purposes and using the samples.

Domain connection. A reference to a broker domain in the Message Brokers Toolkit.

Enqueue file. A file in the Message Brokers Toolkit used to put simple messages on to a WebSphere MQ queues.

ESQL. ESQL is Extended Structured Query Language and is used in the transformation of messages in message flows. It is also used to perform database operations such as querying or updating a database.

ESQL editor. An editor within the Message Brokers Toolkit for creating and editing ESQL.

Event Log. An editor in the Message Brokers Toolkit showing event messages generated as a result of deployment operations and changes to the broker domain configuration.

Event messages. Messages produced by software on a machine indicating a specific event or error.

Execution groups. An execution group represents a collection of message flows within a broker.

Input node. A node that takes a message from a source and inputs it to the message flow for processing.

Java. An object-oriented programming language used for programming the JavaCompute node or user defined nodes in the Message Brokers Toolkit.

Java editor. An editor in the Message Brokers Toolkit for developing Java code. Used in association with a JavaCompute node.

JavaCompute node. A node in a message flow for transforming and routing messages using Java. It is also used for performing database operations using Java.

Mapping. A method of message transformation using drag and drop from references to message definitions and database definitions.

Mapping node. A node in a message flow that uses message mappings to construct an output message using other messages or information from database tables.

Message Brokers Toolkit. A graphical user interface for performing the development and debugging of message flow applications. It is also used for administering WebSphere Message Broker components and deploying message flow applications.

Message Definition editor. An editor in the Message Brokers Toolkit for defining the logical and physical structure of messages.

Message Domain. The Message Domain is a property that can be set on an input node to indicate the type of message that the flow expects to process, and selects the appropriate parser for the flow to use. Examples are XML and MRM.

Message Flow Debugger. A tool for tracing the path of messages through a message flow and viewing the changing content of the message as it is processed by the flow.

Message Flow editor. An editor within the Message Brokers Toolkit for creating message flows by adding and connecting nodes on a canvas.

Message flows. Message flows provide the logic used by the broker to process messages. Message flows are built from nodes programmed with basic logic.

Message mapping editor. An editor in the Message Brokers Toolkit for defining mapping relationships between a source and target message or database.

Message Set editor. An editor in the Message Brokers Toolkit for setting the logical and physical properties of a message set.

Message sets. Message sets contain definitions of messages to be processed by the broker. These message definitions contain information about the logical and physical structure of the messages.

MQInput node. A node in the message flow for taking a message off a WebSphere MQ queue for processing.

MQOutput node. A node in the message flow for putting a message on an WebSphere MQ queue after processing.

mqsideploy. A command run in the command console for deploying message broker archive files.

mqsilist. A command run in the command console to list all the WebSphere Message Broker components on the machine.

mqsistart. A command run in the command console to start a component such as a broker or configuration manager.

mqsistop. A command run in the command console to stop a component such as a broker or configuration manager.

ODBC drivers for Cloudscape. Open Database Connectivity drivers for the embedded Derby database used by the Configuration Manager. These are used when the Derby database is used as a broker database.

Output node. A node that takes a message from a message flow and puts it to an application after processing.

Publish/subscribe. An alternative style of messaging using topics. Messages published on a topic are sent to all applications which subscribe to that topic.

Queue manager. A queue manager is a system program that provides queuing services to applications. It is used to enable communication between the WebSphere Message Broker components, each component requires access to a Queue Manager.

Rational Agent Controller. The Rational Agent Controller is used for message flow debugging in the Message Brokers Toolkit. It must be installed on the same machine as the broker being debugged.

Rules and Formatter Extension. An extension from New Era of Networks providing Rules and Formatter nodes and the associated runtime elements to maintain functionality supplied by earlier releases.

Runtime version information. Information added to message flows and message sets to provide information about the version. This information is visible in deployed resources through the Message Brokers Toolkit.

Sample Deploy Wizard. A wizard for use with the WebSphere Message Broker samples, which can import the sample files, create WebSphere MQ and database resources and deploy the sample to a default configuration.

Subscriptions. A subscription is a registration of an applications interest in a particular topic in publish/subscribe.

System Log. A log viewable in the Windows Event Viewer that displays information about software running as Windows services including WebSphere Message Broker components.

Terminal. Each node in a message flow has a number of terminals. Messages are output to different terminals on a node depending upon the results of processing in the node.

Topic. Used in publications and subscriptions to control the routing of publish/subscribe messages. A publication is about a particular topic.

User Name Server. Used to provide authentication and security for publish/subscribe in a broker domain.

WebSphere Event Broker. Used for the distribution and routing of messages from disparate applications. Is often used for publish/subscribe messaging.

WebSphere Message Broker. Provides storage, transformation and enrichment of data in addition to the functionality provided by WebSphere Event Broker.

WebSphere MQ. A messaging application which enables the Message Brokers Toolkit, Configuration Manager, and brokers to communicate. WebSphere MQ provides many of the available transport protocols between business applications and message flows.

WebSphere MQ Explorer. A graphical user interface for WebSphere MQ for administering WebSphere MQ components such as queue managers and channels.

Windows Event Viewer. A Windows tool for viewing the contents of the Application and System logs.

Abbreviations and acronyms

ASE	Adaptive Server Enterprise
bar	broker archive
BLOB	binary large object
CWF	Custom Wire Format
ESQL	Extended Structured Query Language
HCI	human-computer interaction
IBM	International Business Machines Corporation
ITSO	International Technical Support Organization
ODBC	Open Database Connectivity
RAC	Rational Agent Controller
RDB	relational database
SQL	Structured Query Language
TDS	Tagged Delimited String

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 330. Note that some of the documents referenced here may be available in softcopy only.

- ► *WebSphere Business Integration Message Broker Basics*, SG24-7090-00
- ► *Migrating to WebSphere Message Broker V6*, SG24-7198

Online resources

These Web sites and URLs are also relevant as further information sources:

- ► Product documentation in the information center:

  http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp

- ► This is the online version of the WebSphere Message Broker V6.0:

  http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp
  http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp?topic=/co
  m.ibm.etools.mft.doc/ah24100_.htm

- ► High availability document for message broker:

  http://www-128.ibm.com/developerworks/websphere/library/techarticles/0403_h
  umphreys/0403_humphreys.html

- ► Message broker requirements:

  http://www-306.ibm.com/software/integration/wbimessagebroker/requirements/

- ► WebSphere MQ documentation library:

  http://www-306.ibm.com/software/integration/wmq/library/

- ► DB2 information center:

  http://publib.boulder.ibm.com/infocenter/db2help/index.jsp

- ► Information about rational application developer:

  http://www-306.ibm.com/software/awdtools/developer/application/

- WebSphere Message Broker roadmap:

  http://www-128.ibm.com/developerworks/websphere/zones/businessintegration/roadmaps/wsmb/

- WebSphere Message Broker support downloads:

  http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg27006367

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

New Search List dialog   311
node names   55
node palette   48, 51
node palette, viewing   49
node properties   62
normal level service trace   286
NullPointer errors   293

O

ODBC (Open Database Connectivity)   19
ODBC Data Source Administrator   289–290
ODBC Drivers for Cloudscape   19, 26–27
ODBC drivers for Cloudscape   23
ODBC trace   289–290
omplex types   169
onfiguration Assistant   303
Open Defined   179
Open Map   153
Optional software   20
Oracle   21, 28, 31
output message   67, 121, 150, 153, 271
OutputRoot   67, 271
outstanding deploys   301

P

Package   121
Package Explorer view   114, 121
parsing   136
parsing defined messages   151
password length   24
PDE Runtime Error Log   294
performance   24, 278, 285, 287–288
physical format   137, 143
physical formats   141
Planning for installation   17–18
Platform-specific information   5
point-to-point   8
point-to-point messaging   11
point-to-point routing   10
Pop-up messages   244
pop-up messages   244
pop-ups   244
Preferences   287, 294
prerequisite software   19, 24
previous versions of WebSphere Message Broker
4
Problems View   244, 247
Problems view   58, 67, 247

problems with deployment   299
product documentation   307
product migration   4
project with errors   246
Properties dialog   65
Publish/subscribe   11
publish/subscribe   9, 14, 206
publish/subscribe messaging   8
Publishsubscribe   205
putting and getting messages   15

Q

Queue Name field   63
queue name missing   247
Quick Tour   26

R

Rational Agent Controller   27
Rational Agent Controller (RAC)   20
Rational Agent Controller client   20
Rational Application Developer   22
Rational Application Developer platform   20
Rational Application Development platform   275,
294
Rational® Application Developer platform   19
RDB Definitions Files   182
Readme   26
Reason section   245
Reconnect database   185
Redbooks Web site   330
    Contact us   xxi
referenced project   296
relational database (RDB)   182
relational databases   50
Remote broker not responding   301
Remove Deployed Children   300
repeating elements   180–181
required software   17–18
Reset trace settings   285
reset trace settings   285
Resource   247
Resource Navigator view   54, 142, 144, 184
results of administration operations   244
retry attempts   298
return code   290
Revert   251
roll back in message flows   61
RouteToLabel node   51

X

XML 11
XML declaration statement 160
XML format 48
XML Parser 63
XML parser 48
XML Wire Format 142, 162
XPath 138, 140

Z

zero Min Occurs 170, 175

WebSphere Message Broker Basics

WebSphere Message Broker Basics

IBM®

Redbooks

Introduces WebSphere Message Broker V6

Describes basic installation, configuration, and development tasks

Explores the Message Brokers Toolkit

This IBM Redbook provides an overview of the latest release of WebSphere Message Broker and the Message Brokers Toolkit. It covers the following topics:

► Installation, plus creating and verifying a default configuration

► Developing simple message flows using ESQL, Java, and Message Maps

► Developing simple message sets

► Broker administration tasks and deploying message flow applications

► Diagnosing and fixing problems

► Using the message Flow Debugger

This book also describes where to find more information, including product documentation and sample applications.

This book updates the popular redbook SG24-7090 from WebSphere Business Integration Message Broker v5.0 to WebSphere Message Broker v6.0.