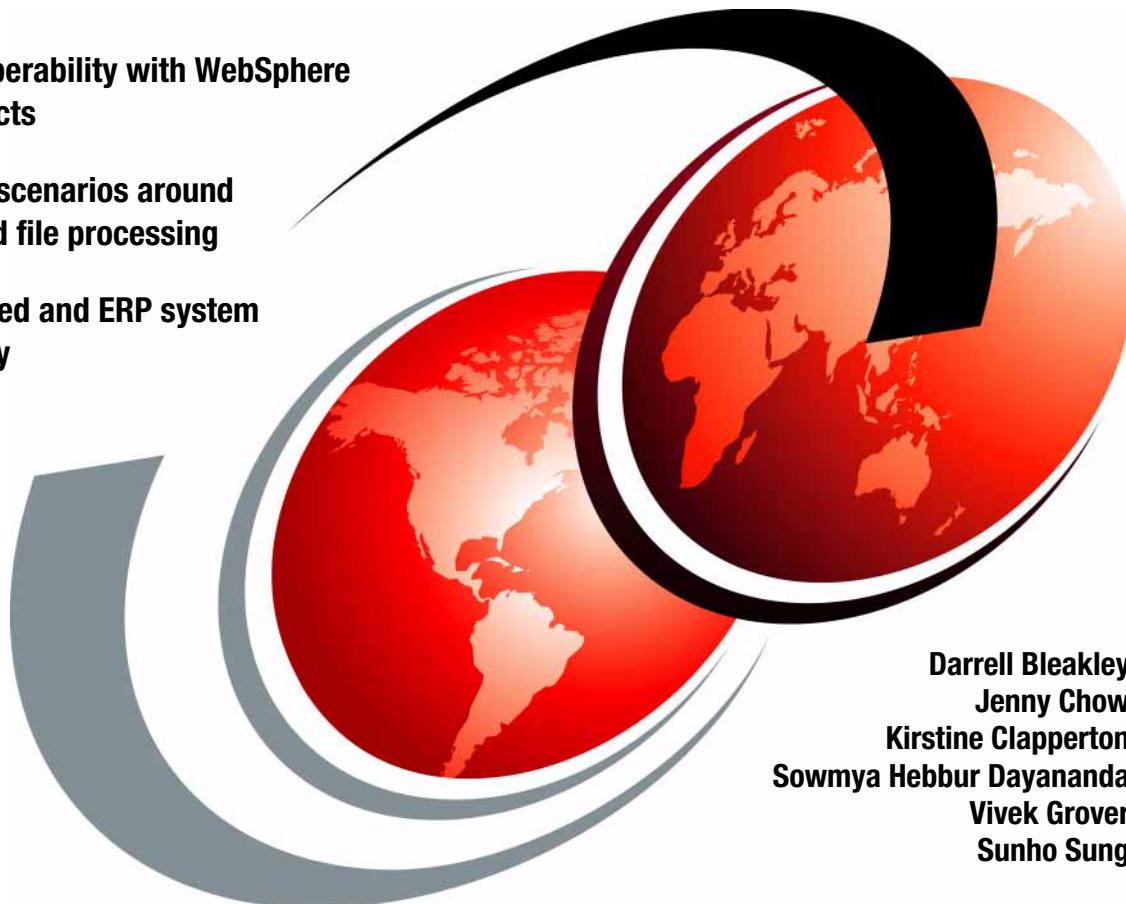


Connecting Your Business Using WebSphere Message Broker V7 as an ESB

SOA interoperability with WebSphere
BPM products

Functional scenarios around
PubSub and file processing

TCP/IP-based and ERP system
connectivity



Darrell Bleakley
Jenny Chow
Kirstine Clapperton
Sowmya Hebbur Dayananda
Vivek Grover
Sunho Sung

Redbooks



International Technical Support Organization

**Connecting Your Business Using WebSphere
Message Broker V7 as an ESB**

May 2010

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

First Edition (May 2010)

This edition applies to WebSphere Application Server V7, WebSphere MQ V7.0.1, WebSphere Business Monitor V7, WebSphere ILOG JRules V7, WebSphere Message Broker V7, WebSphere Integration Developer V6.2.0.2 and V7, WebSphere Process Server V6.2.0.2, V7 and WebSphere Service Registry and Repository V7.

© Copyright International Business Machines Corporation 2010. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Noticesix
Trademarks	x
Prefacexi
The team who wrote this bookxi
Now you can become a published author, too!	xiv
Comments welcome.....	xiv
Stay connected to IBM Redbooks	xv
Part 1. WebSphere Message Broker V7 as an ESB	1
Chapter 1. The IBM SOA Foundation and products in this book	3
1.1 IBM SOA Foundation	4
1.2 The enterprise service bus	6
1.2.1 The role of an enterprise service bus	6
1.2.2 WebSphere DataPower Integration Appliance XI50	7
1.2.3 WebSphere Message Broker	8
1.3 The focus of this book	11
Chapter 2. Introduction to WebSphere Message Broker V7.0.	13
2.1 Usage of WebSphere Message Broker.....	14
2.1.1 Message flows with WebSphere Message Broker	14
2.2 Runtime architecture of WebSphere Message Broker	16
2.2.1 Broker	17
2.2.2 Execution groups	17
2.3 Development environment of WebSphere Message Broker	18
2.3.1 WebSphere Message Broker Toolkit	18
2.3.2 Message flows	19
2.3.3 Message	22
2.3.4 Message modeling	23
2.3.5 Pattern instances	23
2.4 Administering WebSphere Message Broker	25
2.4.1 WebSphere Message Broker Explorer	25
2.4.2 Broker sets	28
2.5 Deploying message flow applications	28
Chapter 3. Mediation with WebSphere Message Broker	31
3.1 Routing with message flows	32
3.1.1 Routing using JavaCompute Out and Alternate terminals	32

3.1.2 Routing using the RouteToLabel node	34
3.1.3 Routing using an application database	34
3.1.4 Routing using a Compute node.....	36
3.2 Aggregation with message flows.....	37
3.3 Protocol transformation with message flows.....	39
3.4 Message transformation with message flows.....	41
3.4.1 Message models.....	44
3.4.2 Message transformation nodes.....	46
3.5 Message enrichment with message flows.....	51
3.5.1 Database content to enrich messages	51
3.5.2 Messages on queues to enrich messages	52
3.5.3 Broker properties and environment values	52
3.5.4 Files.....	52
3.5.5 Web Services content to enrich messages.....	53
Chapter 4. Connectivity options for interoperability with WebSphere Message Broker	55
4.1 Messaging transport	56
4.2 Connectivity options	57
4.2.1 WebSphere MQ	57
4.2.2 Java Message Service (JMS) 1.1	58
4.2.3 WebSphere MQ/JMS	59
4.2.4 SOAP-based Web services.....	61
4.2.5 HTTP and HTTPS.....	63
4.2.6 File	64
4.2.7 Java Connector Architecture (JCA) Adapters	65
4.2.8 TCP/IP.....	67
4.2.9 Service Component Architecture	68
Part 2. Scenarios	71
Chapter 5. Using SCA nodes for simplified integration with WebSphere Process Server	73
5.1 Introduction to WebSphere Message Broker V7.....	74
5.1.1 Overview of WebSphere Process Server	74
5.1.2 Services Component Architecture overview	77
5.1.3 WebSphere Integration Developer overview.....	80
5.2 SCA Nodes and integration advantages in WebSphere Message Broker V7	82
5.2.1 SCA Nodes overview	82
5.2.2 The integration advantages of using SCA nodes of WebSphere Message Broker V7	84
5.2.3 Summary.....	86
5.3 Environment	87

5.4 Scenarios: Developing scenarios using SCA Nodes to integrate WebSphere Message Broker and WebSphere Process Server.....	87
5.4.1 Scenario overview.....	87
5.4.2 Setting up the environment.....	98
5.4.3 SCA Nodes scenario part 1a: Asynchronous Request and Response	99
5.4.4 SCA Node scenario part 1b: SCA Request.....	115
5.4.5 SCA Nodes scenario part 2: A Message Broker flow implements a service using SCA Input and SCA Reply	127

Chapter 6. Enterprise Resource Planning integration with WebSphere Message Broker	143
6.1 Terms associated with WebSphere Adapters.....	144
6.2 EIS connectivity with JCA adapters.....	144
6.3 WebSphere Message Broker V7.0 enhancements to the WebSphere Adapters	145
6.4 WebSphere Adapter overview.....	147
6.5 Using WebSphere Adapters in a message flow	147
6.5.1 Step 1: Preparing the environment for WebSphere Adapters nodes	148
6.5.2 Step 2: Discovering the EIS objects and services.....	148
6.5.3 Step 3: Adding the WebSphere Adapters node to the message flow ..	149
6.5.4 Step 4: Creating and deploying the broker archive file to the broker	149
6.6 The supported SAP adapter interfaces.....	150
6.6.1 SAP inbound operations	150
6.6.2 SAP outbound operations	151
6.7 Application Integration Patterns	152
6.8 WebSphere Message Broker scenario environment.....	153
6.9 Scenarios	154
6.9.1 Scenario 1: SAP Inbound	155
6.9.2 Scenario 2: SAP Outbound.....	172
6.9.3 Scenario 3: Simulated master data synchronization scenario between SAP and Siebel enterprise information systems.....	197

Chapter 7. Service enable TCP/IP-based applications with WebSphere Message Broker	243
7.1 Introduction to TCP/IP-based applications	244
7.1.1 TCP/IP nodes	244
7.1.2 Combining TCP/IP nodes	250
7.2 Scenario environment	253
7.3 Scenarios	253
7.3.1 Scenario 1: Healthcare System integration using TCP/IP.....	253
7.3.2 Scenario 2: Telnet to WebSphere MQ	264

7.3.3 Scenario 3: Configurable Services for TCP/IP nodes	272
Chapter 8. Scenario: File processing	279
8.1 Introduction to file processing	280
8.1.1 File handling using the FileInputStream node	281
8.1.2 File handling using the FileOutputStream node	287
8.1.3 FtpServer configurable service for remote file processing	291
8.1.4 Message transformation with PHPCompute node	292
8.2 Scenario environment	307
8.3 Scenario	307
8.3.1 File processing patterns	307
8.3.2 Scenario overview	308
8.3.3 Record structure	309
8.4 Message flows	311
8.4.1 Preparing the environment	311
8.4.2 Creating a message definition	314
8.4.3 Generating a pattern instance	318
8.4.4 Generated artifacts	324
8.4.5 Implementing message flow	334
8.5 Testing the scenario	340
Chapter 9. Scenario: Publish/subscribe using WebSphere Message Broker V7.0	343
9.1 Introduction to publish/subscribe	344
9.2 Scenario environment	348
9.3 Scenario 1: Comparison and difference	349
9.3.1 Demonstrating publish/subscribe in WebSphere MQ V7.0.1	354
9.3.2 Demonstrating publish/subscribe in WebSphere Message Broker V6	367
9.3.3 Demonstrating publish/subscribe in WebSphere MQ V7.0.1 using WebSphere Message Broker V7	372
9.4 Scenario 2: Migration	379
9.4.1 Migrating a publish/subscribe from WebSphere Message Broker V6.1 to WebSphere Message Broker V7.0	381
9.5 Scenario 3: Content-based filtering	402
9.5.1 Demonstrating publish/subscribe with content-based filtering using Message Broker	405
9.5.2 Publish/subscribe using Selector in WebSphere MQ only	417
9.6 Conclusion	425
Chapter 10. Scenario: Monitoring and WebSphere Business Monitor support	427
10.1 Introduction to the products	428
10.1.1 Business Activity Monitoring	428

10.1.2 System architecture of the scenario	429
10.2 Scenario environment	430
10.3 Scenario	431
10.4 Connectivity	432
10.4.1 WebSphere MQ/JMS	432
10.4.2 Message-Driven Bean	434
10.5 WebSphere MQ	440
10.6 WebSphere Process Server	441
10.6.1 Business integration module	441
10.6.2 Business objects	441
10.6.3 Interfaces	444
10.6.4 Business process	445
10.6.5 Export component and binding	450
10.6.6 ShippingServiceModule Project Interchange	451
10.7 WebSphere Message Broker	452
10.7.1 OrderService message flow	452
10.7.2 Monitoring events	459
10.7.3 Command-line configuration	466
10.7.4 Enabling monitoring	468
10.8 WebSphere Business Monitor	468
10.8.1 Monitor model	469
10.8.2 Business Space	483
10.9 Testing the scenario	486
10.10 Extension: Monitoring WebSphere Message Broker and WebSphere Process Server	491
10.10.1 WebSphere Process Server events	492
10.10.2 WebSphere Business Monitor Model	495
10.11 Additional details	504
10.11.1 Event sequencing	504
10.11.2 Audit capabilities	505
10.11.3 Managing the scenario	505
Chapter 11. WebSphere Service Registry and Repository as governance .	
507	
11.1 Introduction to the WebSphere Service Registry and Repository	508
11.2 WSRR nodes in WebSphere Message Broker	509
11.2.1 Query depth support by RegistryLookup node	511
11.2.2 WSRR configurable services	511
11.3 Scenario	519
Chapter 12. Using WebSphere ILOG JRules together with WebSphere Message Broker	521
12.1 Introduction to this chapter	522

12.1.1	Introducing the concept of Business Rule Management System .	522
12.1.2	WebSphere ILOG JRules overview	523
12.2	Benefits of using WebSphere Message Broker and WebSphere ILOG JRules together	525
12.2.1	WebSphere ILOG JRules enables WebSphere Message Broker in three key business areas	526
12.2.2	WebSphere Message Broker enables and extends ILOG BRMS connectivity	529
12.3	Integration approaches and scenarios	531
12.3.1	Using JRules through Web Services.	531
12.3.2	Message Broker calling JRules through JMS	531
12.3.3	Message Broker calling JRules J2SE Rule Execution Server . .	532
Appendix A.	Additional material	533
Locating the Web material		533
Using the Web material		534
How to use the Web material		534
Related publications		535
IBM Redbooks		535
Online resources		535
How to get Redbooks		536
Help from IBM		536

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 illustrate 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.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

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

CICS®
DataPower®
DB2®
IBM®

IMS™
Redbooks®
Redpaper™
Redbooks (logo) 

System z®
Tivoli®
WebSphere®
z/OS®

The following terms are trademarks of other companies:

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.

SAP is a registered trademark of SAP AG in Germany and in several other countries.

Siebel is a trademark of Siebel Systems, Inc. and may be registered in certain jurisdictions.

PeopleSoft is a registered trademark of PeopleSoft, Inc.

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

Preface

This IBM® Redbooks® publication points out the key features that make WebSphere® Message Broker a powerful choice as an enterprise service bus (ESB) solution in a service-oriented architecture (SOA) environment. In this book, we illustrate the interoperability between the WebSphere Message Broker and the applications in the SOA environment.

We use realistic examples to show the ESB capabilities of WebSphere Message Broker. We also show how to integrate WebSphere Message Broker with a variety of enterprise applications, which include WebSphere Process Server and ESB systems including SAP and Siebel, WebSphere Business Monitor, and WebSphere Service Registry and Repository.

We wrote this book for architects who are planning an SOA solution and application designers who are implementing an SOA solution with WebSphere Process Server and WebSphere Message Broker.

The team who wrote this book

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

Darrell Bleakley is a Senior IT Specialist in the GBS Application Services organization in Raleigh, NC. He has 12 years of experience in complex software and system integration initiatives at IBM. His areas of expertise include middleware and enterprise service bus integration solutions using a variety of products, including WebSphere Message Broker, WebSphere Service Registry and Repository, MQ, and Web-Service standards. He has a Bachelor of Science degree in Computer Engineering and a Masters degree in Engineering Management from the University of Florida.

Jenny Chow is an IBM Senior Certified Executive IT Specialist in the Software Sales and Distribution organization. She is responsible for IBM WebSphere Brand Technical Professionals' role in the New York Area Region. Over the past 17 years she provided extensive contributions to help IBM New York (NY) and New Jersey (NJ) Metro customers to design messaging, connectivity, SOA, and business integration solutions. With her broad IT background and expertise in strategic middleware solution, she was selected to be a member of the IBM Americas Software Technical Leadership Council. Also, she led the WebSphere

Solution Architect Competency Center for East Region in 2009 and the NY Area Region from 2008 to the present. Furthermore, Jenny is a member of several IBM IT Professional Certification Boards. As the founder of the NY/NJ WebSphere MQ User Group, created in 1998, Jenny promoted this user group to become the world's largest and most active user group, which won many Golden Awards for the past five years. Jenny has a Masters of Science degree in Information Systems from Polytechnic Institute of New York University, New York and a Bachelor of Science degree in Computer Sciences.

Kirstine Clapperton is a Software Engineer at the IBM Hursley Laboratory in the United Kingdom (UK). She has been at IBM for nine years and the last six of which were as a member of the WebSphere Message Broker Level 3 service team. She has a first class honours degree (BSc) in Computing and Information systems from the University of London and an MSc in Software Engineering from the University of Oxford. Her areas of expertise include Integration Middleware Software, primarily WebSphere MQ and WebSphere Message Broker products.

Sowmya Hebbur Dayananda is a Software Engineer at the IBM India Software Labs in Bangalore. She has been at IBM for nearly three years with the WebSphere Message Broker development team. She has a Bachelor of Engineering (BE) degree in Computer Science discipline from Vishweshwaraiah Technological University. Her proficiency is IBM Middleware software products, mainly WebSphere Message Broker and WebSphere MQ.

Vivek Grover is an Advisory Software Engineer working in Level 2 support for the IBM AIM organization in Research Triangle Park, North Carolina. He has nine years of experience working with multiple products at IBM. He is an IBM certified System Administrator for WebSphere Message Broker and WebSphere MQ V6 products. He currently works with IBM WebSphere Message Broker and WebSphere Business Events. Vivek works with customers worldwide and focuses on product integration and troubleshooting issues. He is also the e-Support team lead who assists Knowledge Engineers in designing, writing, and publishing technotes, Document Control Facility (DCF), and quarterly e-mails to the customers about technical issues. He has a Masters degree in Management and Information Systems from Northern Illinois University, Illinois. He has written technical articles and education modules for IBM Education Assistant and Developer works.

Sunho Sung is a Senior IT Specialist in the IBM Software Group in Korea. He joined IBM in 2000 and participated in a number of projects as a Web Application Developer and Architect. Since transferring to the IBM Software Group in 2005, he supported the technical sales of WebSphere software on z/OS®, including Integration Middleware Software, WebSphere on System z®, CICS® tools, and PD tools. He has a BSc in Chemistry from Kyungpook National University in

Korea and an MSc (with distinction) in Computing from the University of Northumbria at Newcastle in the UK.



Figure 1 Residency team (from left to right): Sunho Sung, Rufus Credle, Vivek Grover, Jenny Chow, Darrell Bleakley, Kirstine Clapperton, Margaret Ticknor, Sowmya Hebbur Dayananda

Thanks to the following people for their contributions to this project:

Margaret Ticknor
Project Leader, International Technical Support Organization, Raleigh Center

Rufus P. Credle Jr, Tamikia Barrow
International Technical Support Organization, Raleigh Center

Chris Backhouse, Kevin Braithwaite, Andrew Coleman, Dave Dalton, David Hardcastle, Andrew Humphreys, Paul Lacy, Matt Lucas, Sanjay Nagchowdhury, John Reeve, Jonathan Woodford, Lakshman Yatawara
IBM UK

Daniel Tabuenca Velasco
IBM Spain

Guy Hochstetler
IBM US

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author - all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

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 books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:
ibm.com/redbooks
- ▶ Send your comments in an e-mail to:
redbooks@us.ibm.com
- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:
<http://www.facebook.com/pages/IBM-Redbooks/178023492563?ref=ts>
- ▶ Follow us on twitter:
<http://twitter.com/ibmredbooks>
- ▶ Look for us on LinkedIn:
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<http://www.redbooks.ibm.com/rss.html>



Part 1

WebSphere Message Broker V7 as an ESB

In this part, we highlight the key features that make WebSphere Message Broker a powerful choice as an enterprise service bus solution in a service-oriented architecture environment. This part includes the following chapters:

- ▶ Chapter 1, “The IBM SOA Foundation and products in this book” on page 3
- ▶ Chapter 2, “Introduction to WebSphere Message Broker V7.0” on page 13
- ▶ Chapter 3, “Mediation with WebSphere Message Broker” on page 31
- ▶ Chapter 4, “Connectivity options for interoperability with WebSphere Message Broker” on page 55



The IBM SOA Foundation and products in this book

Our purpose in this book is to explain and illustrate the interoperability of WebSphere Message Broker and SOA-based applications. In this chapter, we set the stage by providing a brief background on the IBM SOA Foundation and the products that we discuss in this book. Specifically, we address the following topics:

- ▶ “IBM SOA Foundation” on page 4
- ▶ “The enterprise service bus” on page 6
- ▶ “The focus of this book” on page 11

1.1 IBM SOA Foundation

The IBM SOA Foundation is an integrated, open set of software, best practices, and patterns that provides what you need to get you started with the service-oriented architecture. The SOA Foundation provides full support for the SOA life cycle through an integrated set of tools and runtime components that allow you to use skills and investments across the common runtime, tooling, and management infrastructure.

Because the components are modular, you can pick and choose the pieces that you need to deliver an immediate impact, and what you pick now will work with the pieces you add later. In addition, because the SOA Foundation is scalable, you can start your business small and grow it as fast as it requires. The SOA Foundation provides extensive support for business and IT standards, facilitating greater interoperability and portability between applications. The foundation can also help you to use SOA to extend the value of the applications and business processes that are running your business today.

The IBM SOA reference architecture, shown in Figure 1-1 on page 5 is a way to look at the set of services that go into building an SOA. These capabilities can be implemented on a build-as-you-go basis, which allows you to easily add capabilities and project-level solutions as new requirements are addressed over time. The reference architecture shows the tight integration with other critical IT aspects, such as security, IT monitoring, virtualization, and workload management.

The backbone of the reference architecture is the enterprise service bus that facilitates communication between services. The ESB provides the inter-connectivity capabilities that are required to use services that are implemented across the entire architecture. Transport services, event services, and mediation services are all provided through the ESB. You can implement an ESB with one product or multiple products that are combined to provide the required function. The ESB solution that you select must be optimized to meet the unique business requirements of your enterprise. The solutions must also be capable of changing as your business requirements evolve.

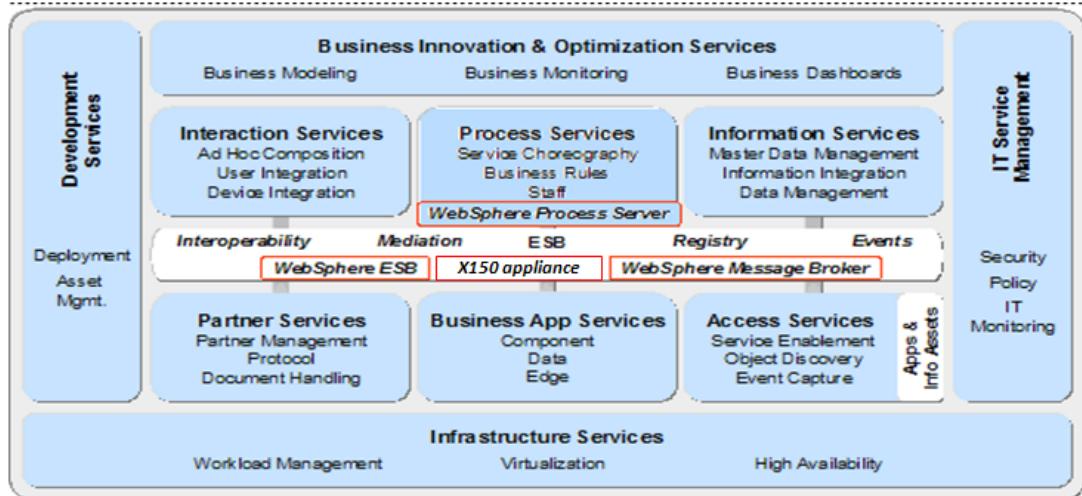


Figure 1-1 IBM SOA reference architecture

Figure 1-2 shows the application of WebSphere Message Broker, DataPower® appliance X150, and WebSphere ESB providing the mediation logic for inter-connectivity among the different applications in a SOA environment.

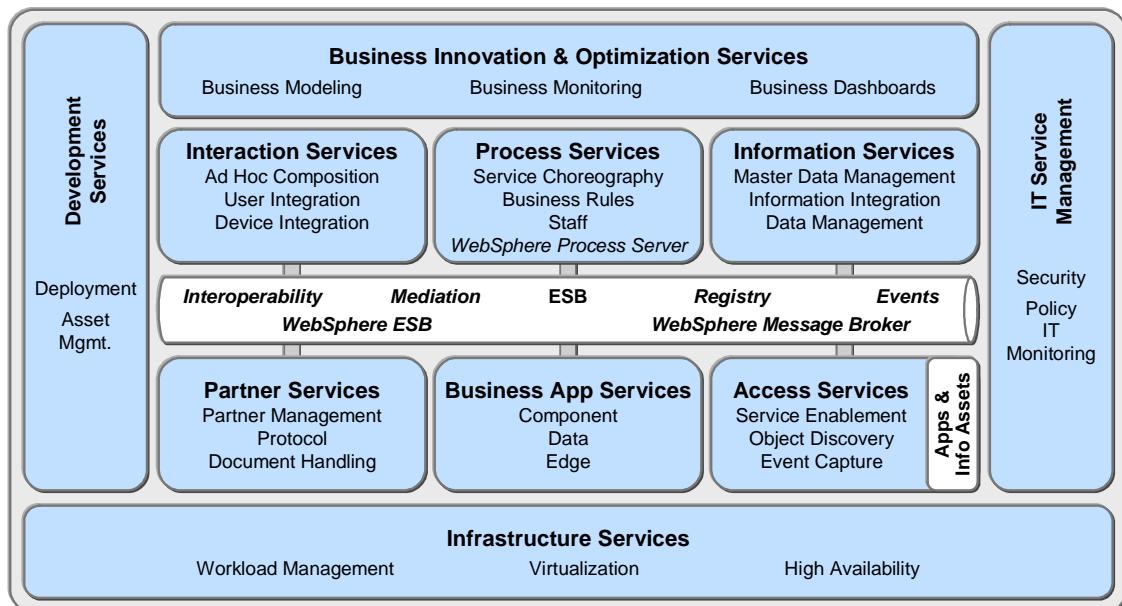


Figure 1-2 IBM SOA reference architecture with WebSphere Message Broker

1.2 The enterprise service bus

An ESB acts as a mediator between service consumers and service providers. It decouples the service consumer from the service provider, providing services to resolve differences in protocol and format. It shrinks the number of interfaces and improves the reusability of interface components to cut cycle time from design to deployment.

An ESB is an architectural concept that is implemented through using two to three products. IBM currently offers two ESB products:

- ▶ WebSphere Enterprise Service Bus (WESB) is built on proven messaging and Web services technologies. It provides standards-based Web services connectivity and XML data transformation. WebSphere ESB is shipped with WebSphere Process Server.
- ▶ WebSphere Message Broker provides universal connectivity, including Web services and any-to-any data transformation. In addition, you can use such products as DataPower and WebSphere Transformation Extender to extend the capabilities of the core ESB products.

ESB products: An ESB is a key part of a complete solution for connectivity. The functionality of an ESB can be served through one product or a combination of products. Selecting the correct ESB product set with the correct set of capabilities is critical for the success of a project. You might need to extend or complement an ESB with additional features or products to deliver a complete solution to meet your connectivity requirements.

1.2.1 The role of an enterprise service bus

In a non-SOA solution, communication between service requesters and providers is accomplished using a series of direct connections, one connection per requester/provider pair, which requires that the requester be aware of the requirements of the provider and vice versa, for example, the transport protocol, message format, and location of the service. As enterprise solutions grow, the web of direct connections and the complexity of managing those connections also grows.

In an SOA solution, an ESB acts as an intermediary between requesters and providers. Connections are made to the ESB rather than directly between the communicating pair. The ESB makes the transition between transport protocols and message formats that the requester and provider use. The location of each service is known to the ESB but not to the service. The advantages that this

decoupling of the consumer's view of a service from the implementation of a service provides are:

- ▶ Reduces the number, size, and complexity of interfaces.
- ▶ Reduces the impact of changes that are made to the format and location of services, both in terms of impact to the applications and in terms of system management.
- ▶ Enables integration between disparate resources.
- ▶ Allows substitution of one service provider for another without the service consumer being aware of the change or without needing to alter the architecture to support the substitution.

The capabilities of an ESB vary depending on the products that are used to implement it. An ESB must provide the following basic functions:

- ▶ Routing messages between services

The requester sends the request to the ESB. The ESB makes the call to the service provider. The ESB determines the destination for the service provider.
- ▶ Converting transport protocols between the requester and service

Without an ESB infrastructure, service consumers connect directly to service providers using a transport protocol that is supported by the provider. With an ESB, there is no direct connection between the consumer and provider. The service consumer connects to the ESB by using its preferred transport protocol without any knowledge of how the connection to the provider is made. The ESB sends the request to the provider using a protocol that is supported by the provider.
- ▶ Transforming message formats between requester and service

Typically, interfaces and operations of disparate services are not identical. The ESB transforms the message from the source into a format that can be accepted by the target.
- ▶ Handling business events from disparate sources

The same service provider that is responsible for a business function can be invoked from a variety of business contexts.

1.2.2 WebSphere DataPower Integration Appliance XI50

IBM WebSphere DataPower Integration Appliance XI50 is the IBM hardware ESB that delivers common message transformation, integration, and routing functions in a network device, cutting operational costs and improving performance. By making on-demand data integration part of the shared

service-oriented architecture (SOA) infrastructure, the XI50 is one of the non-disruptive technologies for application integration.

The XI50 appliance offers following features:

- ▶ Simplifies infrastructure with an Application Optimization option to provide intelligent back-end application workload balancing (available only for 9235 machine type).
- ▶ Bridges to Web 2.0 technologies with JSON filtering and validation, support for REST verbs, and converting/bridging of REST and Web services.
- ▶ Provides fast and flexible application integration with declarative any-to-any transformations between disparate message formats.
- ▶ Reduces integration costs with wirespeed mediation, protocol bridging, transport mediation, and content-based message routing.
- ▶ Lowers operational costs with native connectivity to existing access control, monitoring, database and management systems, and processes.
- ▶ The XI50 can off-load XSLT processing, XPath routing, legacy-XML conversion, and other resource-intensive tasks from servers to reduce latency, improve throughput, and free up compute resources.
- ▶ The XI50 can help protect and XML-enable mainframe systems, instantly connecting them to enterprise SOA and Web services.

1.2.3 WebSphere Message Broker

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

WebSphere Message Broker offers the following features:

- ▶ Universal connectivity:
 - Simplifies application connectivity to provide a flexible and dynamic infrastructure
- ▶ Routes and transforms messages *from anywhere, to anywhere*:
 - Supports a wide range of protocols, for example, WebSphere MQ, Java™ Message Service (JMS) 1.1, HTTP(S), Web services (SOAP and REST), File, TCP/IP, Enterprise Information Systems (including SAP and Siebel), Service Component Architecture (SCA), and user-defined protocols

- Supports a broad range of data formats, for example, binary (C/COBOL), XML, industry (SWIFT, EDI, HIPAA, and so on), IDoc, CSV, and user-defined formats
- Supports interactions and operations that allow you to route, filter, transform, enrich, monitor, distribute, decompose, correlate, sequence, detect, and more
- ▶ Simple programming:
 - Message flows process and route messages. A message flow contains a series of connected nodes that have the required integration logic that is used to operate on messages as they flow through the broker
 - Message trees describe the data in an independent format manner
 - Transformation options include graphical mapping, Java, extended SQL (ESQL), Post Hypertext Preprocessor (PHP), Extensible Stylesheet Language (XSL), and WebSphere Transformation Extender (WTE)
 - You can reuse or modify the supplied patterns based for top-down parameterized connectivity of common use cases to suit the business requirements, for example, Web service façades, message-oriented processing, queue to file, and so on
- ▶ Operational management and performance:
 - Extensive administration and systems management facilities are available for developed solutions
 - A wide range of operating system and hardware platforms are supported with WebSphere Message Broker
 - Offers performance of traditional transaction processing environments
 - Available in Trial, Remote deployment, Get Started, and Enterprise deployment options
 - Tightly integrated with other IBM and non-IBM software products that provide related management and processing environments
- ▶ Support for adapters and files:
 - JCA-based WebSphere Adapters are delivered as built-in message flow nodes to provide connectivity to three major EIS systems, SAP, Siebel, and PeopleSoft systems, which enables the exchange of business data through an integration broker
 - File processing support, including FTP and SFTP, is delivered through the built-in File nodes

WebSphere Message Broker contains a choice of transports that enable secure business to be conducted at any time by providing powerful integration, message, and data transformations in a single place. WebSphere Message

Broker is built on WebSphere MQ; therefore, it supports MQ transport. However, it also supports many other transports, such as HTTP/HTTPS, SOAP, File, TCP/IP, EIS, and so on, that do not use MQ stack.

Drivers for using WebSphere Message Broker as an ESB

Consider the following reasons for using a WebSphere Message Broker as an ESB with other applications:

- ▶ You are currently using WebSphere Message Broker and want to use existing skills.
- ▶ You have extensive heterogeneous infrastructures, including both standard and non-standards-based applications, protocols, and data formats, for example, you are using industry formats, such as SWIFT, EDI, HL7, or HIPAA.
- ▶ You are implementing a wide range of messaging and integration patterns, for example, correlation, message splitting, and aggregation.
- ▶ You need extensive pre-built mediation support.
- ▶ Reliability and extensive transactional support are key requirements.
- ▶ You want to achieve high-performance with horizontal and vertical scaling.
- ▶ You want integration with other IBM WebSphere and Tivoli® products and third-party JMS providers.
- ▶ You want to natively transform non-XML formats without using adapters or user-written data transformations.
- ▶ You must find Web services, enrich the messages, and govern your SOA using WebSphere Service Registry and Repository (WSRR).
- ▶ You must monitor and enrich the business transactions in WebSphere Process Server.
- ▶ You want to extract and transform IDocs on ERP systems (SAP, Siebel, or PeopleSoft).
- ▶ You want advanced publish and subscribe capabilities, including the use of a wide range of transports; high speed, low-latency messaging; and both topic and content-based routing.

For more information, see the technical overview of WebSphere Message Broker Version 7.0 in the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.e.tools.mft.doc/ab20551_.htm

1.3 The focus of this book

Our intent for this book is to point out the key features that make WebSphere Message Broker a powerful choice as an ESB solution in an SOA environment and to illustrate the interoperability between the WebSphere Message Broker and these applications in the SOA environment. We use realistic examples and scenarios to illustrate the ESB capabilities of WebSphere Message Broker and to show how to integrate it with a variety of Enterprise applications, including WebSphere Process Server, EIS systems (including SAP and Siebel), WebSphere Business Monitor, and WebSphere Service Registry and Repository.

Note: For information about ESB design and using WebSphere ESB and WebSphere Message Broker and DataPower appliances as ESB solutions, refer to *Patterns: SOA Design Using WebSphere Message Broker and WebSphere ESB*, SG24-7369.

For information about Connectivity patterns in the application connectivity space, refer to the paper on *Enterprise Connectivity Patterns: Implementing integration solutions with the IBM enterprise service bus products*

<http://www.ibm.com/developerworks/webservices/library/ws-enterpriseconnectivitypatterns/index.html>



Introduction to WebSphere Message Broker V7.0

In this chapter, we introduce the features of WebSphere Message Broker. We also introduce the runtime architecture and the message flows that perform the enterprise service bus functionality.

Specifically, we cover the following topics in this chapter:

- ▶ “Usage of WebSphere Message Broker” on page 14
- ▶ “Runtime architecture of WebSphere Message Broker” on page 16
- ▶ “Development environment of WebSphere Message Broker” on page 18
- ▶ “Deploying message flow applications” on page 28

2.1 Usage of WebSphere Message Broker

WebSphere Message Broker is a platform-independent based ESB that provides universal connectivity. It can be used to integrate disparate applications and is designed to transform various formats of data between any type of applications using a number of supported communications protocols or distribution methods. It is used where there is a need for high-performance and complex integration patterns.

WebSphere Message Broker V7.0 offers simplicity and productivity in terms of developing and managing the WebSphere Message Broker environment. WebSphere Message Broker plays a critical role in SOA and offers a wide range of SOA scenarios in which it can be integrated. The dynamic operational management of WebSphere Message Broker enables administrators to effectively understand and modify broker behavior, which thus enables them to respond quickly to business requirements. WebSphere Message Broker is supported on a large range of platforms and environments.

Installing and configuring WebSphere Message Broker V7.0 was significantly simplified with the removal of the Configuration Manager, the User Name Server components, and the removing the requirement for a system database. For additional information about these changes, refer to the topic “What’s new in Version 7.0” in the WebSphere Message Broker V7.0 Information Center:

<http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp>

In this section, we discuss using WebSphere Message Broker as the ESB when it is integrated with other SOA components.

2.1.1 Message flows with WebSphere Message Broker

Processing logic in WebSphere Message Broker is implemented using *message flows*. Through message flows, messages from business applications can be transformed and routed to other business applications. Message flows are created by connecting *nodes* together. A wide selection of built-in nodes are provided with WebSphere Message Broker. These nodes perform tasks that are associated with message routing, transformation, and enrichment. The base capabilities of WebSphere Message Broker are enhanced by SupportPacs that provide a wide range of additional enhancements.

In the next sections, we summarize the mediation patterns that WebSphere Message Broker supports best and explain why a SOA application-WebSphere Message Broker combination might be used.

Message routing

Packaged with WebSphere Message Broker is a variety of nodes through which connectivity is provided for both standards and non-standards-based applications and services. Routing can be point-to-point or based on matching the content of the message with a pattern that is specified in a node.

Aggregation is an advanced form of message routing. With aggregation, a request message is received, and *multiple* new request messages are generated. Each new message is routed to its destination using a request-reply interaction. WebSphere Message Broker tracks the process, collecting each response and recomposing them into a single output message.

For information about the options for routing with WebSphere Message Broker, see , “In this chapter, we introduce the mediation capabilities of WebSphere Message Broker and describe how messages are routed, transformed, and enriched using a large variety of formats and protocols that WebSphere Message Broker supports.” on page 31, and 3.2, “Aggregation with message flows” on page 37.

Transport protocol conversion

WebSphere Message Broker provides universal connectivity between applications that use disparate transport protocols. WebSphere Message Broker enables connectivity between applications or business processes that use transport protocols, such as Web Services (SOAP, REST), HTTP(S), Java Message Service (JMS), WebSphere MQ, CICS, IMS™, TCP/IP, FTP, SCA, EIS (SAP, Siebel, PeopleSoft), and user-defined transports.

WebSphere Message Broker supports integration with WebSphere Business Adapters. For more information about available adapters, see the WebSphere Adapters page at:

<http://www-01.ibm.com/software/integration/wbiadapters/>

For information about the options for transport protocol conversion with WebSphere Message Broker, see 3.3, “Protocol transformation with message flows” on page 39.

Message transformation and enrichment

One of the key capabilities of WebSphere Message Broker is the transformation and enrichment of in-flight messages. This capability enables business integration without the need for any additional logic in the applications, for example, an application that generates messages in a custom format can be integrated with an application that only recognizes XML. This capability provides a powerful mechanism to unify organizations because business information can now be distributed to applications that handle completely separate message formats.

In WebSphere Message Broker, message transformation and enrichment are dependent 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 WebSphere Message Broker Toolkit contains facilities for defining messages to the WebSphere Message Broker.

Using parsers and message sets, WebSphere Message Broker can validate and check that incoming messages comply with the format that is defined in the message set. A flow can be constructed to reject and handle non-compliant messages. Additionally, complex manipulation of message data can be performed using extended SQL (ESQL), Java, and PHP facilities, which are provided in the WebSphere Message Broker Toolkit.

For information about the options for transport protocol conversion with WebSphere Message Broker, see 3.4, “Message transformation with message flows” on page 41, and 3.5, “Message enrichment with message flows” on page 51.

WebSphere Transformation Extender can be integrated into the WebSphere Message Broker ESB solution to extend the existing capabilities and to simplify transformation development.

2.2 Runtime architecture of WebSphere Message Broker

WebSphere Message Broker consists of a development environment on which message flows and message sets are designed and developed and a runtime environment on which the message flows executes. Figure 2-1 on page 17 shows an overview of the WebSphere Message Broker architecture.

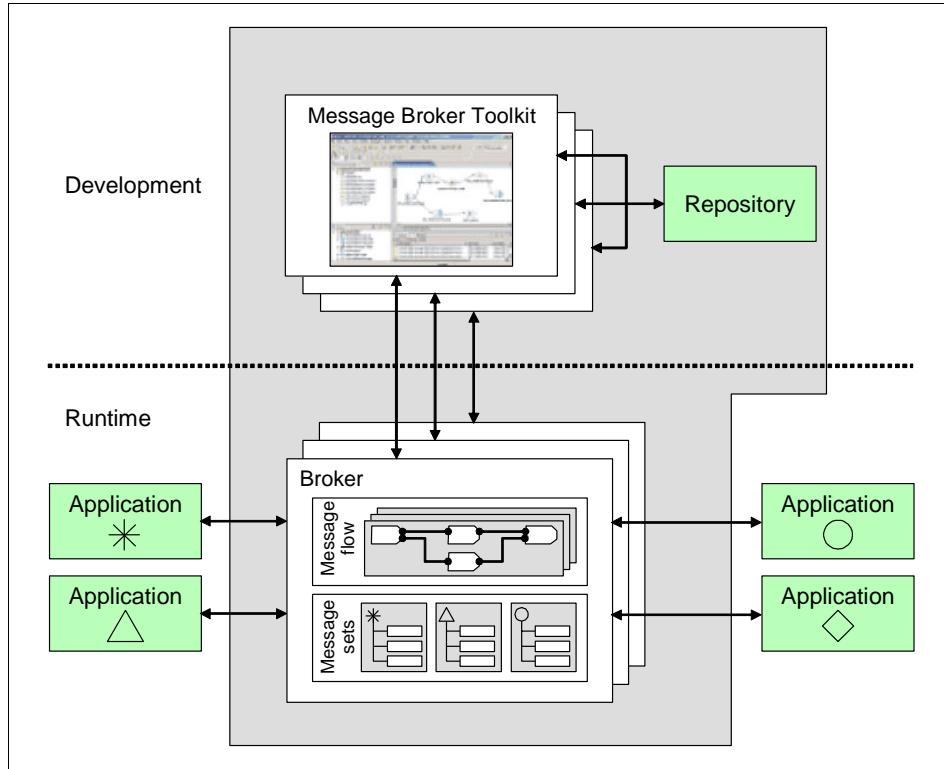


Figure 2-1 Architecture of WebSphere Message Broker

2.2.1 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. Each broker uses an internal repository on the local file system to store the message flows, configuration data, and the message sets that are deployed to it.

2.2.2 Execution groups

Execution groups are processes that host message flows. The execution groups facilitate the grouping of message flows within the broker with respect to functionality, load balancing, or other qualifications that are determined to be necessary. Each broker contains a main execution group. Additional execution groups can be created if they are given unique names within the broker.

Each execution group is a separate operating system process. Therefore, the contents of an execution group remain separate from the contents of other execution groups within the same broker. This separation can be useful for isolating pieces of information for security because the message flows execute in separate address spaces or as unique processes. Execution groups can also be used to scale the broker, for example, on multi-processor platforms different execution groups can be assigned to various processors.

Message flow applications are deployed to a specific execution group. To enhance performance, you can deploy additional message flow instances to an execution group. Each additional instance allocates an additional thread to host the message flow application. The number of threads can be increased on a per message flow basis, based on the processing requirements of that message flow. For each additional instance, an additional thread is started in the execution group to process the incoming messages at run time.

Note: For more information about designing for performance in WebSphere Message Broker, refer to the SupportPacs *IP04* at:

http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg24006518&loc=en_US&cs=utf-8&lang=en

2.3 Development environment of WebSphere Message Broker

In this section, we provide a introduction to the development environment of WebSphere Message Broker and the various mediation capabilities that it offers.

2.3.1 WebSphere Message Broker Toolkit

The WebSphere Message Broker Toolkit is an integrated development environment (IDE) and graphical user interface (GUI) based on the Eclipse platform. Application developers use the WebSphere Message Broker Toolkit. Using it they can create message flows and the associated artifacts and deploy them to the execution groups.

Broker Application Development perspective

The Broker Application Development perspective is the default perspective that is displayed the first time that you start the WebSphere Message Broker Toolkit. Application developers work in this perspective to develop and modify message

sets and message flows. Figure 2-2 shows the Broker Application Development perspective with a message flow open in the Message Flow editor.

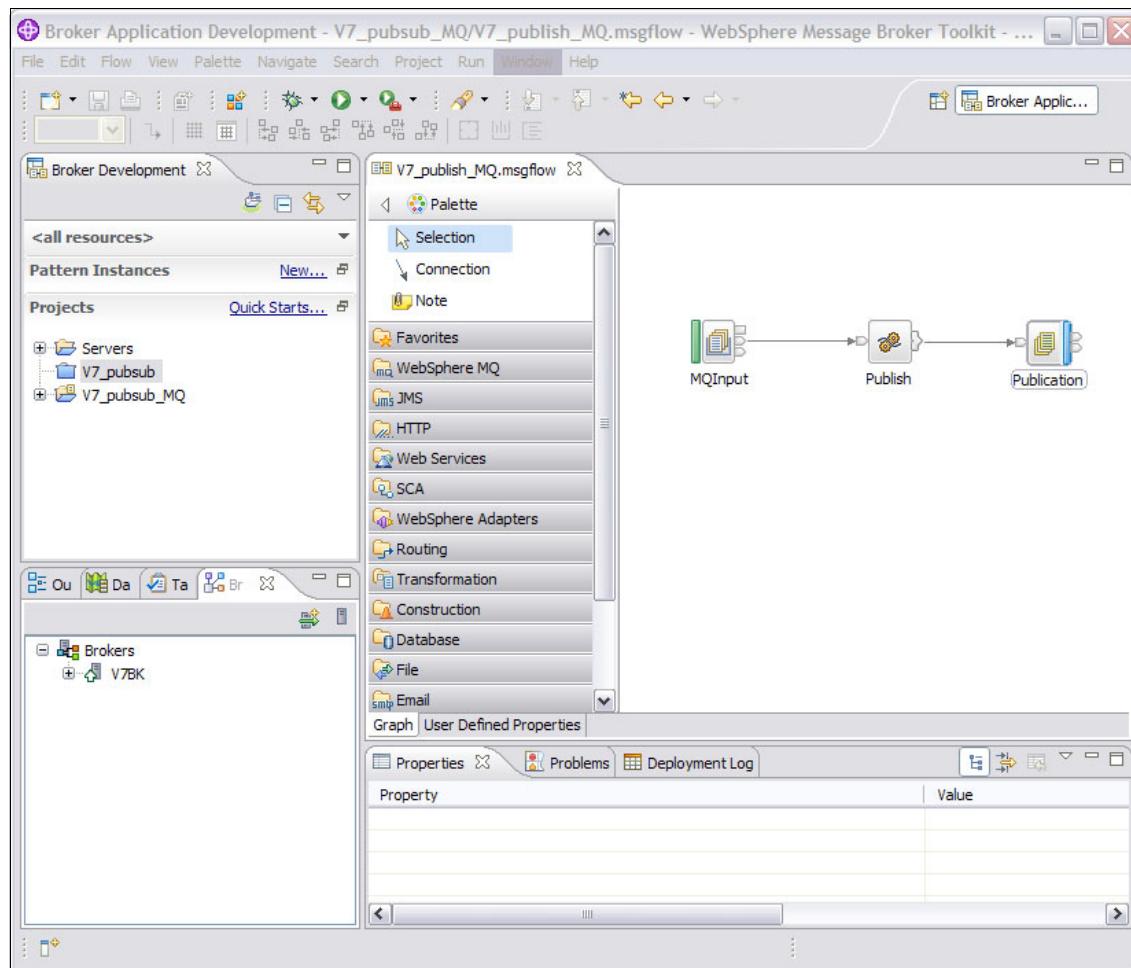


Figure 2-2 Application development perspective

2.3.2 Message flows

Message flows are created by adding nodes to the workspace, defining properties for these nodes, and wiring the nodes together in a logical flow. Table 2-1 on page 20 lists the nodes that the WebSphere Message Broker Toolkit provides for message flow development.

You can find details about these nodes in the WebSphere Message Broker Information Center at:

<http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp>

You can add nodes by using SupportPacs. For a list of SupportPacs, see the Business Integration - WebSphere MQ SupportPacs page at:

<http://www-1.ibm.com/support/docview.wss?rs=849&uid=swg27007205>

Table 2-1 Available node types in a WebSphere Message Broker

Category	Input/output type node	Flow development nodes
WebSphere MQ	MQInput node MQOutput node	MQReply node MQGet node MQHeader node MQOptimizedFlow node (deprecated)
JMS	JMSInput node JMSOutput node	JMSMQTransform node JMSReply node JMSHeader node MQJMSTransform node
HTTP	HTTPInput node HTTPRequest node HTTPReply node	HTTPHeader node
Routing		Filter node Label node RouteToLabel node Publication node AggregateControl node AggregateRequest node AggregateReply node Collector node Resequence node Sequence node
Transformation		Mapping node XSLTransform node Compute node JavaCompute node PHPCompute node

Category	Input/output type node	Flow development nodes
Construction		Trace node Input node Output node TryCatch node Throw node FlowOrder node Passthrough node ResetContentDescriptor node
Database		Database node DataInsert node DataUpdate node DataDelete node Warehouse node DatabaseRetrieve node DatabaseRoute node Extract node (Deprecated)
Validation		Check node (Deprecated) Validate node
Timer		TimeoutControl node TimeoutNotification node
SCA	SCAInput node	SCAReply node SCARequest node SCAAsyncRequest node SCAAsyncResponse node
Web Services	SOAPInput node	SOAPReply node SOAPRequest node SOAPAsyncRequest node SOAPAsyncResponse node SOAPEnvelope node SOAPExtract node Registry Lookup node Endpoint Lookup node
WebSphere Adapters	PeopleSoftInput node SAPInput node SiebelInput node TwineBallInput node	PeopeSoftRequest node SAPRequest node SAPReply node SiebelRequest node TwineBallRequest node
File	FileInput node FileOutput node	
Email	EmailOutput node	

Category	Input/output type node	Flow development nodes
TCP/IP	TCPIPCientInput node TCPIPCientOutput node TCPIPServerInput node TCPIPCientOutput node	TCPIPCientReceive node TCPIPServerReceive node
IMS		IMSRequest node

2.3.3 Message

A *message* is a collection of data that one application sends to another. This data can be business data or elements that are arranged in a predefined structure or a sequence of bytes.

When a message arrives at a message flow input node in a broker, the parser that is configured at the input node interprets and creates a logical tree representation from the bitstream of the message data, which is known as the message assembly. The tree format can then be easily manipulated and updated within the message flow using ESQL, Java, or PHP. After the message is processed, the parser converts it back into a bitstream at the output. The message assembly consists of four tree structures:

- ▶ Message tree structure: The message tree includes all the headers in the message, Properties subtree, and the message body.
- ▶ Environment tree structure: The environment tree stores information in the form of variables as the message passes through the message flow. It is typically used to store information regarding accounting and statistics or correlation attributes that are associated with broker monitoring events. The environment tree is always included in the message and its contents are retained in the output message.
- ▶ Local environment tree structure: This tree also stores information in the form of variables; however, these variables can be referenced or updated in the message flow. Local environment tree structure comprises of several subtrees, such as variables, destination, file, SOAP, service-registry, wildcard, adapter, TCP/IP, or a written destination.
- ▶ Exception list tree structure: This tree contains information about exceptions that can occur during message processing. The exception list tree is typically

built when an exception is thrown at which time the message processing is suspended.

Note: For more information about the Logical tree structure, refer to the following topic in the Information Center:
http://www publib boulder ibm com infocenter wmbhelp v7r0m0 topic com ibm etools mft doc ac00490_.htm

2.3.4 Message modeling

Message modeling is a way to predefine the message formats that are used by sending or receiving applications in an MQ infrastructure. Modeling is usually used for message formats, which are not self-defining, and so the parser must have access to the predefined model that describes the message. It might also be used for runtime validation of incoming messages and enhanced parsing of XML messages. It is typically done using the following components:

- ▶ Message set: A message set is a template that predefines the structure of the messages that a message flow processes. It provides a way to perform logical grouping of messages and its objects. A message set contains messages, elements, types, and groups. It provides message model information that is common to all of the messages in the message set.
- ▶ Message definition file: This file describes the logical structure and physical format of a message, such as XML, Text (also known as Tagged Delimited String), or binary (also known as Custom Wire Format). It is held externally to the message in a message set, and when the message set is deployed, the definition is compiled into a dictionary. It contains messages, elements, types, and groups that make up a message set.
- ▶ Web Services Description Language (WSDL) file: WSDL is an XML format for describing a Web service, describing the composition and interfaces used by the service requesting, and service provider applications.

2.3.5 Pattern instances

Patterns are reusable solutions that encapsulate a tested approach in solving common architectural, deployment, and design tasks in a particular context. Patterns can be used to generate customized solutions to a recurring problem in a more effective way. Patterns in WebSphere Message Broker V7 can be referenced to provide guidance in implementing solutions. Pattern specifications typically include the description of identified solutions. Patterns greatly reduce the development cycles because the resources can be generated from a set of predefined templates, thus increasing the efficiency. Patterns improve the quality

through the asset reuse and the implementation of error handling and logging functions.

WebSphere Message Broker Toolkit V7 displays Pattern instances in the Patterns Explorer, which provide a list of patterns and details on each of the patterns. The description in the Help can guide you towards a suitable pattern to solve a particular problem. Figure 2-3 shows the list of available Patterns that are provided in the Pattern Explorer.

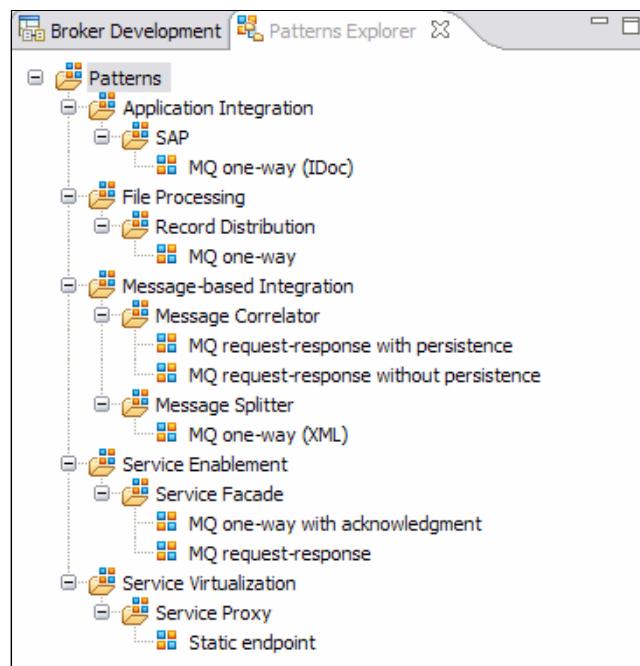


Figure 2-3 Patterns Explorer

Patterns: For more information about using patterns and pattern categories, search ‘Patterns’ in the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp?topic=/com.ibm.etools.mft.doc/ac67850_.htm

For an example of using Patterns and its application in a real-world scenario, see Chapter 6, “Enterprise Resource Planning integration with WebSphere Message Broker” on page 143. Also, see Chapter 8, “Scenario: File processing” on page 279.

You can get additional information about WebSphere Message Broker Patterns from the following developer works articles:

<http://www.ibm.com/developerworks/wikis/display/esbpatterns/>

<http://www.ibm.com/developerworks/library/ws-enterpriseconnectivitypatterns/index.html>

http://www.ibm.com/developerworks/websphere/library/techarticles/0910_phillips/0910_phillips.html

2.4 Administering WebSphere Message Broker

In this section, we provide an introduction to administering WebSphere Message Broker.

2.4.1 WebSphere Message Broker Explorer

The WebSphere Message Broker Explorer is a tool for administrators and provides the capability for enhanced WebSphere Message Broker monitoring and management.

WebSphere Message Broker Explorer is installed as a plug-in for WebSphere MQ Explorer. The Brokers view is added to the MQ Explorer-Navigator pane. The broker administration tasks can then be performed from this Brokers view.

Using the WebSphere Message Broker Explorer, Figure 2-4 on page 27, the user can administer their brokers and WebSphere MQ queue managers, in the same toolkit.

Additionally, the new WebSphere Message Broker Explorer provides the following new capabilities:

- ▶ Create, delete, start, and stop local brokers without using the command line.
- ▶ See the relationships between the brokers and queue managers.
- ▶ Deploy a broker archive file to multiple execution groups in one step.
- ▶ Visualize a brokers accounting and statistics data.
- ▶ Configure broker properties, including creating and modifying configurable services for broker communication with external services, such as SMTP, JMS providers, and Adapters.
- ▶ View the broker administration queue and remove pending tasks that were submitted to the broker.
- ▶ Connect and configure DataPower security settings.

The WebSphere Message Broker Explorer also provides a number of QuickViews that you can use to view the properties of brokers and their resources. These views are automatically displayed in the Brokers folder in the MQ Explorer, Navigator view. There is also a QuickView for viewing the details of broker archive files that are imported into the WebSphere Message Broker Explorer.

Figure 2-4 on page 27 shows the WebSphere Message Broker Explorer.

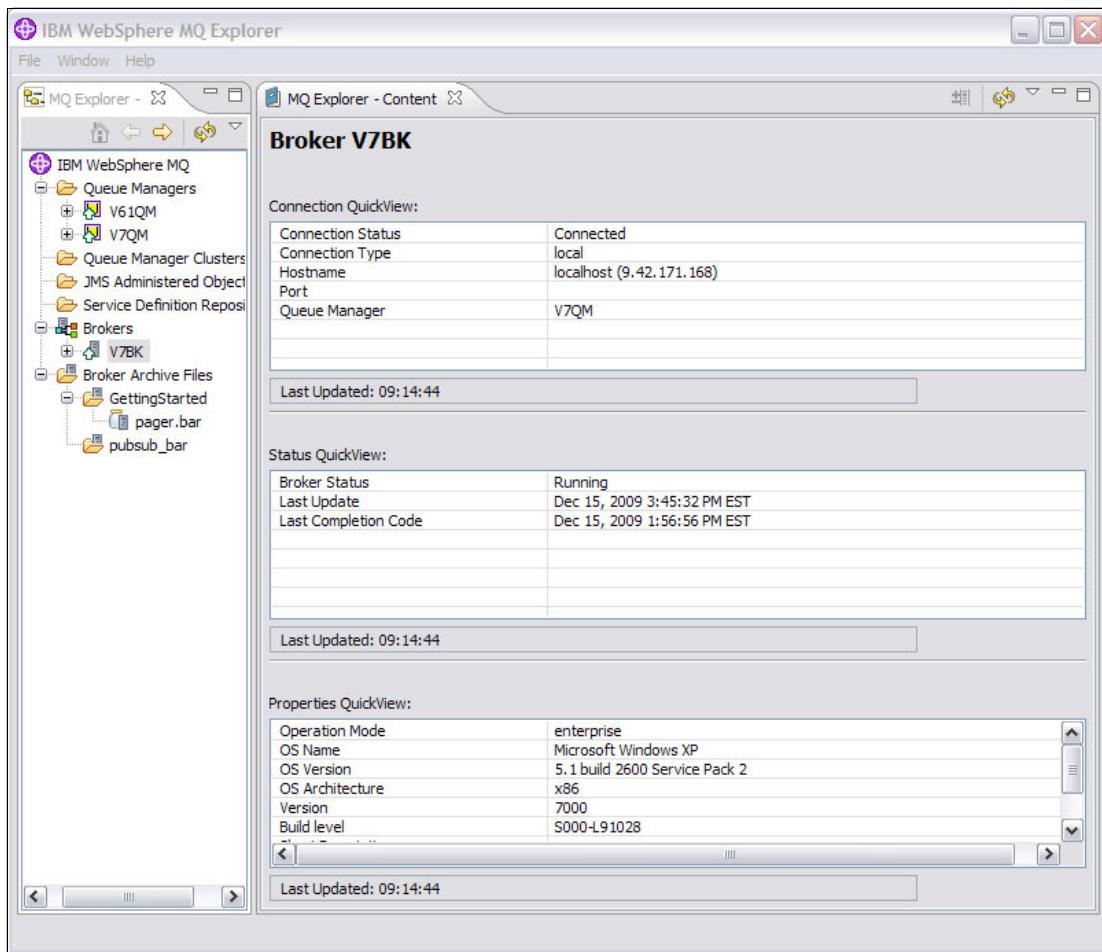


Figure 2-4 WebSphere Message Broker Explorer

In addition to the WebSphere Message Broker Explorer, there are command-line and API-based utilities that you can use to accomplish broker administration tasks, for example, the CMP API Exerciser sample that WebSphere Message Broker provides utilizes the Java administration API. You can use the CMP API Exerciser to view and manage brokers and their execution groups. You can also use it to create, modify, and delete configurable services and general broker administration tasks. The Java administration API is also available to users for scripting broker administration tasks.

2.4.2 Broker sets

You can use a broker set to visually group your brokers in the WebSphere Message Broker Explorer. If you have a large number of brokers displayed, it might be helpful to group the brokers using broker sets. Use broker sets to separate the development, test, QA, and production brokers into logical groups. You can create manual broker sets, automatic broker sets, or a combination of both.

A manual broker set is empty until you add brokers to the broker set from the All broker set or another broker set. You can add or remove brokers from the manual broker set at any time.

An automatic broker set uses broker tags to dynamically add and remove brokers from a set that is based on values that you provide or the current state of brokers.

2.5 Deploying message flow applications

Message flow applications contain message flows and the message sets that are comprised of the message definitions that are used to model the messages within the message flows. Message flow applications can be deployed to the execution groups of the brokers by first adding the components of the message flow application to a *Broker Archive file* (BAR file) and then deploying the bar file to the broker's execution group. You can deploy the BAR files using the command line or with the WebSphere Message Broker Toolkit. Alternatively, WebSphere Message Broker Toolkit provides the capability to deploy the message flows directly to an execution group without first adding them to a BAR file, which results in the deployment of all of the message flow dependent resources.

You can create a bar file using a graphical interface in the WebSphere Message Broker Toolkit from which you can select the components to include. If properties, such as queue names or other configurable properties, are promoted to the bar level, the broker archive editor offers options to configure these properties as corresponding flows are added to the bar file.

For more information about promoted properties, see the WebSphere Message Broker online help system at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.e.tools.mft.doc/ac00640_.htm

The bar file can also be deployed to the broker's execution group using a command line interface. The corresponding command is **mqsideploy**. For more information, search for the **mqsideploy** command topic in the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/an09020_.htm

You can also deploy a bar file using WebSphere Message Broker Explorer. Drag-and-drop the BAR file from the Broker Archive Files folder to the execution group.



Mediation with WebSphere Message Broker

In this chapter, we introduce the mediation capabilities of WebSphere Message Broker and describe how messages are routed, transformed, and enriched using a large variety of formats and protocols that WebSphere Message Broker supports.

Message flows are used to perform mediation functions on messages that flow through WebSphere Message Broker. We address the following basic mediation functions in this chapter:

- ▶ “Routing with message flows” on page 32
- ▶ “Aggregation with message flows” on page 37
- ▶ “Protocol transformation with message flows” on page 39
- ▶ “Message transformation with message flows” on page 41
- ▶ “Message enrichment with message flows” on page 51

3.1 Routing with message flows

Routing a message involves sending an incoming message to a destination that is based on criteria. The destination can be predefined (static) or based on information that is obtained at the time of the message flow (dynamically).

A common routing pattern includes a dynamic lookup of the destination based on the incoming message type and routing the message to that destination. This routing pattern typically consists of the steps shown in Figure 3-1.

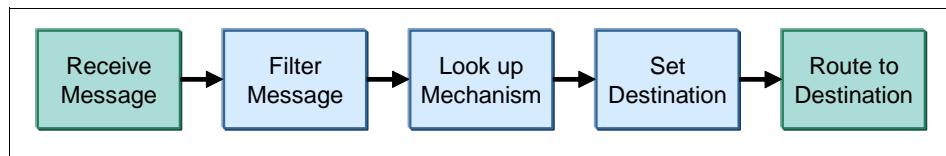


Figure 3-1 A typical routing mechanism

In this section, we discuss the following ways to implement routing in message flows and the nodes that are available to achieve it:

- ▶ Routing using JavaCompute Out and Alternate terminals
- ▶ Routing using the RouteToLabel node
- ▶ Routing using an application database
- ▶ Routing using Compute node

Aggregation is an advanced form of routing with a request/reply flavor. Refer to 3.2, “Aggregation with message flows” on page 37, which discusses this type of mediation.

3.1.1 Routing using JavaCompute Out and Alternate terminals

The JavaCompute node includes two-directional routing capability through two output terminals, labeled in the toolkit as the Out and Alternate terminals, as shown in Figure 3-2 on page 33.

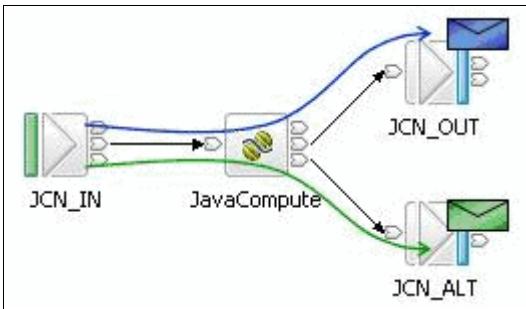


Figure 3-2 Using JavaCompute Out and Alternate terminals

The code extract in Example 3-1 gets the messages from these terminals, so they can be propagated within this method. The disadvantage of this technique is that it only routes to two terminals.

Example 3-1 Propagating to the Out terminal from the JavaCompute node

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

3.1.2 Routing using the RouteToLabel node

As shown in Figure 3-3, the JavaCompute node can be used with a RouteToLabel node and one or more Label nodes to dynamically determine the route that a message takes through the message flow. This flow illustrates an example where the JavaCompute node determines to which label node the RouteToLabel node propagates the message.

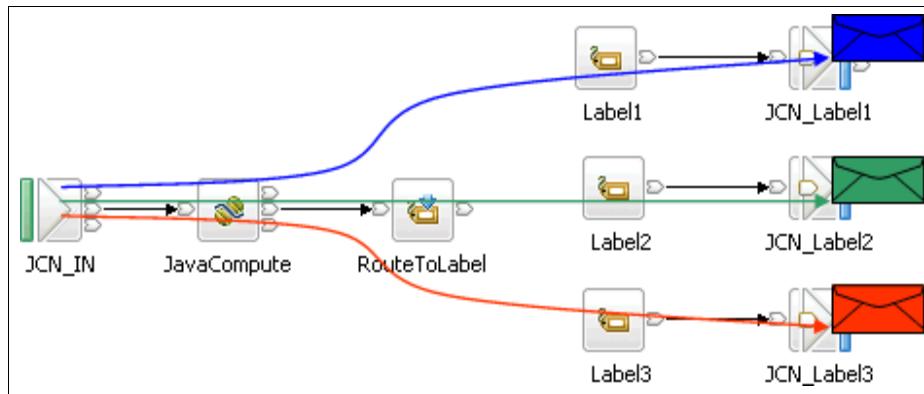


Figure 3-3 Usage of the *RouteToLabel* and *Label* nodes for routing

3.1.3 Routing using an application database

Using an application database table is the recommended way to store any form of routing or transformation data because:

- ▶ It removes business data from the ESQL or Java code in the message flow.
- ▶ It allows for updates in the routing data without changing or redeploying message flows.
- ▶ It makes ESQL or Java easier to read by removing the need to have large If ... then... else type of code structures.
- ▶ It allows routing information to be stored in one place but used by many message flows and possibly other applications.

The following approaches use an application database as a means to find routing information:

- ▶ Directly accessing the application database table

The message flow in Figure 3-4 on page 35 uses a Compute node to look up the destination and route the message to the target.

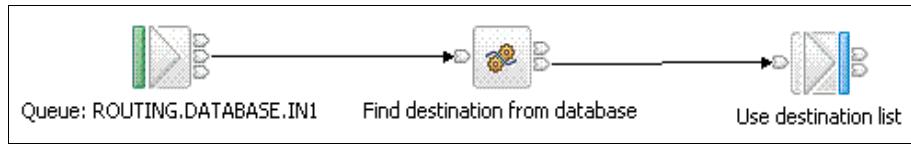


Figure 3-4 Application database lookup of routing information

The disadvantage of this method is that performance overhead is incurred to access the application database each time a message passes through a message flow.

- ▶ Using an in-memory cached version of the application database table

Starting with WebSphere Message Broker V6.0, database tables can be stored in-memory in a message flow using shared variables, which removes the need for continued access to the application database table. This method provides the advantages of using a database table without the performance overhead.

The cached database table can be refreshed by restarting the message flow or by sending a refresh message to the message flow. This approach overcomes the disadvantage here that the cache will not pick up changes to the database table as they happen. To pick up the changes, you must either restart the message flow or add a refresh mechanism to the message flow, as shown in Figure 3-5 on page 36.

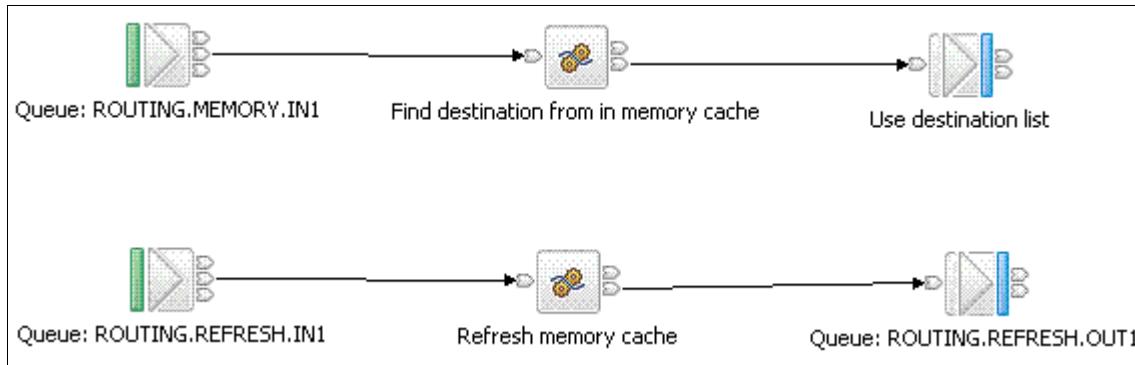


Figure 3-5 Routing with lookup using shared variables and refreshing the cache

Examples: The following examples illustrate routing using an application database:

- ▶ The WebSphere Message Broker samples gallery includes a working example of the message flow that we show in Figure 3-4 on page 35. In the WebSphere Message Broker Toolkit, select **Help → Samples and Tutorials → WebSphere Message Broker Toolkit - Message Broker → Application Samples → Message Routing sample.**
- ▶ The WebSphere Message Broker samples gallery includes a working example of the routing flow in Figure 3-5 on page 36. In the toolkit, select **Help → Samples and Tutorials → WebSphere Message Broker Toolkit - Message Broker → Application Samples → Message Routing sample.**

<http://www-redbooks.ibm.com/abstracts/swg247527>

As an alternative to the application database, you can also use WebSphere Service Registry and Repository (WSRR) to retrieve entities, such as endpoints, user-defined concepts, and definitions that are related to Web Services. The message flows can also route messages to WebSphere MQ queue destinations that are hosted in WebSphere Service Registry and Repository.

3.1.4 Routing using a Compute node

A Compute node can be used to create the output message by modifying the input message using ESQL or by using new information that can be taken from a database or other sources and route it further down the message flow. The compute node has one Failure and five Out nodes (Out, Out1, Out2, Out3, Out4) to which the transformed message can be routed. The message flow in

Figure 3-4 on page 35 uses a Compute node to look up the destination and route the message to the target queues using a destination list. Figure 3-6 shows an example of a Compute node routing messages to multiple destinations that might not all be queues.

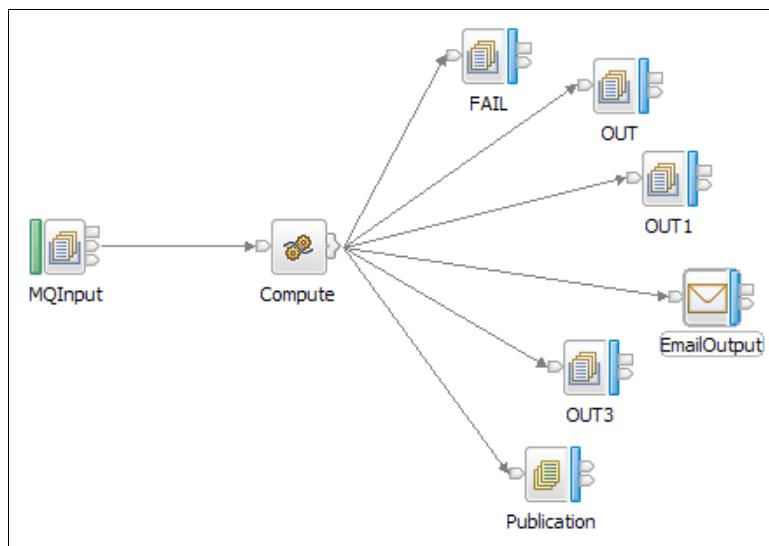


Figure 3-6 Compute node routing to multiple destinations

The message might be transformed and routed to queue destinations.

The message can also be routed to a publication node that publishes the message to the queue manager having the subscription against the topic on which the message is published.

The message might also be sent to an EmailOutput node, which is configured to send e-mail updates to its recipients.

3.2 Aggregation with message flows

Aggregation is a mediation concept that involves collecting the responses from two or more applications to fulfill a request from the requester. The responses are merged into a single message and sent to the requester or to another target application.

We can describe aggregation in message flow applications in three phases:

1. A message is received by the Message Broker. Based on this message, the broker requests information from one or more applications. The following examples are of initial message processing:
 - The initial message is a request that can be forwarded as a request to begin the aggregation.
 - The initial message requires parsing so that parts of the message can be used to build requests, where each request is sent to the corresponding application whose response is aggregated with the others.
 - The initial message might need to be enriched to build the aggregation request.
 - The initial message might not have arrived over the MQ transport, so it must be transformed before the request is built.
2. After the message is processed and the requests are built, the requests are sent to the respective applications. The process of sending the requests is referred to as the *fan-out process*. The fan-out process can involve sending the request to any application that supports the MQ request-reply interaction model. Two built-in nodes are supplied for use in the fan-out flow: The AggregateControl node and the AggregateRequest node.

If an application does not support the MQ request-reply interaction model, another message flow can be used to act as an interface to the application whose response is needed for the application, for example, if a Web Service response is needed for the aggregation, an additional message flow can be introduced to act as an interface between the aggregation message flow application and the Web service by implementing the Web Service with an MQ interface.

For more information about how to expose a Web service as an MQ application service, search on “Broker implements non-web-service interface to new Web service” in the WebSphere Message Broker Information Center:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac34560_.htm

3. The broker waits for the responses to the requests and compiles the responses into a single message. This process is referred to as the *fan-in process*. When the fan-in process completes, the message is delivered to its destination. The AggregateReply node is provided for this function.

Example:

The WebSphere Message Broker samples gallery includes a working example of the aggregation flows described in this section. In the WebSphere Message Broker Toolkit, select **Help → Samples and Tutorials → WebSphere Message Broker Toolkit - Message Broker → Control and Routing → Aggregation sample**.

You can design and configure a message flow that provides a similar function without using the aggregate nodes by issuing the subtask requests to another application (for example, using the HTTPRequest node) and recording the results of each request in the LocalEnvironment. After each subtask completes, you merge the results from the LocalEnvironment in a Compute node and create the combined response message for propagating to the target application. However, all of the subtasks are performed sequentially and do not provide the performance benefits of parallel operation that you can achieve by using the aggregation nodes.

3.3 Protocol transformation with message flows

An SOA solution uses an ESB to promote interoperability between disparate applications and systems. This interoperability often necessitates protocol and message transformation so that applications can communicate. Figure 3-7 on page 40 illustrates a small subset of the protocols that WebSphere Message Broker V7.0 supports.

WebSphere Message Broker provides support for a variety of transport protocols for both inbound and outbound connectivity that extends the scope of a common ESB. Protocol switching with WebSphere Message Broker essentially means that a message flow receives a message over one transport type and sends it out over another. Protocol switching is automatic. However, the message format might also need to be modified for the target system. See 3.4, “Message transformation with message flows” on page 41.

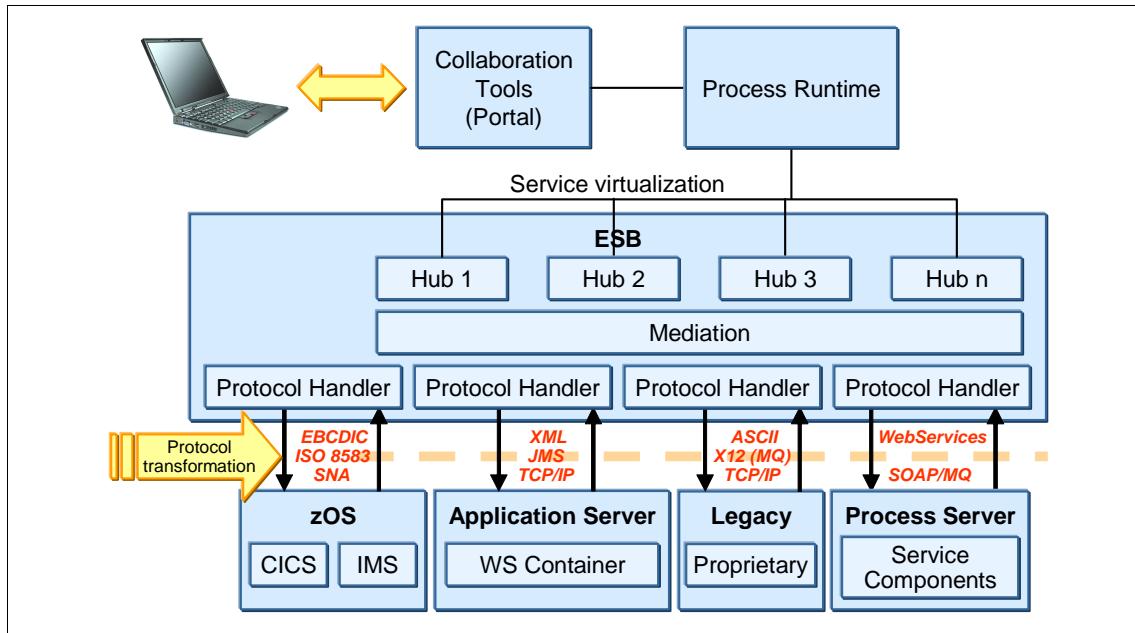


Figure 3-7 Sample protocol switching topology

The transport types that WebSphere Message Broker supports are:

- ▶ WebSphere MQ
- ▶ Java Message Service (JMS)
- ▶ WebSphere MQ/JMS
- ▶ SOAP-based Web services
- ▶ HTTP and HTTPS
- ▶ File
- ▶ Java Connector Architecture (JCA) adapters for SAP, Siebel and PeopleSoft
- ▶ TCP/IP
- ▶ Service Component Architecture (SCA)
- ▶ User-defined

In Chapter 4, “Connectivity options for interoperability with WebSphere Message Broker” on page 55, we discuss the multiple connectivity options that are available with WebSphere Message Broker in greater detail.

The broker run time can be extended to support flat file protocols, such as high availability VSAM or QSAM databases. The extender packages also include SMTP handlers, FTP, and even plain TCP/IP sockets.

WebSphere Message Broker defines a set of extended nodes through SupportPacs, for example, IA11 and IA9Z that can be used to integrate applications of the enterprise information system.

SupportPacs: You can find the SupportPacs on the Web at:

<http://www-1.ibm.com/support/docview.wss?uid=swg27007197#5>

WebSphere Message Broker also interacts with WebSphere Transformation Extender for Message Broker, which provides an additional set of connectors.

3.4 Message transformation with message flows

The WebSphere Message Broker enables communication between sender and receiver applications that might use various message formats. The transformation between multiple message formats is managed by the WebSphere Message Broker and does not require the sender or receiver applications to know about each others' message structures.

Applications typically use a combination of messages, which includes those that are defined by the following structures or standards:

- ▶ C and COBOL data structures
- ▶ Industry standards such as X12, ACCORD AL3, EDIFACT, SWIFT, EDI or HL7
- ▶ XML DTDs or schemas
- ▶ SOAP
- ▶ CSV
- ▶ IDoc
- ▶ User Defined

Mediation flows must be able to transform a message from one format to another with acceptable throughput. Messages in WebSphere Message Broker can be transformed using one of the following ways:

- ▶ Compute node (ESQL)
- ▶ XSLTransform node (Extensible Stylesheet Language Transformations)
- ▶ Mapping node (graphical)
- ▶ JavaCompute node (Java)
- ▶ Runtime message set and definition
- ▶ PHPCompute node (PHP)

When a message arrives from a transport protocol wired to the message flow run time, the message bit stream is parsed using a physical format, such as XML. See Figure 3-8.

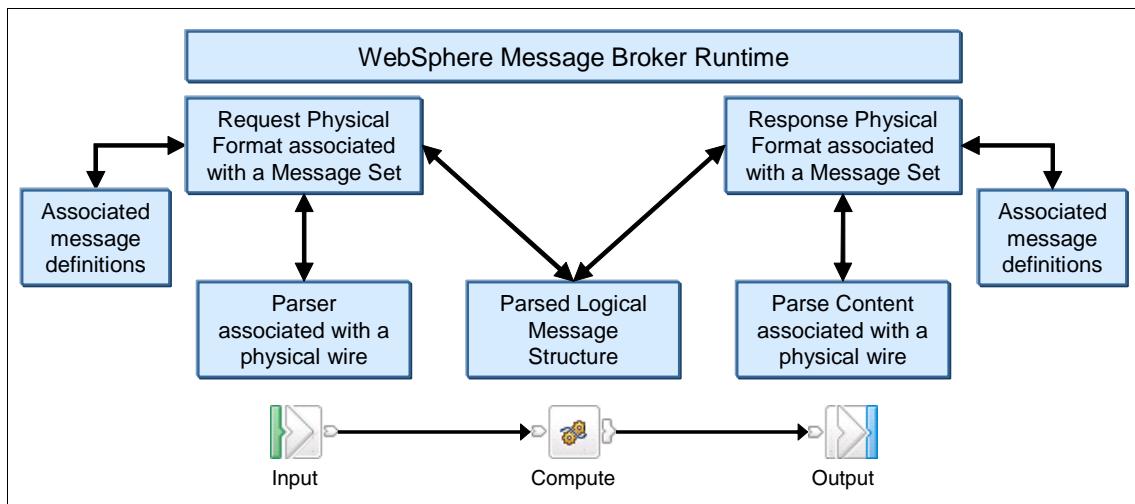


Figure 3-8 WebSphere Message Broker runtime architecture

When the message format is known, the broker parses an incoming message bit stream using the message set and definition that are defined on the flow configuration and converts it into a logical *message tree* for later manipulation. After the message is processed by the message flow, the broker converts the message tree back into a message bit stream. The transformation includes reformatting the message, concatenating the strings, or changing the element values.

The following physical formats are supported by the broker run time:

- ▶ **XML**

This format is the default runtime configuration. The message structure is validated and transformed using the parser specification that is defined inside the message flow.

- ▶ **Text (also known as TDS)**

The Text or tagged delimited string (TDS) physical format is designed to model messages that consist only of text strings. Examples of TDS messages are those that conform to the ACORD AL3, EDIFACT, HL7, SWIFT, and X12 standards. The TDS physical format allows a high degree of flexibility when defining message formats and is not restricted to modeling specific industry standards. Therefore, you can use the TDS format to model your own messages.

► **Binary** (also known as CWF)

The Binary or Custom Wire Format (CWF) is a physical representation of a message that is composed of a number of fixed format data structures or elements, which are not separated by delimiters. The CWF physical format is typically used to describe messages that are mapped to a C structure, a COBOL copybook, or any other programming language data structure definition.

Message set: A message set can have one or more physical formats on each XML, TDS, and CWF format.

WebSphere Message Broker typically supplies a range of parsers to parse and write message formats. Some message formats are self-defining and can be parsed without reference to a model. An example of a self-defining message format is XML. In XML, the message itself contains metadata and data values, enabling an XML parser to understand an XML message even if no model is available.

Most message formats, however, are not self-defining. That is, a binary message that originates from a COBOL program and a SWIFT formatted text message do not contain sufficient metadata to enable a parser to understand the message. The parser must have access to a model that describes the message to parse it correctly.

Table 3-1 lists the supported parsers for WebSphere Message Broker.

Table 3-1 WebSphere Message Broker parsers

Parser name	Parser description
MRM	For modeling a wide range of messages, including XML, fixed-format binary, and formatted text. This domain is usually used for enrichment of the message transformation, for example, creating COBOL cookbook using CWF.
XMLNSC XMLNS XML	For messages that conform to the W3C XML standard. The input bitstream must be a well-formed XML document that conforms to the W3C XML specification. XMLNSC is the preferred domain for generic XML messages and messages using XML namespaces.
DataObject IDOC	Used for messages going in and out of WebSphereAdapter nodes. The DataObject parser parses the business objects that are received from EIS. It is guided by XML schemas that model the EIS business objects. IDOC is implemented by using the SAP BAPI and the IDoc SAP format.

Parser name	Parser description
MIME	For handling multipart MIME messages, such as SOAP with attachments or RosettaNet.
BLOB	A parser that is used with messages that do not need to be interpreted in a logical message tree. The run time internally interprets the message as a BLOB bit stream. It is commonly used with messages that do not have a well-defined element tree structure, such as the ISO8583 standard.
JMSMap JMSStream	For modeling messages that are produced by the implementations of JMS standard. JMSMap domain can be used when handling JMS messages of type MapMessage. JMSStream domain can be used when handling JMS messages of type StreamMessage.
SOAP	Creates WSDL-based logical tree format to work with Web Services and validates incoming messages against a WSDL definition.

3.4.1 Message models

Message modeling is a way to predefine the message formats that applications use. WebSphere Message Broker uses message models to automatically parse and write message formats.

The components of a message model are:

- ▶ Message set projects
- ▶ Message sets
- ▶ Message definition files
- ▶ Message categories

The majority of the model content is described by *message definition files*. These files use the XML schema to represent the messages. Each message definition file describes both the logical structure of the messages and the physical format or formats that describes the appearance of the message bit stream during transmission.

If you are using the MRM domain, you must provide physical format information. This information tells the MRM parser how to parse the message bit stream. If you are using one of the XML domains, physical format information is not needed.

However, if your messages are self-defining and do not require modeling, there are still advantages to modeling them:

- ▶ Enhanced parsing of XML messages

Although XML is self-defining, without a model all data values are treated as strings. If a model is used, the parser knows the data type of data values and can cast the data accordingly.

- ▶ Improved productivity when writing ESQL

When you create ESQL programs for WebSphere Message Broker message flows, the ESQL editor can use message models to provide code completion assistance.

- ▶ Drag-and-drop message maps

When you create message maps for WebSphere Message Broker message flows, the Mapping editor uses the message model to populate its source and target views. Without message models, you cannot use the Mapping editor.

- ▶ More efficient way to implement Web sites

- ▶ PHP is a dynamic scripting language that is most frequently used to implement Web sites. PHPCompute node provides the ability to transform and route messages in WebSphere Message Broker V7.0.

- ▶ Runtime validation of messages

Without a model, it is not possible for a parser to check that input and output messages have the correct structure and data values.

- ▶ Reuse of message models in whole or in part by creating new messages based on existing messages.

- ▶ Automatic generation of documentation.

- ▶ Provision of version control and access control for message models by storing them in a central repository.

Tip: When creating output messages, the MRM parser can automatically generate the XML declaration and other XML constructs based on options in the model, which simplifies the transformation logic. For more information about when to use MRM or XMLNS(C) domains, search on “Which XML parser should you use” in the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp?topic=/com.ibm.etools.mft.doc/ad70530_.htm

3.4.2 Message transformation nodes

After a logical message structure is created, the implementing flow has full access to the parsed elements inside of the message. In the next sections, we describe the nodes that can be used to transform a message within a mediation flow.

JavaCompute node

A JavaCompute node can be implemented as a filtering class (filtering the input content like a Filter node, but using Java instead of ESQL) or as a modifying class (changing part of an incoming message or building a new output message that is independent of the input message). Depending on its content, a message can be propagated to one of the node's two output terminals (routing).

The Java code that is used by the node is stored in a Java project. Full debugging support is provided by using Eclipse Java debugger that is integrated with the Message Flow Visual Debugger.

Example 3-2 shows a message transformation using XPath in a JavaCompute node.

Example 3-2 Transforming a message using XPath in a JavaCompute node

```
...
// Evaluate XPath for the X12 legacy //
MbXPath xpSSN = new MbXPath("/?$$n[set-value($ssn)]", cliElement);
...
// define a value for the XPath variables //
xpSSN.assignVariable("ssn", ...);
...
// EVALUATE XPATH...
outMessage.evaluateXPath(xpSSN);
...
```

Example: For an example of using a JavaCompute node for transforming a message, visit:

<http://www-redbooks.ibm.com/abstracts/swg247527>

Some of the benefits of using a JavaCompute node for message transformation are:

- ▶ J2SE 1.6-based runtime environment
- ▶ Familiar interface for Java developers

- ▶ A wide range of class libraries available
- ▶ The ability to operate on any MRM or XML message using MbXPath or MbElement APIs
- ▶ The ability to off load to IBM System z Application Assist Processor (zAAP) on z/OS
- ▶ Transactional coordination with databases available when using MbSQLStatement but not when using Java Database Connectivity (JDBC) or SQL for Java (SQLJ)

XSLTransform node

The XMLTransform node uses XSLT to transform an XML message to another format, which might or might not be XML, according to the rules that are provided by an Extensible Stylesheet Language (XSL) style sheet.

XSLT is a W3C-standard, XML-based language for transforming XML data. The transformation is usually from one XML document to another, but the output does not have to be in XML. XSLT relies on XPath to access and manipulate elements within a message tree. WebSphere Message Broker supports the use of style sheets that conform to XSLT 1.0 using the XALAN-Java transformation engine.

If an enterprise is using XML to describe business data, XSLT might be the language of choice for transforming between formats. Many XSL transformations might exist already or developer skills might be best suited to creating transformations using this language.

WebSphere Message Broker V7 supports compiled style sheets, which improves the performance of the transformation.

The XSLTransform node depends on the inbound message being XML. It does not have to be in one of the XML domains, but it must be a well-formed XML bit stream, as shown in Figure 3-9.

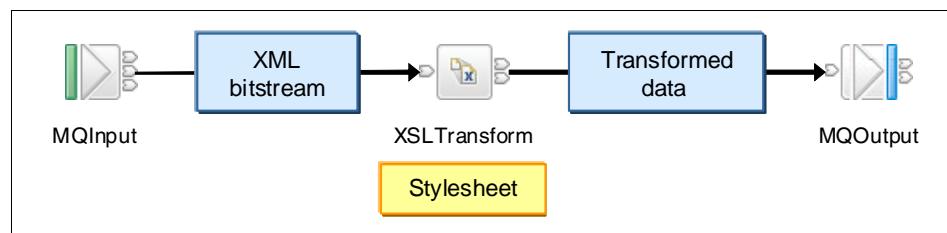


Figure 3-9 Message transformation using XSLT

Example: Visit the following Web site for an example of using this node:

<http://www-redbooks.ibm.com/abstracts/swg247527>

Some of the benefits of using the XSLTransformation node are:

- ▶ Reuse existing XSLT logic or create new XSLT logic for reuse elsewhere
- ▶ Advanced XML tooling is available inside the WebSphere Message Broker Toolkit
- ▶ You have the ability to off load to zAAP on z/OS
- ▶ Incoming data must be in the XML format. A range of output formats is possible
- ▶ Use the XSL debugger or node detailed trace option to debug transformations

Mapping node

The Mapping node is used to construct one or more new messages and populate them with new information without writing code to it. It can modify elements from the entire logical message structure (that is, the Properties structure, LocalEnvironment structure, ExceptionList structure, and others).

The Mapping node uses at least two message set definitions to map and change message content. One message set defines the receiving message structure and the other defines the resulting message structure. The Mapping node provides a graphical way of mapping between fields in an input message or database to one or more output messages or database tables.

Defining the message set: The source and target must be always defined as a database schema or MRM message set definition. The Mapping node cannot operate on other message domains.

Some of the benefits of mapping nodes are:

- ▶ It has an accessible graphical interface
- ▶ It is suitable for someone who is not an experienced Java, ESQL, or PHP developer
- ▶ It can produce rapid results at development time
- ▶ Input and output messages must be defined by a message set or database schema

ResetContentDescriptor node

Use the ResetContentDescriptor node to request that a message be re-parsed by another parser. If the new parser is MRM, you can also specify another message template (message set, type, and format). This node does not re-parse the message, but the properties that you set for this node determine how the message is parsed when it is next re-parsed by the message flow.

Message format conversion: This node does not convert the message from one format to another, for example, if the incoming message has a message format of XML and the outgoing message format is CWF, the ResetContentDescriptor node does no reformatting; instead, it invokes the parser to recreate the bit stream of the incoming XML message, which retains the XML tags in the message. When the message is re-parsed by a subsequent node, the XML tags are invalid, and the re-parse fails.

Some benefits of the ResetContentDescriptor node are:

- ▶ It can parse the message structure without changing its content
- ▶ It can parse the messages to all of the domains that are defined inside the run time

Compute node

Use the Compute node to construct one or more new output messages. These output messages might be created by modifying the information that is provided in the input message. Alternatively, the output messages might be created by using only new information, which might (or might not) be taken from a database. Elements of the input message (for example, headers, header fields, and body data), its associated environment, and its exception list can be used to create the new output message.

The messages are created using ESQL. As with the JavaCompute node, the Compute node must also copy the input structure to the output message structure. The input message cannot be modified during the node execution. Therefore, all changes must be done on the output message structure. Example 3-3 shows simple Compute node code.

Example 3-3 Simple Compute node code

```
CREATE COMPUTE MODULE Test_Flow_1_Compute
  CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
      -- CALL CopyMessageHeaders();
      -- CALL CopyEntireMessage();
      RETURN TRUE;
```

```

END;

CREATE PROCEDURE CopyMessageHeaders() BEGIN
    DECLARE I INTEGER 1;
    DECLARE J INTEGER;
    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;

```

Some benefits of the Compute node are:

- ▶ It is easy to manipulate the element tree structure
- ▶ It rapidly changes SQL code to the output message structure
- ▶ It easily transforms messages from one format to another
- ▶ It has five output terminals to route messages to multiple endpoints.

Examples: For an example of transformation using these nodes, visit the following Web address:

<http://www-redbooks.ibm.com/abstracts/swg247527>

PHPCompute node

Use the PHPCompute node to route and transform an incoming message using the PHP scripting language.

You can use this node to:

- ▶ Examine an incoming message and, depending on its content, propagate it unchanged to the node's output terminal
- ▶ Change part of an incoming message and propagate the changed message to the output terminal using PHP
- ▶ Create and build a new output message that is independent of the input message using PHP
- ▶ Access the file system using the file system functions that are provided in PHP for file handling

Example: For an example of using a PHPCompute node, see Chapter 8, “Scenario: File processing” on page 279 (including a PHPCompute node).

3.5 Message enrichment with message flows

Message enrichment involves taking an in-flight message and changing it before sending it to its destination. The following main sources of data are used to enrich a message:

- ▶ Application databases
- ▶ MQ queues
- ▶ Broker properties and environment values
- ▶ Files
- ▶ Web Services (SOAP, HTTP)

3.5.1 Database content to enrich messages

You can use the following built-in nodes to retrieve data from a supported application database and enrich an in-flight message:

- ▶ Compute node

A Compute node can be configured for database access. The ESQL code in the node is used to retrieve data from the database and to add this data to a message.

- ▶ JavaCompute node

A JavaCompute node can be configured for database access. The Java code in the node is used to retrieve data from the database and to add this data to a message.

- ▶ Mapping node

A Mapping node can be configured for database access. A message mapping file is used to map values that are taken from a database to the message.

You can find detailed information about databases that WebSphere Message Broker supports in the “Supported databases” topic in the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp?topic=/com.ibm.etools.mft.doc/ah10030_.htm

3.5.2 Messages on queues to enrich messages

You can use the MQGet node to enhance a message with all or part of the content of a message that is held on an MQ queue. The node can be included anywhere in a message flow:

1. The node receives an input message on its input terminal from the preceding node in the message flow.
2. The node issues an MQGET call to retrieve a message from a WebSphere MQ queue and builds a result message tree.
3. The node uses the input tree and the result tree to create an output tree that is then propagated to its Output, Warning, or Failure terminal, depending on the configuration of the node and the result of the MQGET operation.

3.5.3 Broker properties and environment values

Using the Compute or JavaCompute node, you can access the broker properties at run time and use the values to enrich a message. The categories of the broker properties are those relating to:

- ▶ A specific node
- ▶ Nodes in general
- ▶ A message flow
- ▶ The execution group

You can find detailed information about broker properties that are accessible through ESQL or Java in the “Broker properties” topic in the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp?topic=/com.ibm.etools.mft.doc/ak01012_.htm

You can find further information about SupportPacs on the WebSphere Message Broker Product Support Web site at:

<http://www-1.ibm.com/support/docview.wss?rs=849&uid=swg27007205>

3.5.4 Files

WebSphere Message Broker supplies built-in nodes to process data from files. Using these built-in nodes in WebSphere Message Broker, data from a file can be read in and the content can be used by a broker message flow to enrich a message. These nodes provide large file handling, batch processing, and stream parsing.

3.5.5 Web Services content to enrich messages

WebSphere Message Broker can participate in a Web services environment as a service requester, a service provider, or both. WebSphere Message Broker provides built-in SOAP and HTTP nodes. These nodes can send outbound requests or receive messages from other applications and make any required changes in the message flow. Also, WebSphere Message Broker can use WebSphere Service Registry and Repository to enrich and add value to SOA processing by adding information to enrich service artifacts. WebSphere Message Broker ships built-in primitive nodes to interact with WebSphere Service Registry and Repository. These nodes can retrieve the service data dynamically and enrich the outbound message in the message flow.



Connectivity options for interoperability with WebSphere Message Broker

WebSphere Message Broker, in its role as an enterprise service bus (ESB), supports a variety of transport protocols that make it ideally suited for providing universal connectivity in heterogeneous IT environments. In this chapter, we describe the connectivity options that are available in WebSphere Message Broker and how you can implement them.

Specifically, we cover the following topics in this chapter:

- ▶ “Messaging transport” on page 56
- ▶ “Connectivity options” on page 57

4.1 Messaging transport

Messaging is a loosely-coupled style of interaction among applications, and it involves an intermediary called a *messaging provider*. In this overview, we discuss the messaging concepts that you must know.

Messaging is an asynchronous communications protocol, but WebSphere Message Broker provides the option to model application communications both asynchronously and synchronously.

With *asynchronous messaging*, applications send messages to the messaging provider and continue processing without waiting for a response. The messaging provider delivers the message to the target application. The messaging provider offers the application quality of service levels that specify how the message delivery is managed, for example, the quality of service can determine if there is a guarantee that messages are delivered, that there are no duplicates, and that a message sequence is maintained.

With asynchronous messaging, sending and receiving applications do not need to know of each other's existence or the nature of the messages that each application understands. Each application is concerned with defining the format of the messages that it uses to communicate and with establishing access to the services that are offered by the messaging provider. The messaging provider provides services to dynamically route and optionally transform the message so that it is understood by the receiving application.

Although messaging is usually asynchronous, it is also possible to have *synchronous messaging*, which means that both applications are available simultaneously and communicate directly in a tightly-coupled manner. The sending application waits for a response before any further processing. Synchronous messaging can be used when the performance overhead that is added by introducing an intermediary is not justified or when guaranteed delivery is not as high a priority. Response time is often an important aspect of synchronous messaging. Synchronous messaging means that updates and changes to an application must be made known to its messaging partner.

Many applications must interact with one another in support of a business. These applications change as the business itself changes. Therefore, loosely-coupled applications that use a messaging provider can be key to enabling the exchange of data between disparate applications that are distributed across the enterprise. Not surprisingly, asynchronous messaging plays a significant role in service-oriented architecture (SOA).

WebSphere Message Broker provides support for a variety of messaging transport protocols, which includes WebSphere MQ, Java Message Service

(JMS), WebSphere MQ/JMS, SOAP-based Web services, HTTP and HTTPS, File, Java Connector Architecture (JCA) adapters for SAP, Siebel and PeopleSoft, TCP/IP, and Service Component Architecture (SCA).

4.2 Connectivity options

In this section, we introduce the various transport protocols that WebSphere Message Broker supports and describe the appropriate connection points for each transport type that client applications can use to send messages to, and receive messages from, WebSphere Message Broker.

4.2.1 WebSphere MQ

The WebSphere MQ Enterprise Transport supports WebSphere MQ applications that connect to WebSphere Message Broker to benefit from message routing and transformation options.

MQInput and MQGet nodes

Use the MQInput node if the messages arrive at the broker on a WebSphere MQ queue, and the node is to be at the start of a message flow.

Use the MQGet node to retrieve a message from a WebSphere MQ queue if you intend to get the message after the start of the message flow.

MQOutput and MQReply nodes

Use the MQOutput node if the target application expects to receive messages on a WebSphere MQ queue.

Use the MQReply node if the target application expects to receive messages on the WebSphere MQ reply-to queue that is specified in the input message MQMD.

Figure 4-1 on page 58 shows a sample message flow, which makes use of the MQInput, MQGet, MQReply, and MQOutput nodes.

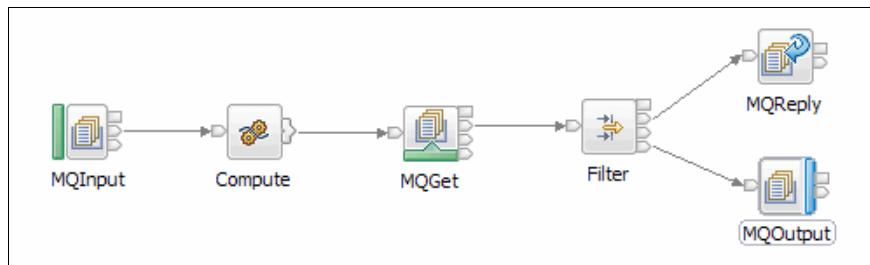


Figure 4-1 The MQ nodes

For additional information about configuring and using each of the WebSphere MQ nodes, go to the WebSphere Message Broker Information Center at:

<http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp>

In the WebSphere Message Broker Information Center, refer to the following topics:

- ▶ MQInput node (ac04560_)
- ▶ MQGet node (ac20806_)
- ▶ MQOutput node (ac04570_)
- ▶ MQReply node (ac04580_)

4.2.2 Java Message Service (JMS) 1.1

The WebSphere Message Broker JMS Transport sends and receives JMS messages using destinations accessible through a JMS provider. The build-in JMS nodes act as JMS clients.

The JMS nodes work with WebSphere MQ JMS provider, WebSphere Application Server, and any other JMS provider that conforms to the Java Message Service Specification V1.1.

JMSInput node

Use the JMSInput node if a JMS application sends the messages. The JMSInput node acts as a JMS message consumer and can receive all six message types that are defined in the JMS Specification V1.1.

JMSOutput and JMSReply nodes

Use the JMSOutput node if the messages are for a JMS destination. The JMSOutput node acts as a JMS message producer and can publish all six message types that are defined in the JMS Specification V1.1.

The JMSReply node has a similar function to the JMSOutput node, but the JMSReply node sends JMS messages only to the reply destination that is supplied in the JMSReplyTo header field of the JMS message tree. Use the JMSReply node to treat a JMS message that is produced from a message flow as a reply to a JMS input message and when you have no other routing requirements.

Figure 4-2 shows a sample message flow, which makes use of the JMSInput, JMSReply, and JMSOutput nodes.

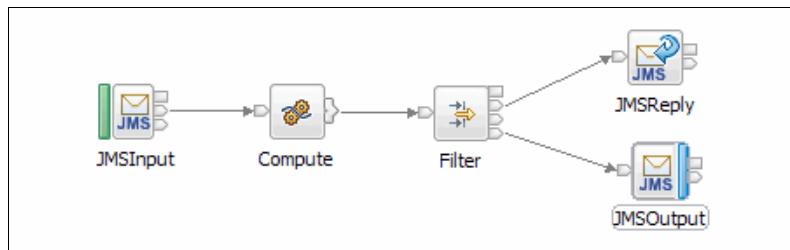


Figure 4-2 The JMS nodes

For additional information about configuring and using each of the JMS nodes, go to the WebSphere Message Broker Information Center at:

<http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp>

In the WebSphere Message Broker Information Center, refer to the following topics:

- ▶ JMSInput node (ac24820_)
- ▶ JMSOutput node (ac24830_)
- ▶ JMSReply node (ac37130_)

4.2.3 WebSphere MQ/JMS

WebSphere Message Broker provides support that includes the ability to transform MQ messages to JMS and vice versa.

WebSphere Message Broker can receive messages that have a WebSphere MQ JMS provider message tree format and transform them into a format that is compatible with messages that are to be sent to a JMS client application.

Conversely, WebSphere Message Broker can transform a message with a JMS message tree structure into a message that is compatible with the format of messages that are produced by the WebSphere MQ JMS provider.

MQJMSTransform node

Use the MQJMSTransform node if the input message is of the WebSphere MQ JMS provider format and the output message is to be sent to a JMS client application.

Figure 4-3 shows a sample message flow, which takes an MQ input message, transforms the message using the MQJMSTransform node, and outputs a JMS message.

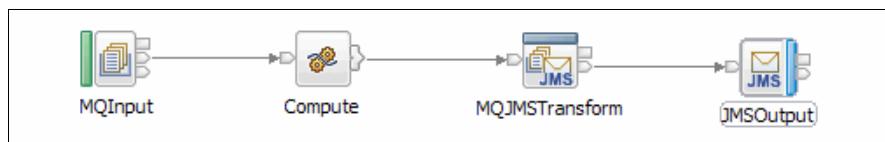


Figure 4-3 The MQJMSTransform node

JMSMQTransform node

Use the JMSMQTransform node if the input message is a JMS message and the output message must be in the format of messages that are produced by the WebSphere MQ JMS provider.

Figure 4-4 shows a sample message flow, which takes a JMS input message, transforms the message using the JMSMQTransform node, and outputs an MQ message.

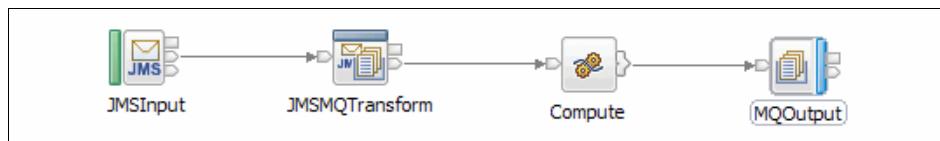


Figure 4-4 The JMSMQTransform node

For additional information about configuring and using each of the JMSMQ/MQJMS Transform nodes, go to the WebSphere Message Broker Information Center at:

<http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp>

In the WebSphere Message Broker Information Center, refer to the following topics:

- ▶ MQJMSTransform node (ac24850_)
- ▶ JMSMQTransform node (ac24840_)

4.2.4 SOAP-based Web services

The WebSphere Message Broker Web services support includes both provider and consumer scenarios, WS-Security and WS-Addressing support, and integration with DataPower appliances for Web service security. The standards that are supported include SOAP 1.1/1.2, WSDL 1.1, MTOM/XOP, SOAP with Attachments, and Basic Profile 1.1 compliant.

Integrated support for WebSphere Service Registry and Repository (WSRR) is also provided. Message flows can access, use, and select specific service registry entities dynamically at runtime.

SOAP input and request nodes

The SOAPInput, SOAPRequest, and SOAPAsyncRequest nodes can be used to process client SOAP messages and to configure the message flow to behave like a SOAP Web Services provider.

The SOAPInput node processes client SOAP messages, thereby enabling the broker to be a SOAP Web Services provider. It is typically used in conjunction with the SOAPReply node.

The SOAPRequest node sends a SOAP request to a remote Web service. The node is a synchronous request and response node and blocks after sending the request until the response is received.

The SOAPAsyncRequest node sends a Web services request, but the node does not wait for the associated Web service response to be received. It does, however, wait for the HTTP 202 acknowledgment before continuing with the message flow and blocks if the acknowledgement is not received. The SOAPAsyncRequest node is used with the SOAPAsyncResponse node to construct a pair of message flows that call a Web service asynchronously.

SOAP reply and response nodes

The SOAPReply node is used if the target application expects to receive SOAP messages in response to a message that is sent to the SOAPInput node.

The SOAPAsyncResponse node is used to deliver a SOAP message in response to a message that is received by the SOAPAsyncRequest node.

Figure 4-5 on page 62 shows a sample Web Service Provider message flow, which makes use of the SOAPInput and SOAPReply nodes.

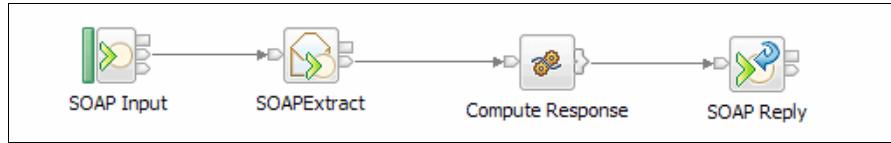


Figure 4-5 SOAP Web Service Provider message flow

Figure 4-6 shows a sample Web Service Consumer message flow, which makes use of the `SOAPRequest` node.



Figure 4-6 SOAP Web Service Consumer message flow

Figure 4-7 shows a sample Asynchronous Consumer message flow, which makes use of the `SOAPAsyncRequest` and `SOAPAsyncResponse` nodes.

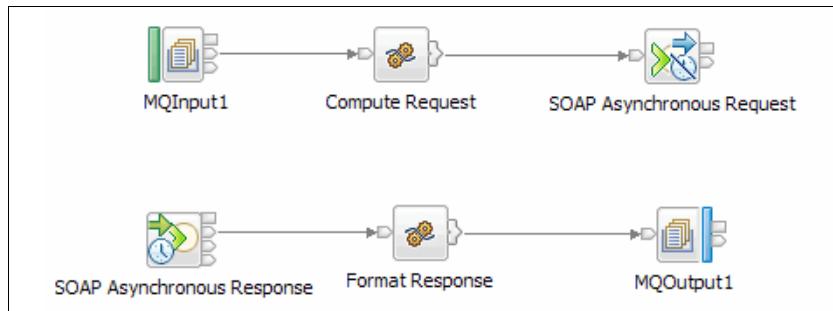


Figure 4-7 SOAP Asynchronous Consumer message flow

For additional information about configuring and using each of the SOAP nodes, go to the WebSphere Message Broker Information Center at:

<http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp>

In the WebSphere Message Broker Information Center, refer to these topics:

- ▶ [SOAPInput node \(ac56170_\)](#)
- ▶ [SOAPRequest node \(ac56190_\)](#)
- ▶ [SOAPAsyncRequest node \(ac56200_\)](#)
- ▶ [SOAPReply \(ac56180_\)](#)
- ▶ [SOAPAsyncResponse \(ac56210_\)](#)

4.2.5 HTTP and HTTPS

The WebSphere Message Broker HTTP Transport connects applications that use the HTTP protocol. Message flows can use the HTTP transport to work with SOAP-based Web services, other Web services standards, such as REST or XML-RPC, and general HTTP messaging, where the payload might or might not be XML.

In the case of SOAP-based Web services, several advantages exist if you use the SOAP domain instead of the HTTP transport nodes. Using the SOAP domain and nodes provides the following advantages:

- ▶ Support for WS-Addressing, WS-Security, and SOAP headers
- ▶ A common SOAP logical tree format that is independent of the actual bitstream
- ▶ Runtime checking against WSDL

HTTP input and request nodes

The **HTTPInput** node receives an HTTP message from an HTTP client for processing by a message flow. If you include an **HTTPInput** node in a message flow, you must either include an **HTTPReply** node in the same flow or pass the message to another flow that includes an **HTTPReply** node.

The **HTTPRequest** node interacts with a Web service using all, or part, of the input message as the request that is sent to that service. This node can be used in a message flow that does or does not contain an **HTTPInput** or **HTTPReply** node.

If you use the **HTTPInput** node with the **HTTPReply** and **HTTPRequest** nodes, the broker can act as an intermediary for Web services, and Web service requests can be transformed and routed in the same way as other message formats that WebSphere Message Broker supports. Web service requests can be received either in standard HTTP (1.0 or 1.1) format or in HTTP over SSL (HTTPS) format. You can select the **Use HTTPS** property to handle either HTTP or HTTPS requests.

HTTPReply node

The **HTTPReply** node returns a response from the message flow to an HTTP client. This node generates the response to an HTTP client from which the input message was received by the **HTTPInput** node and waits for the confirmation that it was sent.

Figure 4-8 on page 64 shows a sample Web Service Provider message flow, which makes use of the **HTTPInput** and **HTTPReply** nodes.

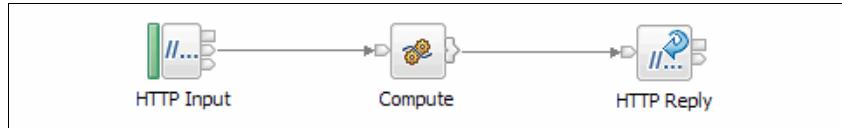


Figure 4-8 HTTP Web Service Provider message flow

Figure 4-9 shows a sample Web Service Consumer message flow, which makes use of the `HTTPRequest` node.

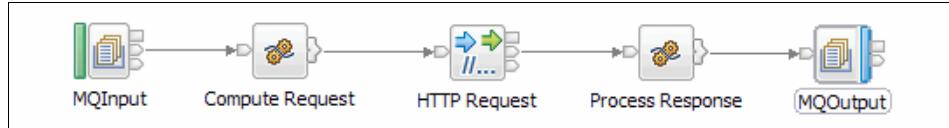


Figure 4-9 HTTP Web Service consumer message flow

For additional information about configuring and using each of the HTTP nodes, visit the WebSphere Message Broker Information Center at:

<http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp>

In the WebSphere Message Broker Information Center, refer to these topics:

- ▶ `HTTPInput` node (ac04565_)
- ▶ `HTTPRequest` node (ac04595_)
- ▶ `HTTPReply` node (ac04585_)

4.2.6 File

File processing support, including FTP and SFTP, is delivered through built-in nodes. This support allows large files to be handled without using excessive storage. It provides comprehensive support for record detection and sophisticated record identification that allows specific records to be identified in a larger flow.

FileInput node

The `FileInput` node can be used in any message flow that must accept messages in files. One or more messages can be read from a single file, and each message is propagated as a separate flow transaction. A file can be a single record or a series of records.

The `FTP -> Transfer protocol` property can be used to select either `FTP` or `SFTP`.

FileOutput node

The FileOutput node can be used to write messages to files. The node writes files as a sequence of one or more records. Each record is generated from a single message that is received on the In terminal of the node.

Figure 4-10 shows a sample File transport message flow. The FileInput node monitors the input directory and is connected to a FileOutput node that writes the file to the output directory.

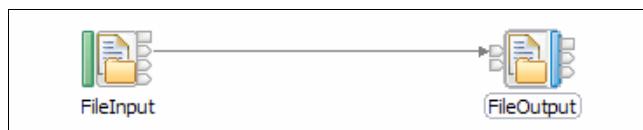


Figure 4-10 File nodes

For additional information about configuring and using each of the File nodes, go to the WebSphere Message Broker Information Center at:

<http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp>

In the WebSphere Message Broker Information Center, refer to these topics:

- ▶ FileInput node (ac55150_)
- ▶ FileOutput node (ac55160_)

File nodes: We describe the comprehensive message flow scenarios that make use of the File nodes in detail in Chapter 8, “Scenario: File processing” on page 279.

4.2.7 Java Connector Architecture (JCA) Adapters

The WebSphere Adapters nodes can be used to communicate with enterprise information systems (EIS). WebSphere Message Broker provides message flow nodes that support connectivity with the SAP, Siebel, and PeopleSoft adapters.

Enterprise Metadata Discovery (EMD) is used for key data structure discovery and accelerated message set generation. The WebSphere Adapters nodes need an adapter component to access the EIS. A license for the adapter components is required.

The WebSphere Adapters nodes can be used to interact with the Enterprise Information System. The SAP, Siebel, and PeopleSoft adapters are supported by the following message flow nodes in WebSphere Message Broker:

- ▶ SAPInput node
- ▶ SAPReply node
- ▶ SAPRequest node
- ▶ SiebelInput node
- ▶ SiebelRequest node
- ▶ PeopleSoftInput node
- ▶ PeopleSoftRequest node

Figure 4-11 shows the Toolkit icon representation for each of the WebSphere Adapter nodes.

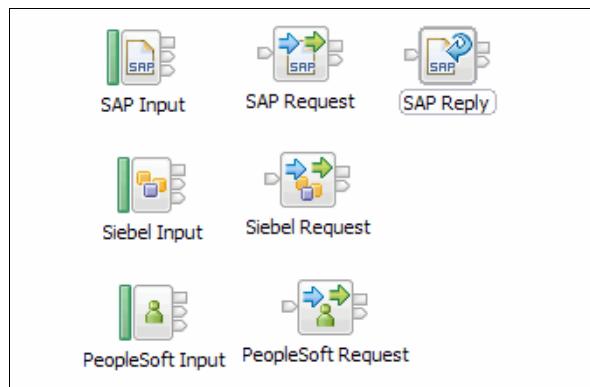


Figure 4-11 WebSphere Adapters nodes

The WebSphere Adapters' input nodes monitor an EIS for a particular event. When that event occurs, business objects are sent to the input node. The input nodes construct a tree representation of the business object, which the rest of the message flow can use.

The WebSphere Adapters' request nodes can send and receive business data. They request information from an EIS and propagate the data to the rest of the message flow.

For additional information about configuring and using each of the WebSphere Adapters nodes, go to the WebSphere Message Broker Information Center at:

<http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp>

In the WebSphere Message Broker Information Center, refer to these topics:

- ▶ SAPInput node (ac37290_)
- ▶ SAPReply node (bc22000_)
- ▶ SAPRequest node (ac37300_)
- ▶ SiebelInput node (ac37310_)
- ▶ SiebelRequest node (ac37320_)
- ▶ PeopleSoftInput node (ac37330_)
- ▶ PeopleSoftRequest node (ac37340_)

SAP WebSphere Adapters nodes: We describe the comprehensive message flow scenarios that make use of the SAP WebSphere Adapters nodes in detail in Chapter 6, “Enterprise Resource Planning integration with WebSphere Message Broker” on page 143.

4.2.8 TCP/IP

WebSphere Message Broker provides support for TCP/IP connectivity of existing applications that use raw TCP/IP sockets for transferring data.

TCP/IP Client nodes

The TCPIPCientInput node can be used to create a client connection to a raw TCP/IP socket and to retrieve data over that connection.

The TCPIPCientOutput node can be used to create a client connection to a raw TCP/IP socket and to send data over that connection to an external application.

The TCPIPCientReceive node can be used to receive data over a client TCP/IP connection.

TCPIP Server nodes

The TCPIPServerInput node can be used to create a server connection to a raw TCP/IP socket and to retrieve data over that connection.

The TCPIPServerOutput node can be used to create a server connection to a raw TCP/IP socket and to send data over that connection to an external application.

The TCPIPServerReceive node can be used to receive data over a server TCP/IP connection.

Figure 4-12 on page 68 shows the Toolkit icon representation for each of the TCP/IP nodes.

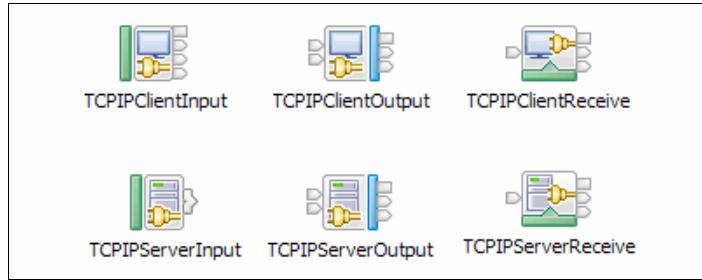


Figure 4-12 TCP/IP nodes

For additional information about configuring and using each of the TCP/IP nodes, go to the WebSphere Message Broker Information Center at:

<http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp>

In the WebSphere Message Broker Information Center, refer to these topics:

- ▶ TCPIPClientInput node (ac67300_)
- ▶ TCPIPClientOutput node (ac67310_)
- ▶ TCPIPClientReceive node (ac67320_)
- ▶ TCPIPServerInput node (ac67330_)
- ▶ TCPIPServerOutput node (ac67340_)
- ▶ TCPIPServerReceive node (ac67350_)

TCP/IP nodes: We discuss the comprehensive message flow scenarios that make use of the TCP/IP nodes in detail in Chapter 7, “Service enable TCP/IP-based applications with WebSphere Message Broker” on page 243.

4.2.9 Service Component Architecture

Service Component Architecture (SCA) simplifies creating and integrating SOA business applications by separating the business logic from its implementation, thus allowing the developer to focus on assembling an integrated application without knowing the details of its implementation.

WebSphere Message Broker provides built-in message flow nodes that allow interoperability with WebSphere Process Server. The SCA nodes support inbound and outbound scenarios with WebSphere Process Server and WebSphere Enterprise Service Bus (WESB).

The nodes are configured by a Broker SCA definition. The Broker SCA Definition wizard creates a Broker SCA definition that includes objects that define the

binding, interface, and message format information to permit interoperability between WebSphere Message Broker and WebSphere Process Server.

SCAInput node

The SCAInput node listens for SCA inbound requests (from WebSphere Process Server to WebSphere Message Broker).

SCAReply node

The SCAReply node is used to send a message from the broker to the originating client in response to a message that a SCAInput node receives.

SCARequest node

The SCARequest node is used to send a request to WebSphere Process Server. The node is configured using a Broker SCA Definition (.outsca) file. Depending on the contents of the.outsca file, requests are either:

- ▶ Two-way, synchronous: The node sends the request and then blocks until it receives a response or the time out period is exceeded.
- ▶ One-way: The node sends a request only.

SCAAsyncRequest node

The SCAAasyncRequest node is used to send an SCA outbound request to a service component that runs in WebSphere Process Server.

SCAAsyncResponse node

The SCAAasyncResponse node receives the response from a business process that is running in WebSphere Process Server and to which the previous asynchronous request was made. The SCAAasyncResponse node can be in the same message flow or in a separate message flow.

Figure 4-13 shows the Toolkit icon representation for each of the SCA nodes.

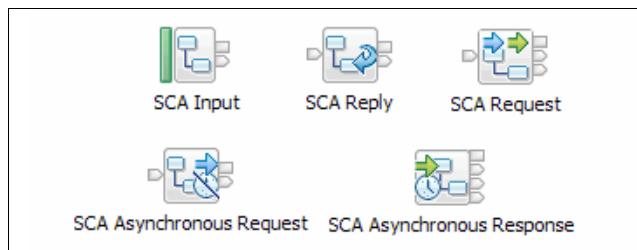


Figure 4-13 SCA nodes

For additional information about configuring and using each of the SCA nodes, go to the WebSphere Message Broker Information Center at:

<http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp>

In the WebSphere Message Broker Information Center, refer to these topics:

- ▶ SCALnput node (ac68510_)
- ▶ SCAResponse node (ac68520_)
- ▶ SCAResponse (bc34002_)
- ▶ SCAAsyncRequest (ac68530_)
- ▶ SCAAsyncResponse (ac68540_)

SCA nodes: We discuss the comprehensive message flow scenarios that make use of the SCA nodes in detail in Chapter 5, “Using SCA nodes for simplified integration with WebSphere Process Server” on page 73.



Part 2

Scenarios

In this part of the book, we use scenarios to illustrate the ESB capabilities of WebSphere Message Broker. To show how to integrate WebSphere Message Broker with a variety of Enterprise applications. These applications include WebSphere Process Server, EIS systems including SAP and Siebel, WebSphere Business Monitor, and WebSphere Service Registry and Repository. This part includes the following chapters:

- ▶ Chapter 5, “Using SCA nodes for simplified integration with WebSphere Process Server” on page 73
- ▶ Chapter 6, “Enterprise Resource Planning integration with WebSphere Message Broker” on page 143
- ▶ Chapter 7, “Service enable TCP/IP-based applications with WebSphere Message Broker” on page 243
- ▶ Chapter 8, “Scenario: File processing” on page 279
- ▶ Chapter 9, “Scenario: Publish/subscribe using WebSphere Message Broker V7.0” on page 343
- ▶ Chapter 10, “Scenario: Monitoring and WebSphere Business Monitor support” on page 427

- ▶ Chapter 11, “WebSphere Service Registry and Repository as governance” on page 507
- ▶ Chapter 12, “Using WebSphere ILOG JRules together with WebSphere Message Broker” on page 521



Using SCA nodes for simplified integration with WebSphere Process Server

In this chapter, we describe one of the key features in WebSphere Message Broker V7, SCA Nodes, which helps to enable integration with another IBM BPM product called WebSphere Process Server seamlessly. We start this chapter with a basic introduction for those readers who are new to WebSphere Process Server and its Services Component Architecture (SCA) and WebSphere Process Server' build-time Eclipse tool, which is called WebSphere Integration Developer (WID). Also, we demonstrate how to set up and deploy this integration using the SCA Nodes very quickly between two powerful IBM solutions: WebSphere Message Broker and WebSphere Process Server.

In this chapter, we provide:

- ▶ “Introduction to WebSphere Message Broker V7” on page 74
- ▶ “WebSphere Integration Developer overview” on page 80
- ▶ “SCA Nodes overview” on page 82
- ▶ “Scenario overview” on page 87

5.1 Introduction to WebSphere Message Broker V7

WebSphere Message Broker V7 has five built-in SCA nodes that are major integration enhancements between IBM ESB and BPM solutions. We start this chapter with an overview of the IBM BPM run-time product, WebSphere Process Server, with its core architecture component, which is called Services Component Architecture. We also provide an overview for the BPM build-time product, WebSphere Integration Developer (WID). Before WebSphere Message Broker V7 was available you used native MQ binding and Web Services binding to integrate these two solutions together. There was more work and less capabilities. With the WebSphere Message Broker V7, built-in SCA nodes solve these integration challenges. In this chapter, we also compare the work with and without SCA node usages.

In the last section of this chapter, you can explore all of the uses of the SCA nodes using three lab scenarios.

5.1.1 Overview of WebSphere Process Server

WebSphere Process Server is the IBM answer for the Business Process Management (BPM) run-time environment on SOA. It provides the capabilities on which to execute the functions that are necessary to link together services in a meaningful way. It is based on J2EE and uses WebSphere Application Server (WAS) as the run-time engine.

WebSphere Process Server includes three layers, as shown in Figure 5-1 on page 75.

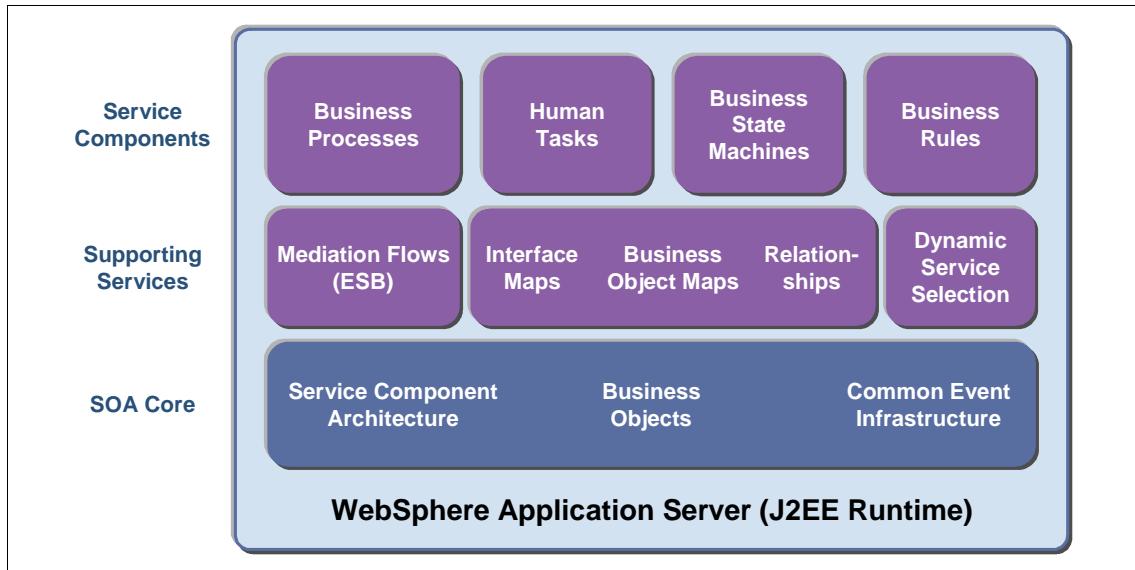


Figure 5-1 Overview of the components in WebSphere Process Server

The components of the WebSphere Process Server are:

- ▶ The SOA Core layer consists of the following components:
 - The Service Component Architecture presents all elements of business transactions, including access to Web services, Enterprise Information System (EIS) service assets, business rules, workflows, databases, and so on, as service components. The SCA separates business logic from implementation so that you can focus on assembling an integrated application without knowing the implementation details. Service components can be assembled graphically in IBM WebSphere Integration Developer (build-time Eclipse Tool), and the implementation can be added later.
 - Business objects (BO) define the data that flows between SCA components. Business Objects provide an abstraction for data access and are based on a data access technology called Service Data Objects (SDO) that provide a universal means of describing disparate data. Business objects provide rich features to map, manage, and transform data to underlying IT and are described through a standards-based XML schema.
 - With the Common Event Infrastructure (CEI), service components can emit events that can be captured by business monitors, such as WebSphere Business Monitor for real-time monitoring of business processes.

- ▶ *Supporting Services* are components that are needed in any integration solution, including data transformation and synchronization services. This layer contains the following components:
 - There is an ESB component that provides mediation flows that can operate on messages to perform XML-based data transformation, protocol transformation between various transports, custom Java operations, and routing; however, WebSphere Message Broker provides more comprehensive capabilities than the universal ESB, which is beyond J2EE, and WebSphere Message Broker can be used for the enterprise level to fulfill the business needs.
 - Using *interface maps*, you can invoke components by translating calls to other components. It is possible for interfaces of existing components to match semantically but not syntactically and is especially true for components that already exist and services that need to be accessed.
 - With *business object maps*, you translate one type of business object into another type of business object. You can use these maps in a variety of ways, for example, in an interface map to convert one type of parameter data into another.
 - Use *relationships* to establish relationship instances between object representations of the same logical entity in disparate back-end systems. You might want to access the same logical entity within business integration scenarios, for example, the same customer's address might need to be updated in various back-end systems, such as an enterprise resource planning (ERP) system and a customer relationship management (CRM) system. These relationships can be established and managed automatically using the relationships service component. These relationships are typically accessed from a business object map when translating one business object format into another.
 - Use *selectors* for dynamic selection and invocation of separate services, which all share the same interface. WebSphere Process Server offers a Web-based interface to enable dynamic updates to the selection criteria and target services, which means that a module that was deployed at a later time can still be called by this selector component, enabling dynamic changes to the integration solution.
- ▶ WebSphere Process Server provides the following service components:
 - A *business process* component implements a Business Process Execution Language (BPEL)-compliant process. You can develop and deploy business processes that support long and short-running business processes and a compensation model within a scalable infrastructure. You can create BPEL models in WebSphere Integration Developer or import from a business model that you created in WebSphere Business Modeler or any other modeling tool that supports the BPEL standard.

- *Human tasks* are stand-alone components that you can use to assign work to employees. Additionally, the human task manager supports the ad hoc creation and tracking of tasks. WebSphere Process Server also supports multi-level escalation for human tasks, including e-mail notification and priority aging.
- *Business state machines* provide another way of modeling a business process. Some processes are easily described as a sequential flow of activities, and they can be modeled as business processes. However certain processes are driven by events rather than a sequence, and in this case, the business state machine is a better fit for modeling the process. One example is an ordering process where you can modify or cancel the order at any time during the order process until the order is fulfilled.
- *Business rules* are a means of implementing and enforcing business policy through externalization of business functions. As a result, dynamic changes of a business process are enabled for a more responsive business environment.

5.1.2 Services Component Architecture overview

SCA is an architecture model for creating service-oriented applications. In the SCA, each service is seen as a component. Each of these components has nothing exposed but its interface and references. It can be seen as the building block, as shown Figure 5-2.

The core concepts are:

- ▶ Services are called Components
- ▶ Each Component has an Interface
- ▶ A caller of a Component has a Reference to that Component

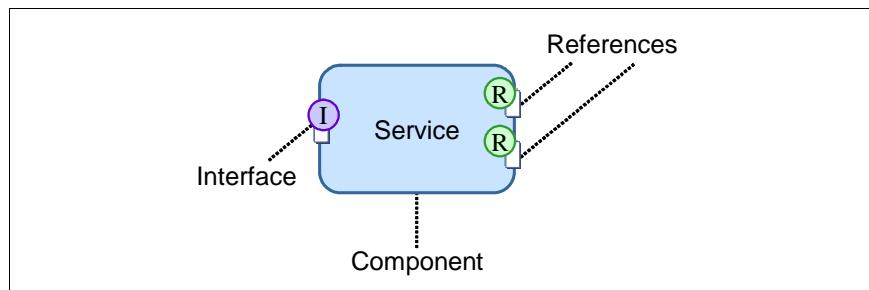


Figure 5-2 SCA service component

The following terms describe the SCA service component and the WebSphere Process Server implementation:

- ▶ **SCA interface**

By definition, an interface is the place at which independent and often unrelated systems meet and communicate with each other. An interface can be defined by any programming or interface definition language. WebSphere Process Server currently supports a Java interface definition and an XML definition (Web Services Description Language (WSDL) port type). The arguments that are described in an XML schema are exposed to programmers as SDOs. The WebSphere Integration Developer tooling primarily generates interfaces using the WSDL representation.

- ▶ **SCA references**

An SCA reference specifies other SCA services that a component uses. These can be soft links that do not have to specify which specific component will be used.

The users can call or expose services using imports and exports with the appropriate bindings that specify how data is being transferred. Imports and exports allow modules to publish their services and to use services from other sources. Figure 5-3 illustrate SCA Import and SCA Export.



Figure 5-3 SCA Import and Export

Import: An *import* allows you to use functions that are not a part of the module that users are assembling. Use an import to invoke an external function.

Export: An *export* is a published interface from a component that offers its service to the outside world. The export allows an external program to invoke services that the component provides.

Now, this service component can be expanded to a high-level view of an SCA implementation. An SCA component is configured by a configuration file, which is defined in an XML-based format called Service Component Definition Language (SCDL). The configuration, expressed in SCDL, defines how the component interacts with other components. The SCA architecture is the approach that WebSphere Process Server and several IBM and other vendor products use to link all of the services together.

The SCA implementation specifies the implementation type of the component's interface. Developers can implement business services in their language of

choice, for example, Java, BPEL, or state machine. Current implementation types include business process, human task, interface map, selector, business rules, business state machine, and Java, as shown in Figure 5-4.

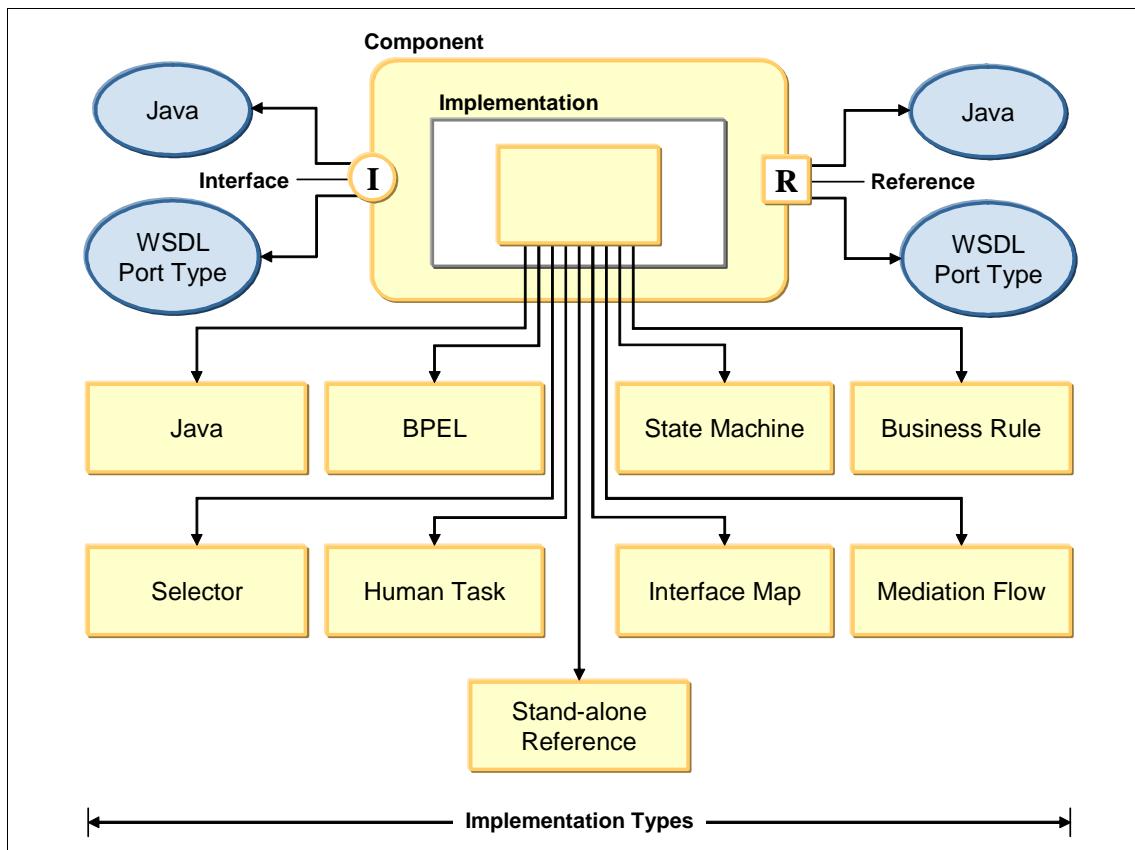


Figure 5-4 High-level view of an SCA service component

The benefits of using SCA are:

- ▶ Encapsulate components for reuse:
 - Service Components are wired together to form deployable solutions
 - Business Objects are the data flowing between Service Components
- ▶ All components (for example, services, rules, human interactions) are represented consistently and invoked identically and encapsulation and reuse reduce development cost. It helps to increase productivity and reduced cost.

5.1.3 WebSphere Integration Developer overview

WebSphere Integration Developer (WID) is the development (or build-time) and test environment for WebSphere Process Server. WebSphere Integration Developer is based on the Eclipse framework, which provides a consistent user interface that is likely to be familiar to users of development offerings from IBM and other vendors.

The primary perspective for working with WID is the Business Integration perspective. When you create a module, an assembly diagram for the module is created. As you add components to the module, the assembly diagram is populated with the added components. The Business Integration view is the most commonly used window in the WID tooling. It contains a list of all of the various types of artifacts with which you work. Inside the Business Integration view there is a folder per module, shown in Figure 5-5. Each module folder contains all of the entries of interest to the project.

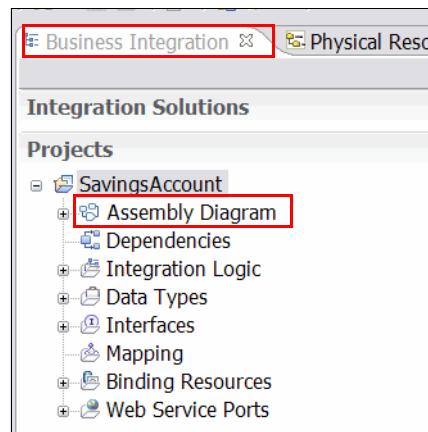


Figure 5-5 Assembly Diagram under Business Integration perspective

Figure 5-6 on page 81 shows a full window for this Assembly Diagram. From the assembly diagram, you can define the interfaces to each component, generate binding information for access to the module, and the implementation for service components.

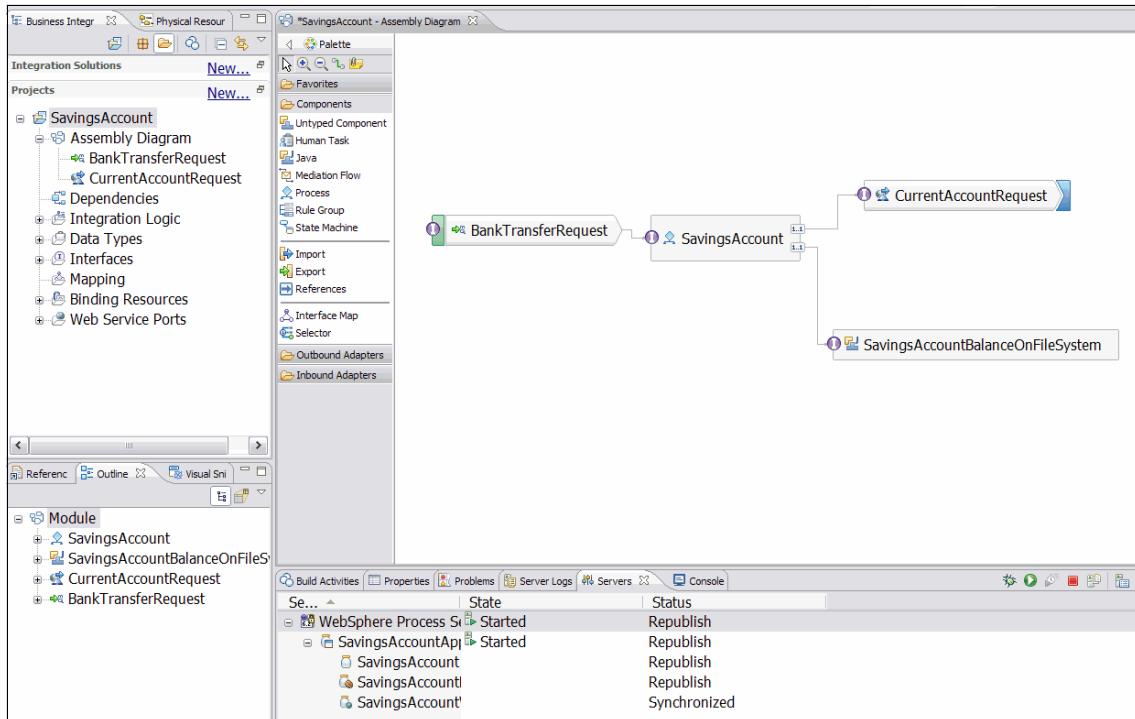


Figure 5-6 Assembly Diagram for service component setup

Figure 5-7 shows the diagram for the Interface Binding. Although this is a simplification of the WID capabilities, the purpose is to provide a background for the scenario that you follow to use SCA nodes in WebSphere Message Broker to communicate with a business process hosted on WebSphere Process Server.

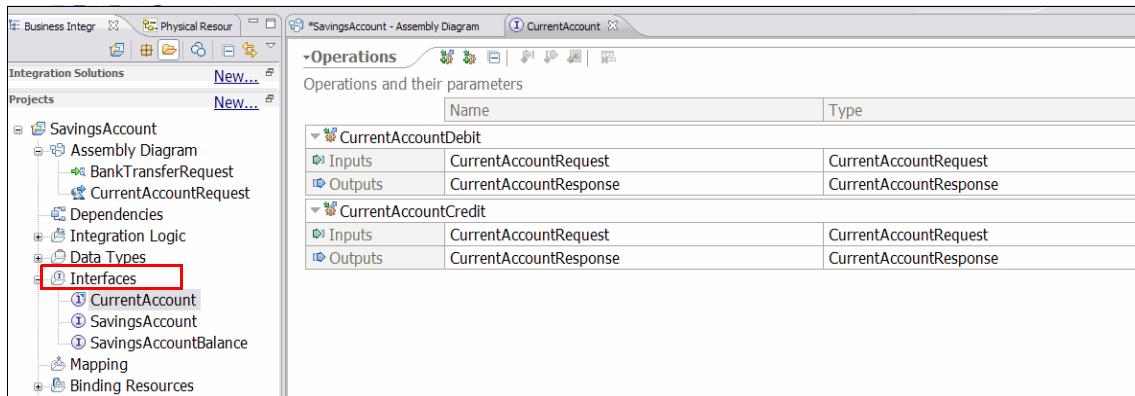


Figure 5-7 Full window of Interface Binding under Business Integration perspective

5.2 SCA Nodes and integration advantages in WebSphere Message Broker V7

In this section, we discuss the known integration advantages in WebSphere Message Broker.

5.2.1 SCA Nodes overview

WebSphere Message Broker V7 introduces five new SCA nodes that allow WebSphere Message Broker to interoperate with WebSphere Process Server more easily and seamlessly. These nodes are configured using SCDL information and allow communication with WebSphere Process Server using Web Services or MQ bindings. The importer, generator, and exporter SCA wizards are all new features in WebSphere Message Broker V7.

The SCA nodes are configured by a Message Broker SCA Definition, which is a new Message Broker artifact introduced in WebSphere Message Broker V7. The Broker SCA Definition contains the:

- ▶ SCDL: Contains binding and property information
- ▶ WSDL: Defines the interface that is being used
- ▶ XSD: Defines the message model

There are two types of Broker SCA Definition:

- ▶ Inbound: An inbound Broker SCA Definition has a suffix of .insca.

The inbound Broker SCA Definition is used to configure SCALocalInput and SCALocalReply nodes, as shown in Figure 5-8. An inbound Broker SCA Definition is derived from a SCA Import component.

- ▶ Outbound: An outbound Broker SCA Definition has a suffix of .outsca.

The outbound Broker SCA Definition is used to configure SCAResponse, SCAAsyncRequest, and SCAAsyncResponse nodes. An outbound Broker SCA Definition is derived from a SCA Export component.

SCA Input and SCA Reply Nodes

In this section, we explain the function of each SCA node.

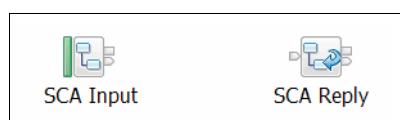


Figure 5-8 SCA Input and SCA Reply Nodes

The SCAInput node listens for SCA inbound message requests from WebSphere Process Server. A WebSphere Process Server SCA Import component can use Message Broker as an SCA endpoint. The message received by the broker might be an SOAP/HTTP message or an MQ message on a queue, depending upon the binding used by the node.

The SCAResponse node sends a reply back to the client that originated the SCA request, for example, it allows a response message to be sent from the Message Broker back to the originating WebSphere Process Server SCA client in response to a prior message received by an SCA Input node.

The SCAInput and SCAResponse nodes are analogous to the SOAPIInput and SOAPIResponse nodes, and are used in a message flow that provides an SCA endpoint.

SCA Request Node

Figure 5-9 illustrates a SCA Request node.

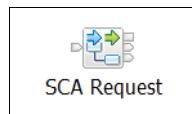


Figure 5-9 SCA Request node

The SCAResponse node sends a synchronous request to the WebSphere Process Server. The message sent by Message Broker can be a SOAP/HTTP message or an MQ message, depending upon the binding used by the node. Depending on the operations defined in the interface, requests are either:

- ▶ Two-way, synchronous: The node sends the request and then blocks for a specified amount of time until it receives a response or the time-out period is exceeded.
- ▶ One-way: The node sends a request only.

SCA Asynchronous Request and Response Nodes

The SCAAsyncRequest and SCAAsyncResponse nodes, shown in Figure 5-10 on page 84, are used to construct a pair of message flows that invoke a WebSphere Process Server service component asynchronously.

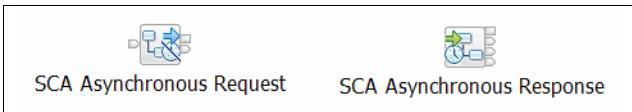


Figure 5-10 SCA Asynchronous Request and Response nodes

The SCAAsyncRequest node sends an SCA outbound request to a business process that WebSphere Process Server hosts. The SCAAsyncResponse node receives the response from a business process that is running in WebSphere Process Server and to which the previous asynchronous request was made. The SCAAsyncResponse node can be in the same message flow or in a separate message flow.

Calling a WebSphere Process Server service component asynchronously means that the SCAAsyncRequest node sends a request but does not wait for the associated response.

The nodes are used as a pair and correlate responses and requests. The message sent by the broker might be a SOAP/HTTP or an MQ message, depending upon the binding used by the node.

5.2.2 The integration advantages of using SCA nodes of WebSphere Message Broker V7

In this section, we compare the work effort and benefit by using SCA nodes in V7 verses the native MQ and Web Services bindings in V6.1. There are three types of comparisons: General Advantages, Web Services Binding, and MQ Binding.

General advantages of SCA Nodes

Table 5-1 shows the general advantages of SCA Nodes.

Table 5-1 General advantages

Message Broker V6.1	Message Broker V7
Requires prior knowledge about WebSphere Process Server and WebSphere Message Broker to configure message sets, message flows, and nodes.	Better out-of-the box user friendly experience. Just a few clicks to get WebSphere Message Broker working with WebSphere Process Server.
If there is a change in binding type, it changes nodes in the message flow, for example, moving between SOAP nodes to MQ nodes.	Always uses SCA nodes. Binding tab on node dynamically changes content according to the binding found in Broker SCA definition.

Message Broker V6.1	Message Broker V7
Needs separate SOAP Reply and MQ Output nodes for sending replies back to WebSphere Process Server.	SCA Reply is binding agnostic. If you have multiple SCA Inputs with separate bindings, you can get a single SCA reply.
	Has equivalent performance as V6.1.
	It can migrate from Web Services or MQ applications to SCA applications by generating a Broker SCA definition from the existing message set and using the .outsca/.insca to create SCA message flows.

Advantages of SCA Nodes for Web Services binding

Table 5-2 describes the advantages of SCA Nodes for Web Services binding.

Table 5-2 Web Services Binding

Message Broker V6.1	Message Broker V7
Exports specific WSDL and XSDS from WID and imports into WebSphere Message Broker (WID → WebSphere Message Broker). If there are multiple WSDLs, it can cause potential user error if the wrong WSDL is chosen.	Exports Project Interchange and imports into WebSphere Message Broker. Creates message set and nodes that are already configured.
Creates SCA Import or SCA Export manually and ties up with Web Service URL (WebSphere Message Broker → WID).	Exports from WebSphere Message Broker. SCDL is created and can be imported into WID, which points to WSDL. Contains correct endpoint information.
Needs SOAP extracts if propagating SOAP body only.	Has options inside the node. Propagates SOAP body or whole SOAP message.
Has no dynamic terminals. You must perform routing after node based on content.	Has dynamic terminals that allow routing according to the operation found in message.
Always creates sync request node when selecting consumer option.	Chooses sync or async node when you drag-and-drop .outsca onto canvas.

Advantages of SCA Nodes for MQ binding

Table 5-3 on page 86 shows the advantages of SCA Nodes for MQ Binding.

Table 5-3 MQ binding

Message Broker V6.1	Message Broker V7
Has MQMD Header node needed to fill in Target Function Name and Correlation ID information (Outbound).	Fills in automatically MQMD information by SCA Request.
Needs to export XSD information and manually copy across Queue information (Queue Name, Queue Manager) from WID to WebSphere Message Broker.	Creates Message Definition directly from Project Interchange. Queue information is configured on the node directly from Broker SCA Definition.
Exports the schema and imports it into WID and then copies it across queue information from WebSphere Message Broker to WID.	Exports the SCDL and schema. Imports into WID, all information that is configured from SCDL and linked to XSD.
Needs to set correlation info through the compute node or MQ Header node (extra node needed in the flow) for inbound.	Sets the SCA Reply correlation ID according to what is defined in the SCDL.
Always defaults to BLOB.	Detects if the message is XML or non XML when you create a message definition file from the WID Project Interchange.
N/A	Set LocalEnvironment.Destination.SCA.Request.Operation to specify the request operation dynamically.
N/A	When using SCA Async nodes, it can set and get information that must be shared between the SCAAsyncRequest and the SCAAsyncResponse nodes using LocalEnvironment.Destination.SCA.Request.UserContextLocalEnvironment.SCA.Response.UserContext.
N/A	SCA Input creates a Reply Identifier that is accessible in the Local Environment. If you have a request between the Input and Reply nodes, you can still send a reply back to the originating client. So you can have the SCA Reply node in a separate flow if the Reply ID is stored.

5.2.3 Summary

In Message Broker V7, the new SCA nodes, wizards, and the Broker SCA Definition give you far more simplicity to interoperate between WebSphere Message Broker and WebSphere Process Server. In the next section, we give you a step-by-step guide of how to use the new features.

5.3 Environment

We developed the scenarios in the Windows® environment with the following software:

- ▶ WebSphere Message Broker v7.0
- ▶ WebSphere MQ v7.0.1
- ▶ WebSphere Process Server v6.2.0.2
- ▶ WebSphere Integration Developer v6.2.0.2

5.4 Scenarios: Developing scenarios using SCA Nodes to integrate WebSphere Message Broker and WebSphere Process Server

Two SCA Node sample scenarios (5.4.3, “SCA Nodes scenario part 1a: Asynchronous Request and Response” on page 99 and 5.4.5, “SCA Nodes scenario part 2: A Message Broker flow implements a service using SCA Input and SCA Reply” on page 127) are packaged with WebSphere Message Toolkit V7. These sample scenarios are purposely left unfinished so that you can finish them to learn how to use SCA Asynchronous Request, Asynchronous Response, and Input and Reply nodes. In this section, we guide you as you complete these tasks. You can use the samples as the foundation to use SCA Nodes to interoperate with WebSphere Process Server in your own environments.

In addition, you will learn how to build another scenario (5.4.4, “SCA Node scenario part 1b: SCA Request” on page 115) by changing the SCA Asynchronous Request and Asynchronous Response nodes to the SCA Request node. Some business processes might have synchronous processes that must finish the transaction within a required time period. For this case, use the SCA Request node to achieve this type of requirement.

5.4.1 Scenario overview

The SCA sample scenario consists of two parts. The first part (5.4.3, “SCA Nodes scenario part 1a: Asynchronous Request and Response” on page 99) of the scenario uses SCA Asynchronous Request and SCA Asynchronous Response nodes. It consists of two message flows that:

- ▶ Create a bank transfer form for you to specify the transfer type and the transfer amount.

- ▶ Create the transfer request to send to the savings account that is hosted by WebSphere Process Server.

As an alternative, you might need to implement a synchronous process. You can change two SCA nodes (Asynchronous Request and Asynchronous Response) to a SCA Request node, which is referred to in 5.4.4, “SCA Node scenario part 1b: SCA Request” on page 115. All of the artifacts remain the same, although, you must modify specific Java code in the SetupTransferRequest Java Compute Node and SetupTransferReply Java Compute Node for the synchronous process. We provide the new Java code changes for the Java Compute Nodes later in this chapter.

The second part of the scenario (5.4.5, “SCA Nodes scenario part 2: A Message Broker flow implements a service using SCA Input and SCA Reply” on page 127) uses SCA Input and SCA Reply nodes. It provides an extra message flow that:

- ▶ Creates the current account hosted by WebSphere Message Broker that is linked to the savings account.

Figure 5-11 shows the relationships between the message flows on WebSphere Message Broker and the business process on WebSphere Process Server.

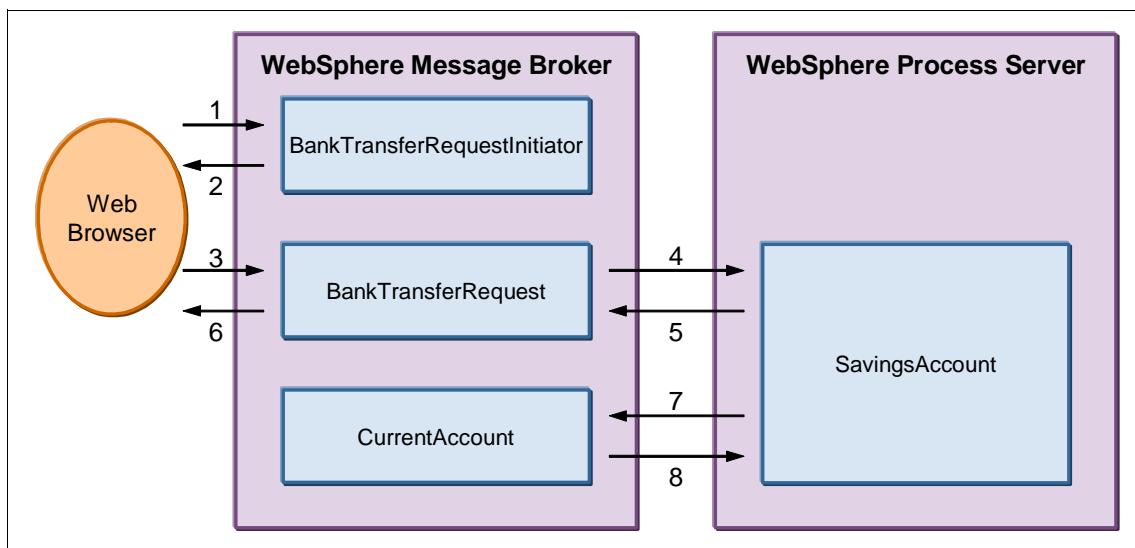


Figure 5-11 Full SCA Nodes scenario between WebSphere Message Broker and WebSphere Process Server

Part 1a of the scenario: SCA Asynchronous Request and Asynchronous Response

In this section, we discuss Part 1a of the scenario:

1. A Web browser starts the process by sending a request to the BankTransferRequestInitiator message flow on WebSphere Message Broker.
2. The BankTransferRequestInitiator message flow sends an HTML input form to you.
3. You submit the form data to a second message flow, BankTransferRequest, on WebSphere Message Broker.
4. The BankTransferRequest message flow primes the data that was submitted to send as a request to the SavingsAccount business process on WebSphere Process Server.
5. The SavingsAccount business process updates the savings account balance and returns the response to the BankTransferRequest message flow.
6. The BankTransferRequest message flow returns this response to the Web browser.

At this point, it only updates the Savings Account on WebSphere Process Server, and no Current account is updated yet on WebSphere Message Broker. For a complete round trip update, users must complete the second scenario to update the Current Account.

Part 1b of the scenario: SCA Request

For the first part of the scenario, you also have an alternative way to implement synchronous process by using SCA Request node. This synchronous process is named Part 1b. All of the steps from 1 to 6 are the same except there is a given time out period on step 5 (for example, 120 seconds as default). If the process does not complete in this time period, it will time-out. Some of the Java code in JavaCompute nodes also needs to be changed. We supply the new Java code on Example 5-1 on page 119 and Example 5-2 on page 123.

Part 2 of the scenario: SCA Input and SCA Reply

This scenario introduces a link from the current account to the savings account. The SavingsAccount business process primes data to send as a request to the CurrentAccount message flow on WebSphere Message Broker.

The CurrentAccount message flow updates the current account balance and returns the response to the SavingsAccount business process.

On the Message Broker side

In this section, we discuss the Message Broker side of the scenario.

The bank transfer request initiator message flow

This simple message flow, shown in Figure 5-12, comprises the following nodes:

- ▶ **HTTPInput**
- ▶ **JavaCompute (TransferRequestForm)**
- ▶ **HTTPReply**

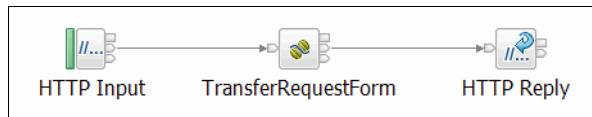


Figure 5-12 *BankTransferRequestInitiator message flow*

The broker receives a request in the form of a URI from a browser client using the HTTP transport.

The JavaCompute node:

1. Reads the appropriate balances from the file system.
2. Creates the message body that displays the account balances along with an HTML input form within all of the fields that are required to send a transfer request to WebSphere Process Server.
3. Changes the content type of this message to text or HTML, which is then sent as the response to the request.

Note: The Java source code for the TransformRequestForm JavaCompute node is in the SCA nodes sample in the WebSphere Message Broker Toolkit for your reference.

The bank transfer request message flow

The message flow in Figure 5-13 on page 91 receives a form that is submitted from a browser client. The form is posted using HTTP transport to the HTTP port on which the message flow is listening.

This message flow comprises the following nodes:

- ▶ **HTTPInput**
- ▶ **JavaCompute (SetupTransferRequest, SetupTransferFailedReply, SetupTransferReply)**
- ▶ **SCAAsyncRequest**

- ▶ SCAAsyncResponse
- ▶ HTTPReply

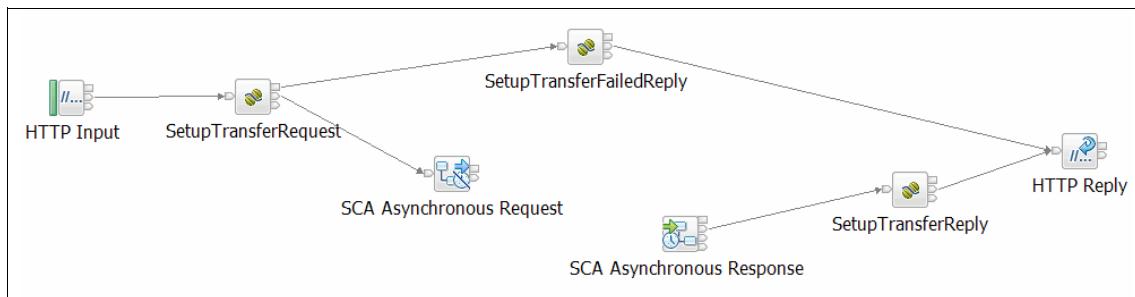


Figure 5-13 BankTransferRequest message flow

The message flow receives the property values as MIME message parts:

1. A JavaCompute node, SetupTransferRequest, processes these property values by creating a request message that specifies the amount to transfer, and it overrides the value in the Local Environment for the operation to invoke according to what the user specified in the HTTP form.

Note: The Java source code for SetupTransferRequest JavaCompute node can be found in the SCA nodes sample of Message Broker toolkit for your reference, or you can open up the JavaCompute node to review the Java code.

2. The request identifier is copied to your context so that it is not lost when the response message arrives at the SCA Asynchronous Response node. This identifier is required to reply back to the originating client.
3. The transfer can also be validated at this stage. If an invalid or blank amount is specified, an MbUserException is generated. If validation fails, the user exception is propagated through the Failure terminal. The exception message is included in a reply message to the originating client. You click **OK** to be routed back to the bank transfer form so that a correct amount can be specified.

Note: The Java source code for the SetupTransFailedReply JavaCompute node is in the SCA nodes sample of the Message Broker toolkit for user reference.

4. If validation is successful, the transfer request message is propagated to a SCAAsyncRequest node. This node sends the message to the business

process that is hosted by WebSphere Process Server. This business process contains the business logic that is related to the savings account processing.

5. After the request is processed, an MQ response message is sent back to WebSphere Message Broker where it is processed by the corresponding SCAAsyncResponse node. If the bank transfer request was successfully processed, the new savings balance is included in the reply message. If the bank transfer request was unsuccessful, the previous savings balance is used.
6. The request identifier is also retrieved from the Local Environment and copied to the Destination folder so that the reply message can be sent back to the originating client.

BankTransferSyncRequest message flow

Optional Part 1b lab uses synchronous process.

Note: Copy the BankTransferRequest message flow, and rename it to BankTransferSyncRequest.

The message flow receives a form that is submitted from a browser client. The form is posted through HTTP transport to the HTTP port on which the message flow is listening.

The message flow in Figure 5-14 comprises the following nodes:

- ▶ **HTTPInput**
- ▶ **JavaCompute** (SetupTransferRequest, SetupTransferFailed Reply, SetupTransferReply)
- ▶ **SCARequest**
- ▶ **HTTPReply**

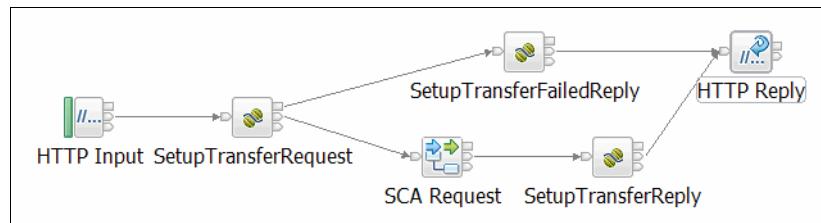


Figure 5-14 *BankTransferSyncRequest message flow*

The message flow process and artifacts are the same as the Part 1a scenario. However, for step 1 on page 91 and step 2 on page 91, comment out the areas in

the SetupTransferRequest Java Compute Node and SetupTransferReply Java Compute Node. Use SCA Request node and provide a time-out period.

The message flow receives the property values as MIME message parts:

1. A JavaCompute node, SetupTransferRequest, processes these property values by creating a request message that specifies the amount to transfer, and it overrides the Local Environment for the operation to invoke according to what the user specified in the HTTP form.

Note: The Java code in the SetupTransferRequest JavaCompute node for Part 1b is supplied in Example 5-1 on page 119.

2. The request identifier does not need to be copied to your contest when the SCA Request node is used because it is preserved in the message flow. This identifier is required to reply back to the originating client that submitted the HTTP form.
3. The transfer can also be validated at this stage. If an invalid or blank amount is specified, an MbUserException is generated. If validation fails, the user exception is propagated through the Failure terminal. The exception message is included in a reply message to the originating client. Click **OK** to be routed back to the bank transfer form so that a correct amount can be specified.

Note: The Java source code for the SetupTransferFailedReply JavaCompute node for Part 1b is supplied in Example 5-2 on page 123.

4. If validation is successful, the transfer request message is propagated to a SCA Request node. This node sends the message to the business process that WebSphere Process Server hosts. The business process contains the business logic related to the savings account processing.
5. The SCA Request node waits for a specified time period for the response message to be sent back to WebSphere Message Broker from the business process on WebSphere Process Server. If the bank transfer request successfully processed, the new savings balance is included in the reply message. If the bank transfer request was unsuccessful, the previous savings balance is used. If a response message is not received by the SCA Request node within the time-out period, the input message and an exception list is propagated to the failure terminal. You can optionally extend the flow to wire the failure terminal to another Compute node so that a failure message can be constructed and then sent back to the originating client using the HTTP reply node.
6. The Java code in the SetupTransferReply JavaCompute node does not need to access the request identifier from the user context in the Local Environment

in the same way as when the flow contains SCA Asynchronous nodes.

Note: The Java code in the SetupTransferReply JavaCompute for Part 1b is supplied in Example 5-2 on page 123.

The Current Account Request Message Flow

In both Part 1 and Part 2 of the scenarios, the process is initiated by the bank transfer request initiator and the bank transfer request message flows. Part 2, shown in Figure 5-15, adds the CurrentAccountRequest message flow that is called from a business process within the WebSphere Process Server.

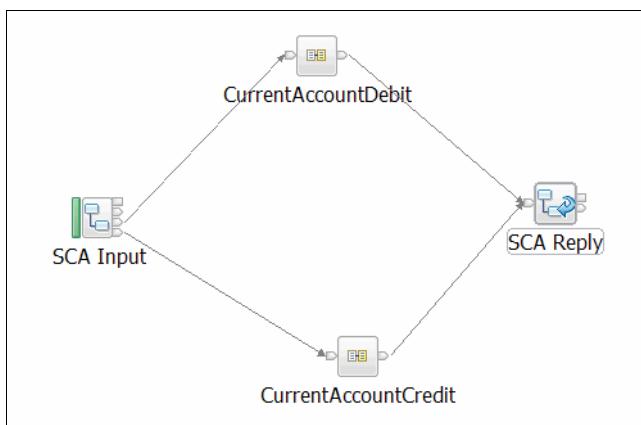


Figure 5-15 CurrentAccountRequest message flow

The flow is started by the SCAInput node and ended by the SCAResponse node. The routing from the SCAInput node depends on the operation that is invoked. In each case, the message is propagated automatically from a terminal that is dedicated to the operation.

Two subflows for credit and debit

The CurrentAccountDebit subflow, shown in Figure 5-16, contains the logic to perform the required debit operation.

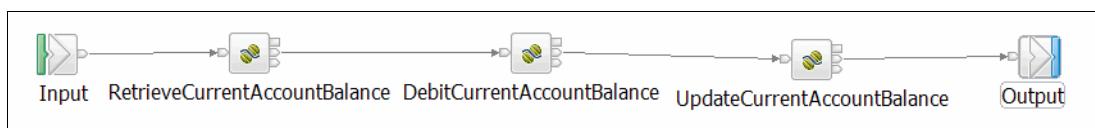


Figure 5-16 Current Account Debit Subflow

The CurrentAccountCredit subflow, Figure 5-17, contains the logic to perform the required credit operation.

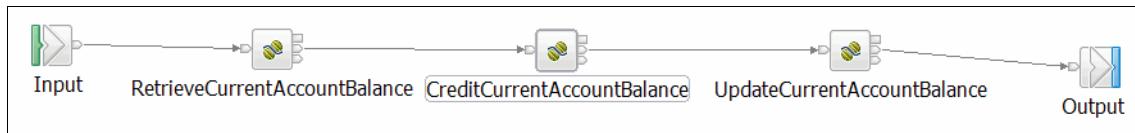


Figure 5-17 Current Account Credit subflow

On the WebSphere Process Server side

In this section, we discuss the processes on the WebSphere Process Server side.

The savings account

The savings account is hosted on WebSphere Process Server. It contains:

- ▶ A business process (SavingsAccount) that contains all of the business logic.
- ▶ An MQ export binding (BankTransferRequest) that exposes this business process to WebSphere Message Broker.
- ▶ A Java component (SavingsAccountBalanceOnFileSystem) that retrieves and updates the balance from a flat file.

The assembly diagram and the SavingsAccount process differ for the two parts of the scenarios.

Part one scenario

The components are connected to the assembly diagram shown in Figure 5-18.

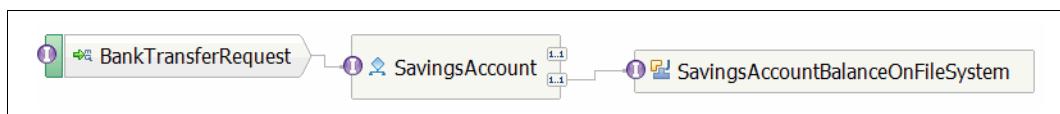


Figure 5-18 First part of Assembly Diagram

The implementation of the business process is split into two parts that correspond to the two operation requests:

- ▶ Transfer money to the savings account
- ▶ Transfer money to the current account

The implementation of the business process is split into two parts that correspond to the two operation requests, as shown in Figure 5-19:

- ▶ Transfer money to the savings account
- ▶ Transfer money to the current account

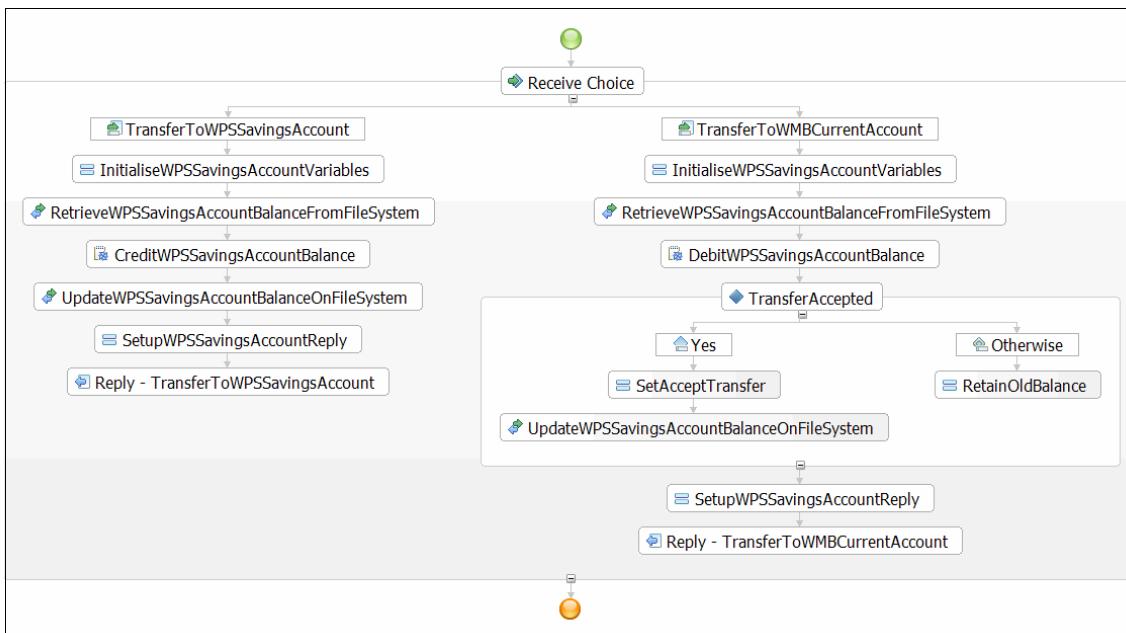


Figure 5-19 Transfer Money Business Process

The process for the Part 1 scenario is:

1. If the incoming message request is to transfer to the savings account, the business process variables are initialized from this message. The savings balance is read from a file by using a Java interface and credited by the requested amount. This new balance is then written back to the file by using the Java interface. A message response is created with the relevant values filled in from the business process variables and sent back to the caller, which is always the case in the First part of the Workshop.
2. If the incoming message request is to transfer to the current account, the business process variables are initialized from this message. The savings balance is read from a file by using a Java interface and debited by the requested amount. If the new balance is greater than or equal to zero, the new balance is written back to the file by using the Java interface. If the new balance is less than zero, the previous balance remains in the file. A message response is created with the relevant values filled in from the business process variables and sent back to the caller.

Part two scenario

The components for the Part 2 scenario are connected in the assembly diagram that we show in Figure 5-20.

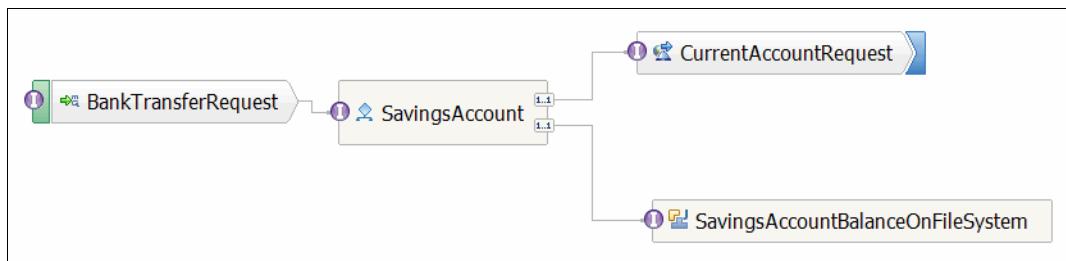


Figure 5-20 Assembly Diagram for the second scenario

The implementation of the business process, in this case, includes calls to Message Broker as a Service Component, as shown in Figure 5-21.

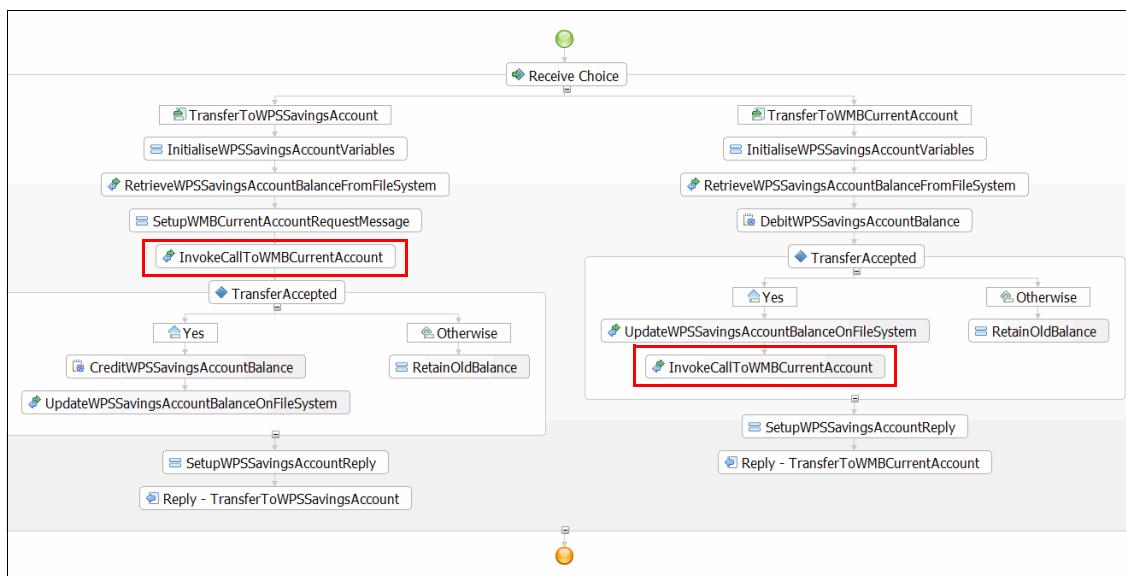


Figure 5-21 Business Process calls to Message Broker as a Service Component

The operations that are invoked by `InvokeCallToWMBCurrentAccount` differ in the two branches of the process. In the left branch (`TransferToWPSSavingsAccount`), the operation is `CurrentAccountDebit`, and in the right branch (`TransferToWMBCurrentAccount`), the operation is `CurrentAccountCredit`.

Note: In both Part one and Part two of these scenarios, the process is initiated by the bank transfer request initiator and the bank transfer request message flows from WebSphere Message Broker.

5.4.2 Setting up the environment

In this section, we provide a list of items that we performed or considered for the set up of our environment:

1. The SCA nodes sample scenario is located in the Sample Gallery under transport and connectivity.
2. Ensure that the local queue manager is called MB7QMGR, which exists with listener port, 2414. If not, create one.
3. Ensure that the following local queues exist (the sample creates them for you). If not, create them:
 - RECEIVE
 - SEND
4. Create a broker that associated with the local queue manager, MB7QMGR.
5. Deploy the SCAnodesSample.bar BAR file to the broker.
6. A corresponding savings account, SCA service application, is implemented as a business process and deployed on WebSphere Process Server. The project interchange file, WIDSavingsAccount.zip, in the SCAnodesSample project, must be exported from WebSphere Message Broker to the local file system and then imported into WebSphere Integration Developer on the same local machine.
7. Deploy the SavingsAccount application to WebSphere Process Server. This application provides the SCA framework for a two-way message request-response exchange between WebSphere Message Broker and WebSphere Process Server applications.
8. Alternatively, deploy the WIDSavingsAccount.ear enterprise archive file, in the SCAnodesSample project, directly to WebSphere Process Server. The enterprise archive file must be exported from WebSphere Message Broker to the local file system, and then use either the administrative console or a script to install the application. See the WebSphere Process Server information center for additional information.
9. Restart WebSphere Process Server to start the listener port. See the WebSphere Process Server information center for additional information.

10. Export the SavingsAccount.txt file from WebSphere Message Broker to a temporary folder on the file system. This file contains the initial savings account value of 100, which you can edit:
 - On Windows: this temporary folder is C:\tmp
 - On Linux®: this temporary folder is /tmp

5.4.3 SCA Nodes scenario part 1a: Asynchronous Request and Response

In this scenario, we demonstrate how you can use and deploy the SCAAsyncRequest and SCAAsyncReponse nodes to interoperate with WebSphere Process Server.

On the WebSphere Message Broker Toolkit

Perform the following steps:

1. Start and open the Message Broker toolkit in the quick launch toolbar, as shown in Figure 5-22.



Figure 5-22 Message Broker Toolkit icon

2. When the toolkit is open in the workspace, import the SCA Project into the workspace.
3. To simplify this process, you can import the SCA sample message flows and message set:
 - a. On the MB toolkit, click **Help → Samples and Tutorials → WebSphere Message Broker Toolkit - Message Broker**.
 - b. Scroll to Transports and Connectivity, look for the SCA Nodes sample, and click **More information**.
4. Select **Import and deploy the sample** in one step, or Import the sample, then deploy it separately.
5. Click **select All** to finish. Figure 5-23 on page 100 shows the result of the Import.

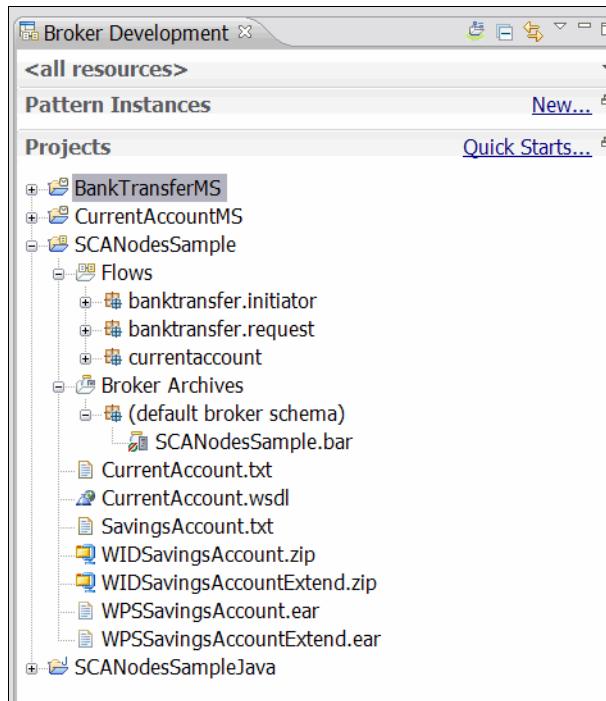


Figure 5-23 Import SCA Project result

On the WID toolkit

Perform the following steps:

1. Import the Project into WID that implements the SavingsAccount business process. Double-click the WID icon, shown in Figure 5-24, to start it.



Figure 5-24 WID Icon

2. Select the workspace **C:\...(your) directory...\wid\SCAwkspace**. Use Browse to get to the right location, if necessary.
3. Import the SavingsAccount project into the Workspace. Select **File → Import**, and then select **Project Interchange** and **Next**.
4. Navigate to **C:\...(your) Directory ..\SCA\resources\WIDSavingsAccount.zip**, and check **Select All** to finish.

5. Wait for the Workspace to complete building, as shown in Figure 5-25, and then confirm that the resources were imported successfully (it might take a little time for all of the resources to appear).

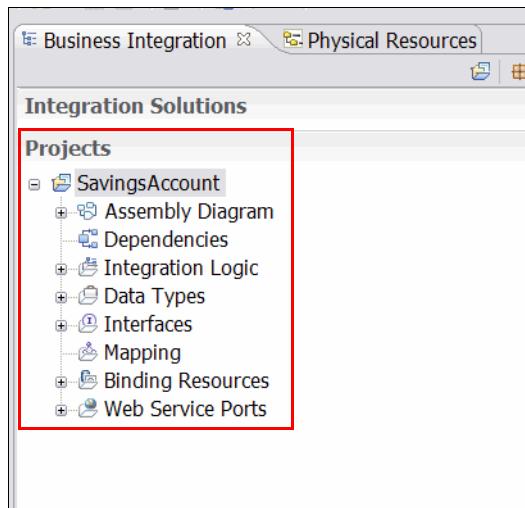


Figure 5-25 Savings Account display on WID

6. Click **Savings Account** → **Assembly Diagram** to display Figure 5-26.

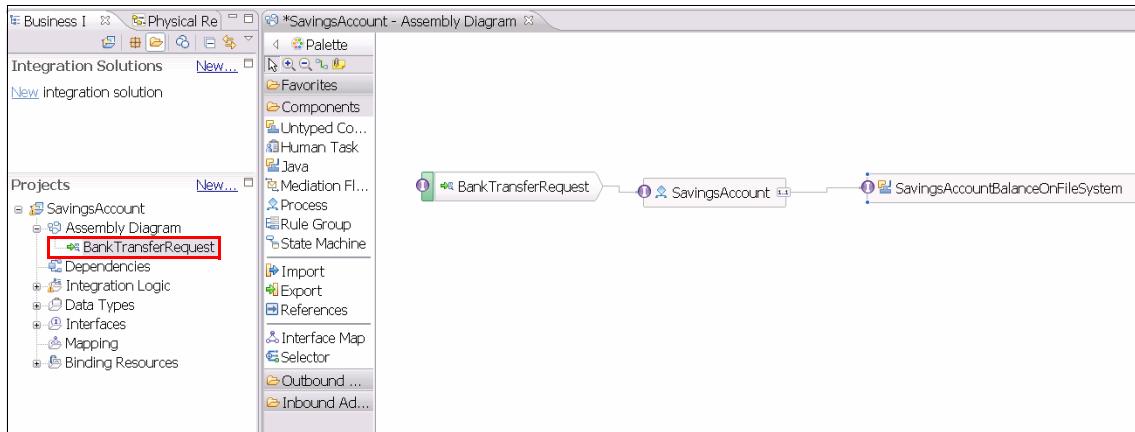


Figure 5-26 Bank Transfer Request Assembly Diagram

7. Click the Assembly Diagram end-point to verify MQ the binding configuration, which we show in Figure 5-27 on page 102, which is usually where the WID developer has to set up.

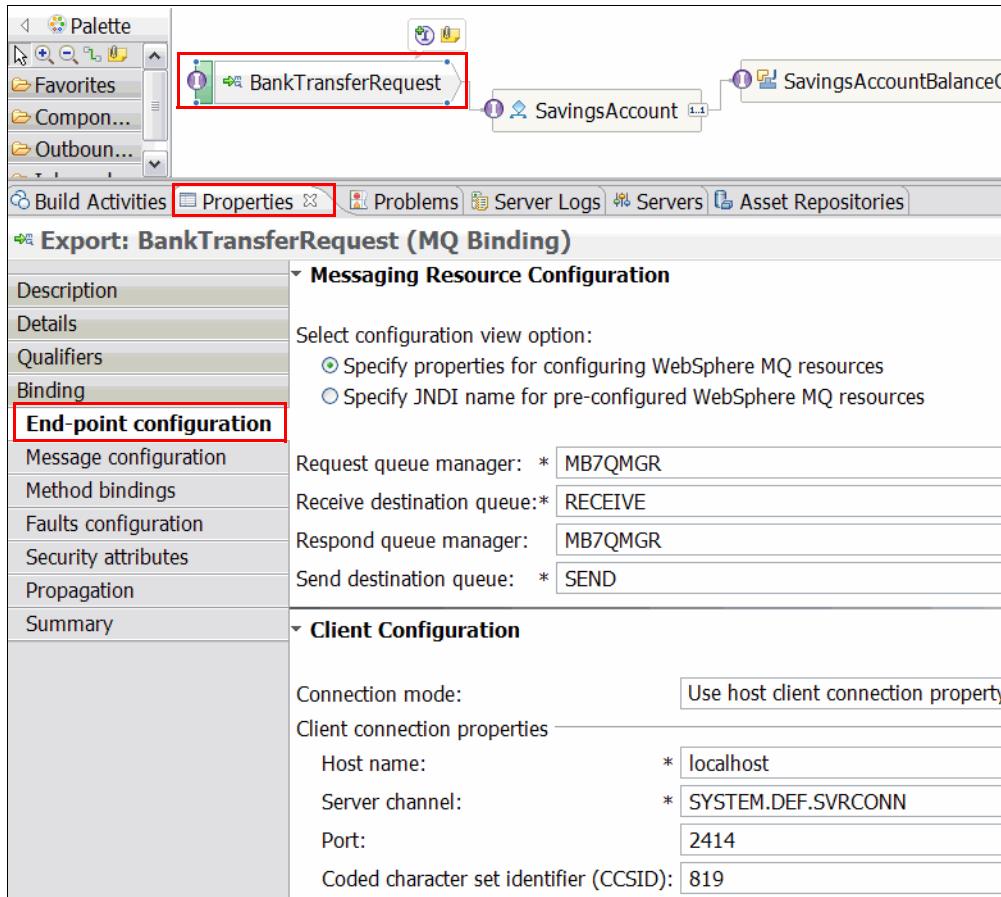


Figure 5-27 End-point MQ binding configuration

8. Having developed a Project within WID, the next logical step is to export that project that Message Broker needs to create the message definitions. In fact, the Project was exported into WID and has fulfilled this requirement. But within the user's own environment, this is the necessary step to implement.
9. Select **File → Export → Project Interchange**, and press **Next**. Check only SavingsAccount.
10. Browse to a directory C:\..\(your directory..\SCANodesSample\), and set it as the destination. Export the project as SavingsAccountSCAExport.zip to distinguish it from the project that you imported into the WID workspace. The WebSphere Process Server resource files are packaged as part of the SCA module that is exported in the form of a project Interchange file that contains a MQ export binding that is used to interface with the broker.

Switching back to the WebSphere Message Broker toolkit

To switch back to the WebSphere Message Broker toolkit:

1. Create a new Message Set called BanktransferMS, as shown in Figure 5-28, which contains the message definitions for the message that is exchanged with the business process that is running in WebSphere Process Server. Click **File → New → Message Set**.
2. When Figure 5-28 is displayed click **Next**. Select XML documents (general) on the message data, and leave all of the other boxes unchecked.

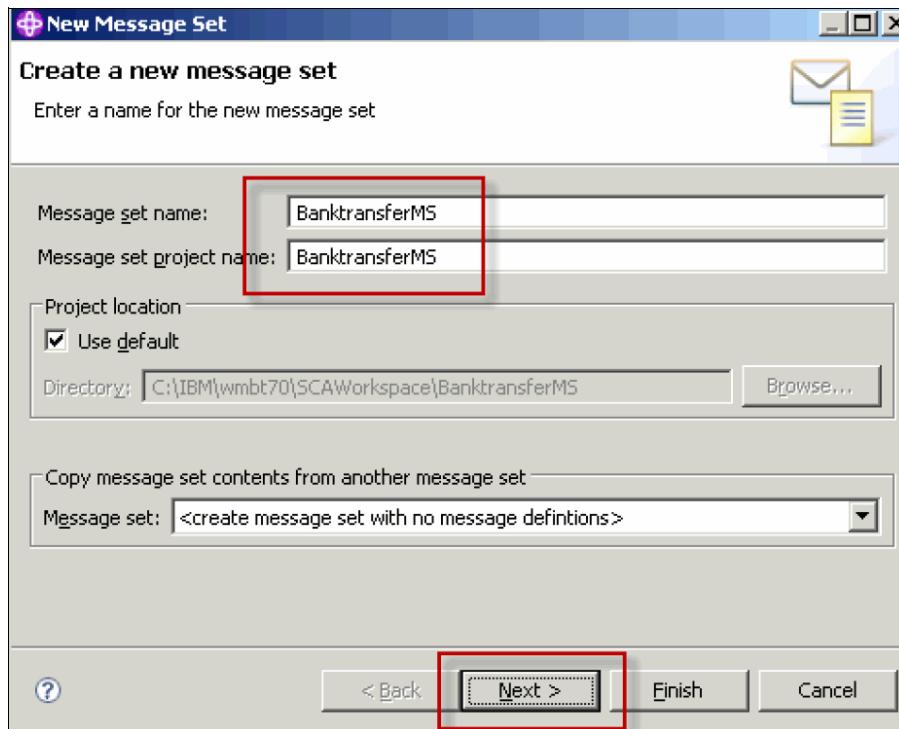


Figure 5-28 BanktransferMS message set

3. Click **Next** → **Finish**. On the tip window that is displayed, click **OK**.
4. Highlight the new message set, and right-click it. Select **New → Message Definition File From → SCA Import or Export**, as shown in Figure 5-29 on page 104.

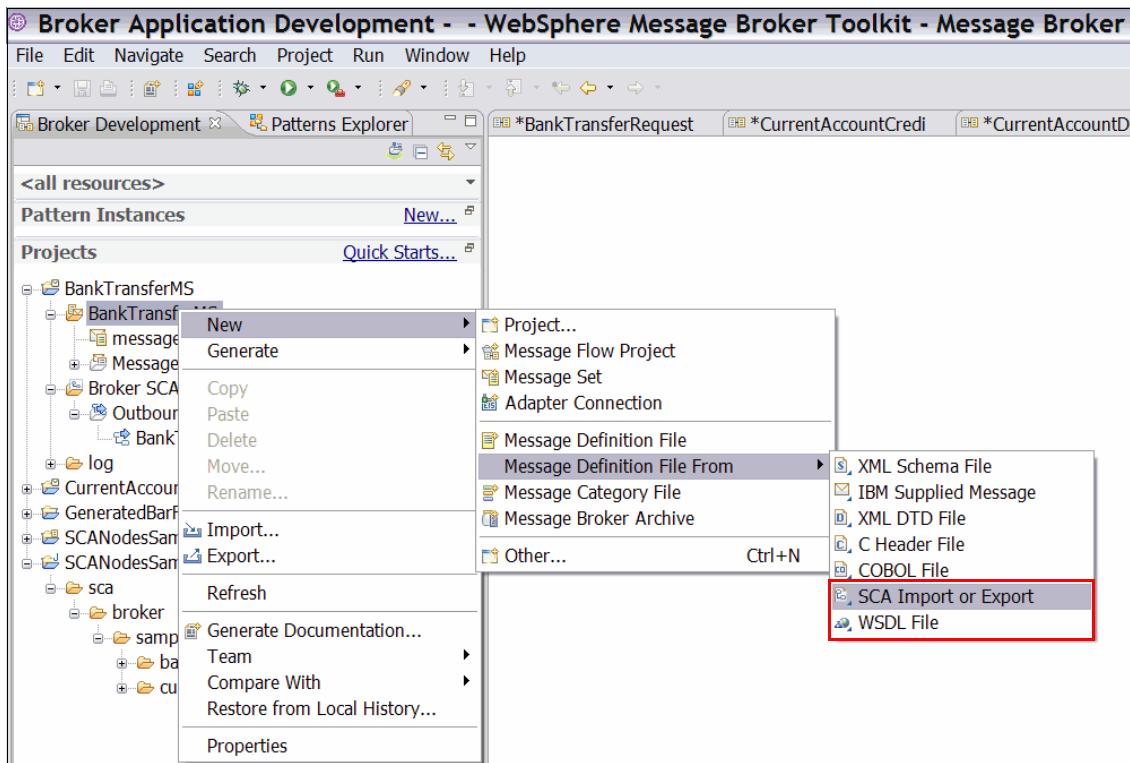


Figure 5-29 Export SCA Savings Account to new message definition file

5. Choose **Select archive from outside workspace**, and click **Browse**. Navigate to C:\...(your) directory ...\\SCA\\resources, select **SavingsAccountSCAExport.zip**, and click **Open**.
6. Finish the new message definition creation. When the new message definition is created, a call-out is displayed that invites you to drag-and-drop the imported Broker SCA definition onto the message flow editor.
7. Complete the message flow that was partially created by dragging the SCA definition onto the canvas. Locate and double-click the message flow **BankTransferRequest.msgflow** to open it in the message flow editor, as shown in Figure 5-30 on page 105.

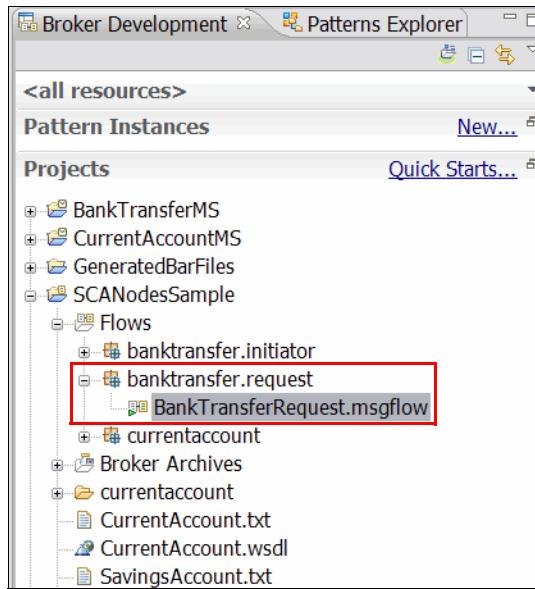


Figure 5-30 BankTransferRequest message flow

8. Drag-and-drop BankTransferRequest.outsca from BankTransferMS SCA Definitions onto the canvas, as shown in Figure 5-31.

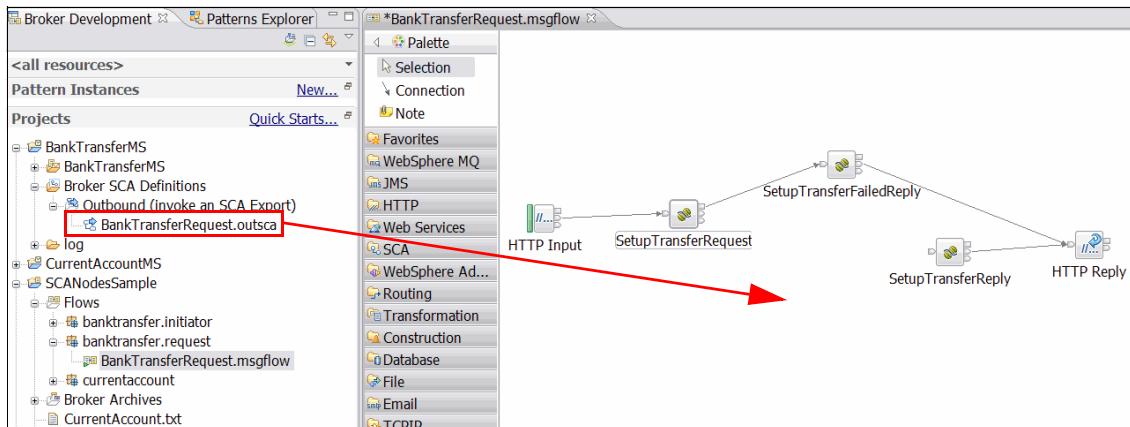


Figure 5-31 Drag BankTransferRequest.outsca to the canvas

9. A message is displayed that asks you to configure the service invocation on WebSphere Process Server. Select service **TransferToSavingsAccount**, clear the check box to specify asynchronous invocation, and press **OK**, as shown in Figure 5-32 on page 106.

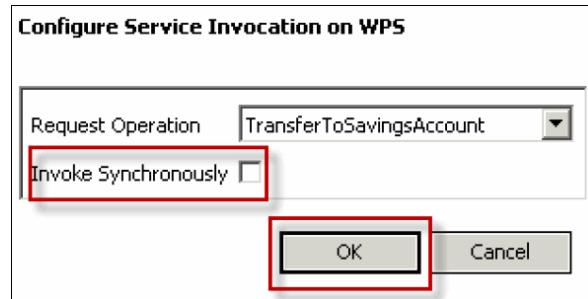


Figure 5-32 Configuring Service Invocation

10. Wire the Out terminal of SetupTransferRequest (JavaCompute) node to the In terminal of the SCA Asynchronous Request node, and the Out terminal of the SCA Asynchronous Reply node to the In terminal of the SetupTransferReply (JavaCompute) node, as shown in Figure 5-33. The message flow is complete.

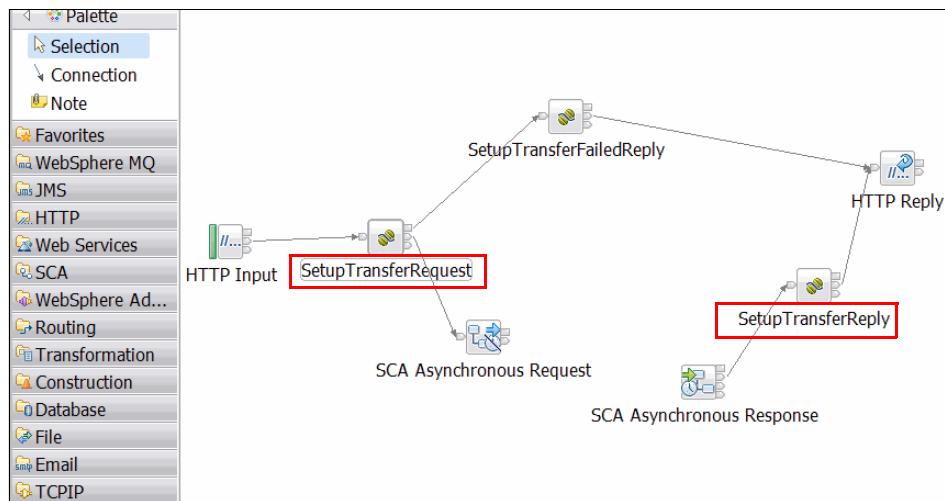


Figure 5-33 BankTransferRequest Message flow

11. Examine the property details of the service by right-clicking the SCAAsynchronousRequest node. In the Properties folder, select the **Basic** tab, and note that the Broker SCA definition is provided by the resource SCA/BankTransferRequest.outsca, as shown in Figure 5-34 on page 107. This is the resource that you exported from WID to create a message definition within Message Broker.

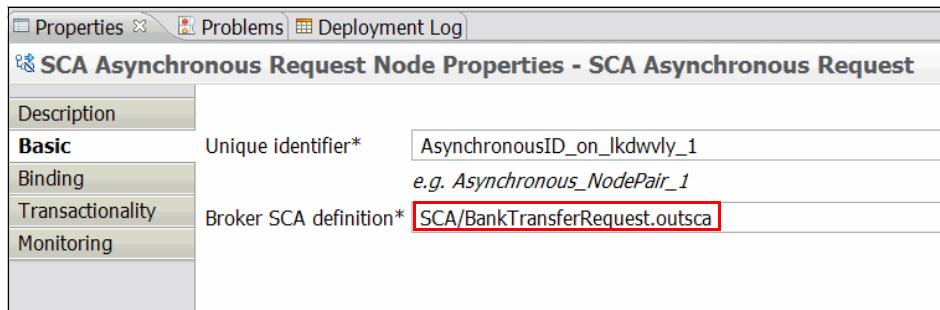


Figure 5-34 Broker SCA definition

12. In the Properties folder, select the **Binding** tab, as shown in Figure 5-35. Note that the binding type is MQ and invokes the TransferToSavingsAccount operation. The key point of this is that the SCA Asynchronous Request node constructs an MQ message that contains the service request and sends it to a local queue called RECEIVE that is owned by the queue manager MB7QMGR. The process that is running in WebSphere Process Server retrieves the message from the RECEIVE queue and performs the processing logic that is required. It subsequently constructs an MQ message that contains the response and puts it on the Reply-to queue, which is called SEND. The SCA Asynchronous Response node retrieves the message from this queue and propagates it onwards for further processing as required.

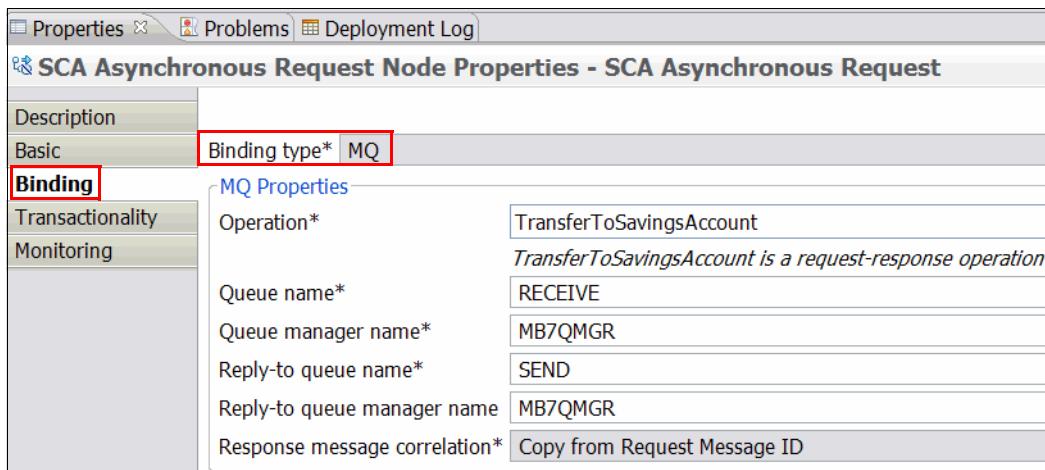


Figure 5-35 MQ Binding

13. Deploy the message flows and message set to the Message Broker runtime, as shown in Figure 5-36 on page 108. Because this is the partial existing product sample, this archive creation is already done for you. For other

environments, use the rest of these instructions for the archive creation. Locate the SCANodesSample folder in the Broker Development View. Right-click the SCANodesSample, and select **New → Message Broker Archive**. Give the new broker archive the name SCANodesSample, and click **Finish**.

14. Select both message flows that are shown at this time and the BanktransferMS message set. Click **Build broker archive**. When the Build completes, press Ctrl-S to save the new Broker Archive file. Click **OK** to respond to the confirmation message that the operation completed successfully. Press Ctrl-S to save the new archive file.

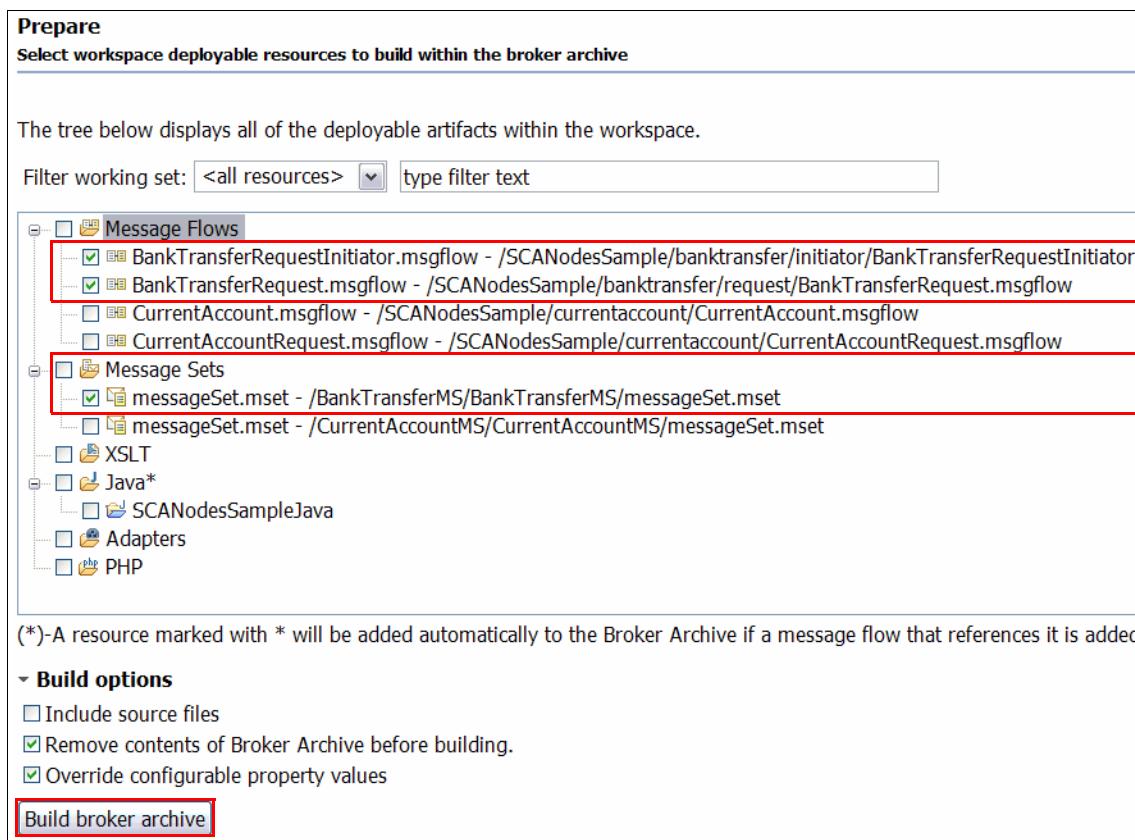


Figure 5-36 Select resources for deployment

15. Use the Default Execution Group for the Broker resources deployment, or create another new Execution Group. In this case, the user creates a new execution group called SCANodeSample using MQ Explorer, as shown in Figure 5-37 on page 109. In the MQ Explorer, make sure that the Message

Broker, MB7BROKER, is running and connected. If not, right-click the broker name, and select **Start**.

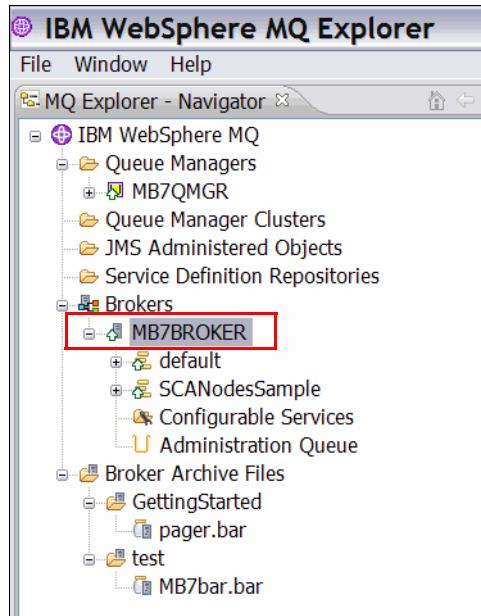


Figure 5-37 Create Execution Group called SCANodeSample

16. After the Execution Group is created successfully, drag-and-drop the Broker Archive File, SCANodesSample.bar, from the MB toolkit to the Execution Group in MQ Explorer, as shown in Figure 5-38 on page 110.

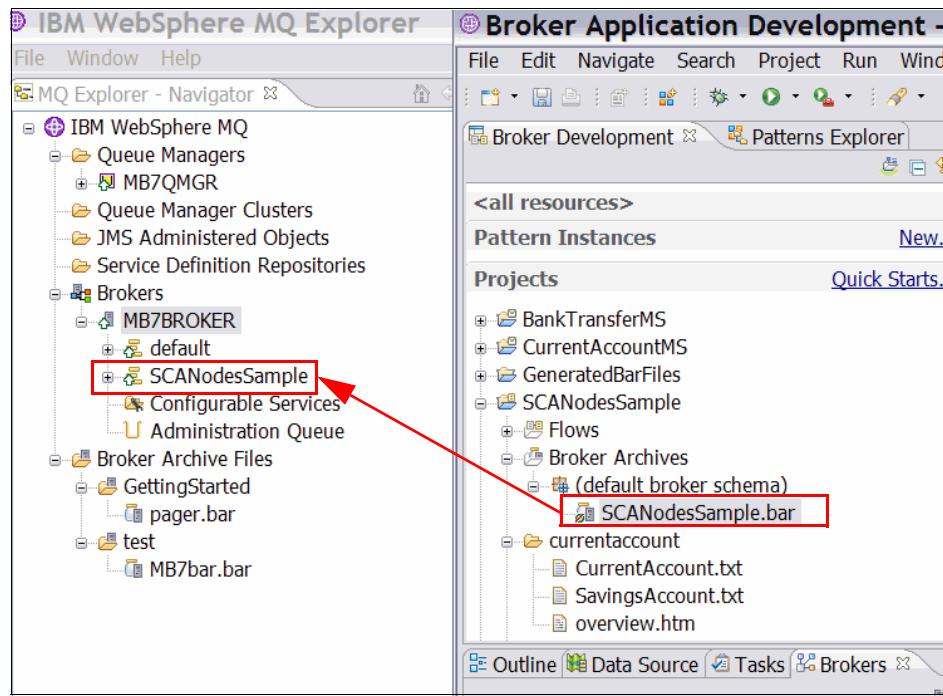


Figure 5-38 Drag SCANNodesSample bar file from development toolkit onto the EG on MQ Explorer

17. It is necessary to configure resources on queue manager, MB7QMGR correctly. We already completed this step for the user in this sample. But for your reference, the following MQSC script file can be run to create the resources.

MQSC Script file for creating the MQ resources (or create them manually using MQ Explorer):

```
DEFINE QL(RECEIVE) REPLACE
DEFINE QL(SEND) REPLACE
DEFINE LISTENER(TCP2414) TRPTYPE(TCP) PORT(2414 CONTROL(QMGR)
REPLACE
START LISTENER(TCP2414)
DEFINE QL(MB7QMGR.DEADQ) REPLACE
ALTER QMGR DEADQ(MB7QMGR.DEADQ)
```

Deploying the project to WebSphere Process Server

In this section, follow the instructions to deploy the project to WebSphere Process Server.

On the WID toolkit

Now we must make the Web services application available to WebSphere Process Server so that it can be invoked by Message Broker for testing. To deploy the project to WebSphere Process Server on the WID toolkit:

1. In WID, open the Servers view to start WebSphere Process Server.
2. Open the Console view to ensure that no fatal errors are logged, as shown in Figure 5-39. Confirm that WebSphere Process Server started successfully. A message is displayed in the console stating that the server is open for e-business.

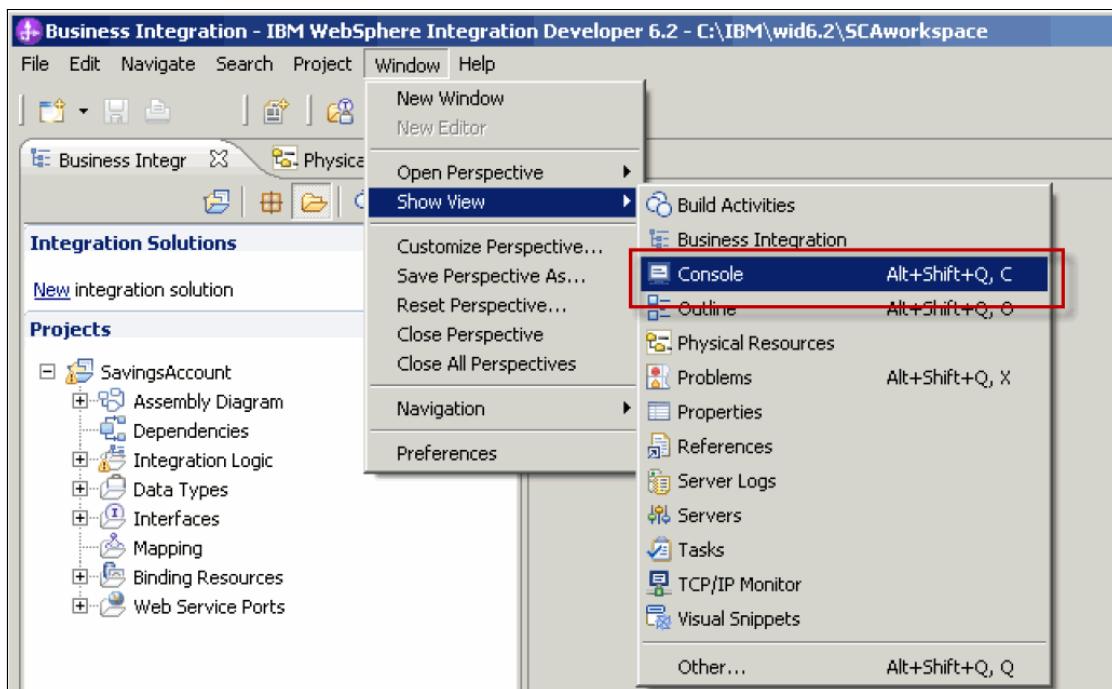


Figure 5-39 Display the Console view

3. Right-click **WebSphere Process Server**, and select **Add and Remove Projects**.
4. Highlight **SavingsAccountApp**, and click Add. Click **Finish** to complete the deployment of the Project to WebSphere Process Server, as shown in Figure 5-40 on page 112.

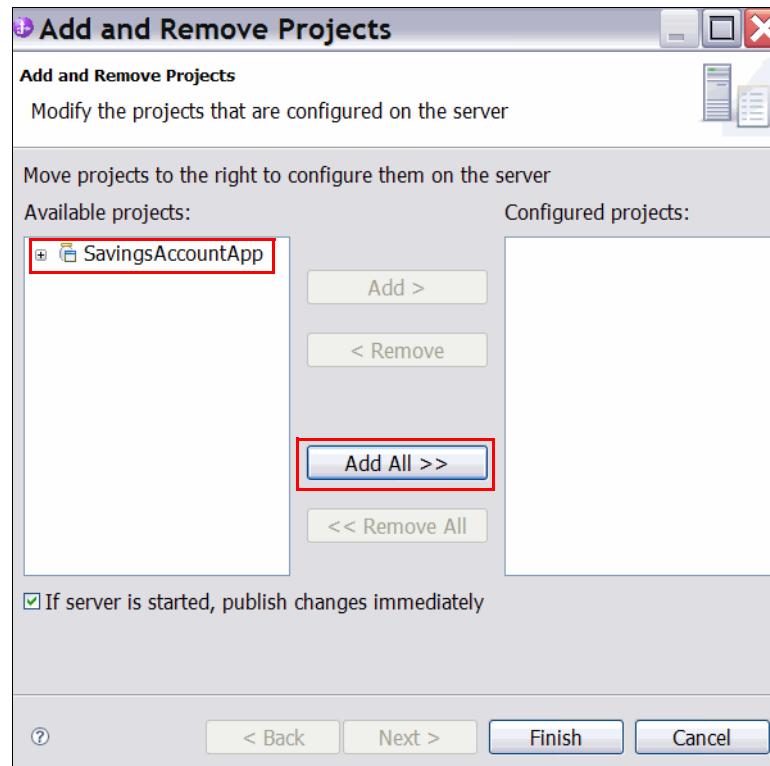


Figure 5-40 Add SavingsAccountApp project

The status for WebSphere Process Server and the deployed application are started and synchronized, as shown in Figure 5-41.

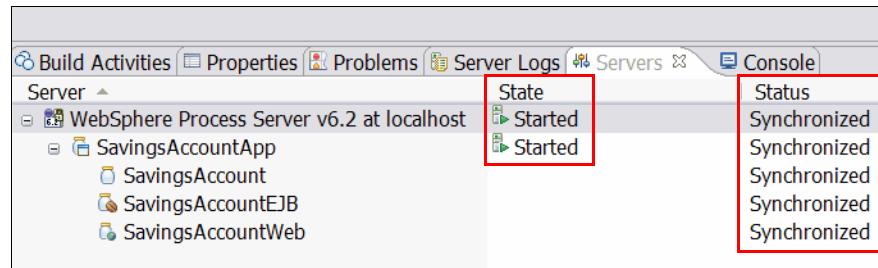


Figure 5-41 Deploy the applications

Switching back to the WebSphere Message Broker Development toolkit

To switch back to the WebSphere Message Broker Development toolkit:

1. The Web Services applications needs account files to be available in the local file system. Locate files **SavingsAccount.txt** and **CurrentAccount.txt** in the Message Broker toolkit, as shown in Figure 5-42. Click **CurrentAccount.txt**, and simultaneously hold the Ctrl key as you click **SavingsAccount.txt** to highlight both.

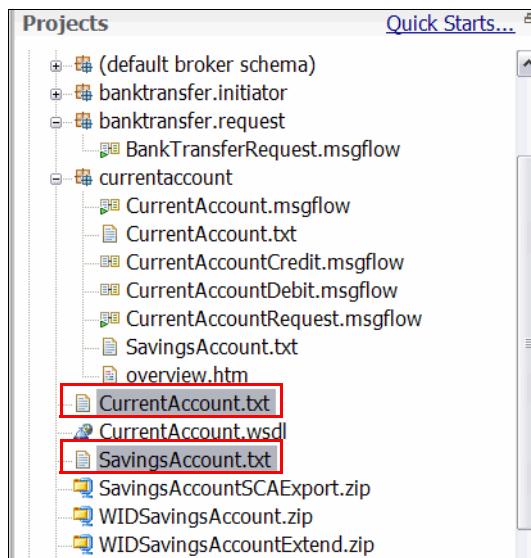


Figure 5-42 Locate two text files

2. Select **File** → **Export from the toolkit**.
3. Under the General folder, select **File System**, and click **Next**.
4. Ensure that **CurrentAccount.txt** and **SavingsAccount.txt** are selected. Specify C:\tmp (or /tmp on Linux) as the destination directory, as shown in Figure 5-43 on page 114, and click **Finish**.

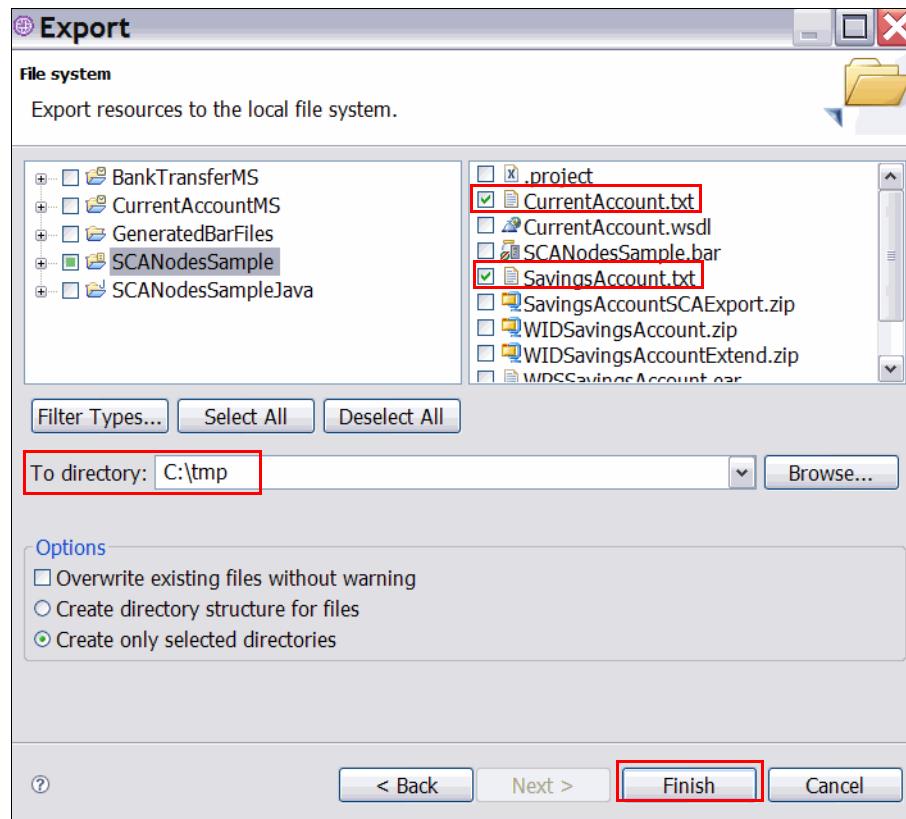


Figure 5-43 Select the files and destination

Running the Part 1a scenario

The sample uses two message flows. The first one, the Bank Transfer Request Initiator, is invoked from a browser, and it simply returns a form that is to be completed and resubmitted. The second flow, the bank transfer request, invokes a service that is running in WebSphere Process Server to carry out the required transaction. The results are returned to the message flow and subsequently to the browser window where the request was initiated:

1. Open a Web browser session to <http://localhost:7080/TransferRequestForm>.
2. From the pull-down menu, select **Transfer To Savings Account**, and in the Amount entry field, enter an amount of 50, as shown in Figure 5-44 on page 115. Click **Submit**.

Note: It might take a little time for you to receive the result.

Bank Transfer Form

Request Transfer To Savings Account

Amount 50

Submit

You have 100.00 in your Savings Account

You have 100.00 in your Current Account

Figure 5-44 Transfer money to Savings Account

- The result, shown in Figure 5-45, returns to the browser session. If it works, this scenario is complete. Click **OK** to return to the Bank Transfer Form entry display.

Your transfer of 50.00 has been successful!

Your old savings balance was 100.00

Your new savings balance is 150.00

OK

Figure 5-45 Transaction complete

5.4.4 SCA Node scenario part 1b: SCA Request

In this scenario, we demonstrate how you can use SCA Request node to integrate WebSphere Message Broker and WebSphere Process Server in a synchronous process. It is the same process as Part 1a except users replace Asynchronous Request and Asynchronous Response nodes with Request node.

On the WebSphere Message Broker toolkit

Perform the following steps:

1. Open the Message Broker toolkit.
2. You reuse the Part 1a artifacts, so copy the message flow BankTransferRequest.msgflow. Give it the new name: BankTransferSyncRequest.msgflow.
3. Double-click **BankTransferSyncRequest.msgflow**. Delete SCAAsyncRequest and SCAAsyncResponse nodes.
4. Drag-and-drop BankTransferRequest.outsca from BankTransferMS SCA Definitions onto the canvas, as shown in Figure 5-46.

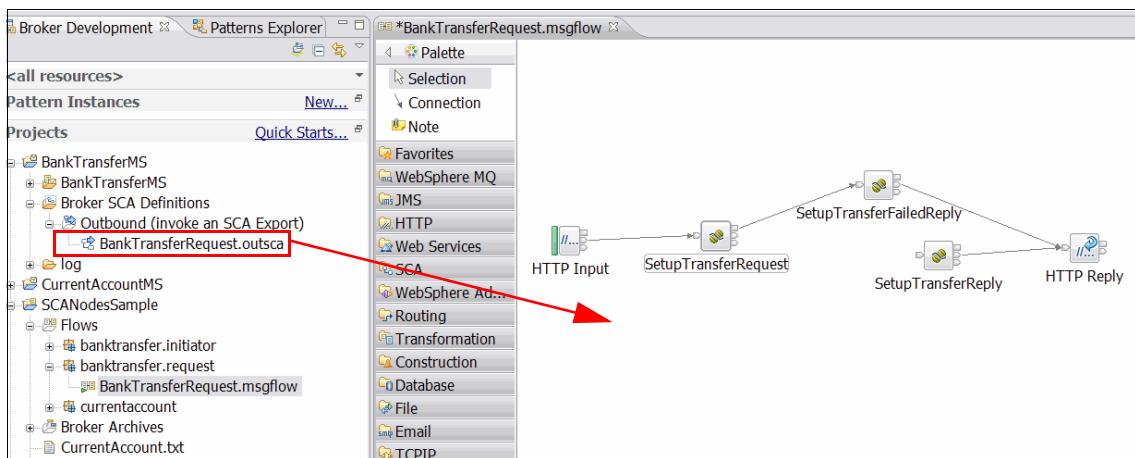


Figure 5-46 Drag BankTransferRequest.outsca to the canvas

5. A message is displayed that asks you to configure the service invocation on WebSphere Process Server, as shown in Figure 5-47 on page 117. Specify the Synchronous Invocation option, and click **OK**.

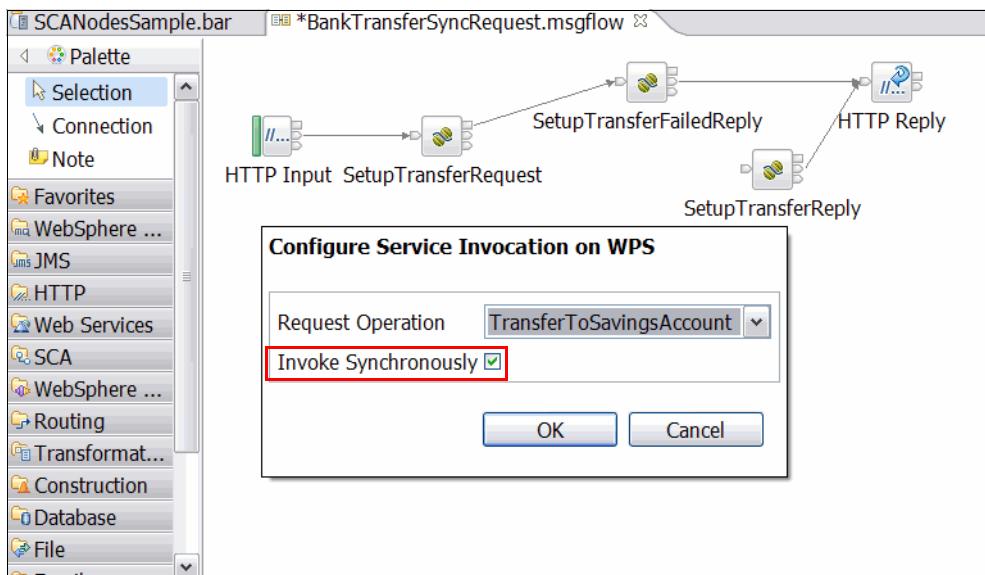


Figure 5-47 Select the *Invoke Synchronously* option for the SCA Request

6. Wire the Out terminal of SetupTransferRequest (JavaCompute) node to the In terminal of the SCA Request node and the Out terminal of the SCA Request node to the SetupTransferReply (JavaCompute) node. Select the **Basic** tab of the Properties folder, and note that the Broker SCA definition is provided by resource SCA/BankTransferRequest.outsca, as shown in Figure 5-48 on page 118. This is the resource that you exported from WID to create a message definition within Message Broker.

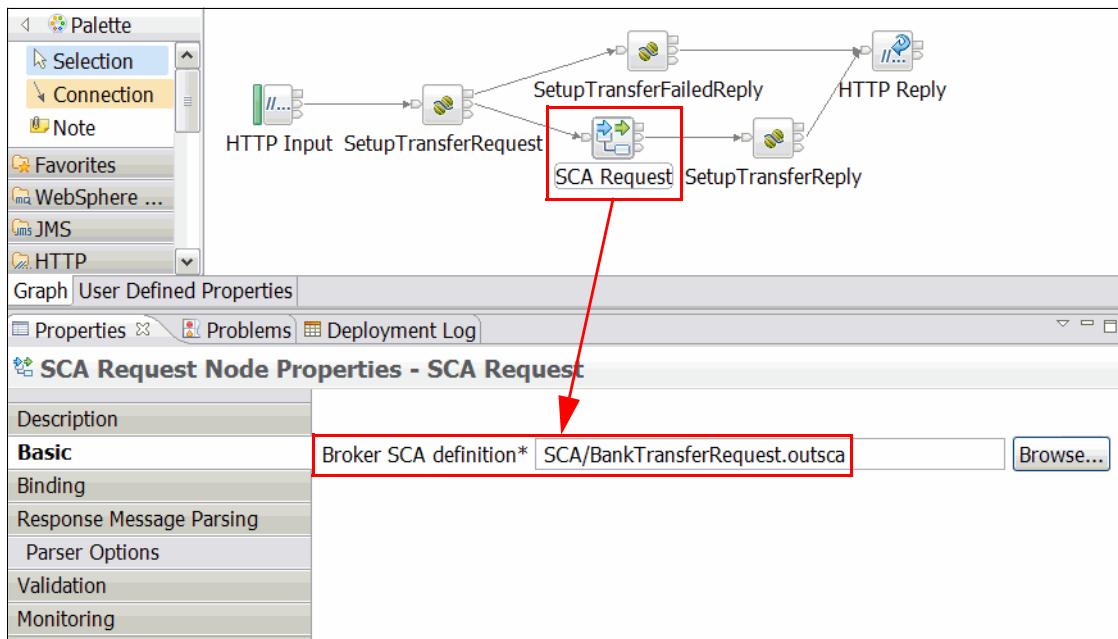


Figure 5-48 Broker SCA definition source for SCA Request Node

7. In the Properties folder, click the **Binding** tab, as shown in Figure 5-49 on page 119. Note that this uses an MQ Binding and invokes the TransferToSavingsAccount operation. Request time-out is set to 120 seconds (the default value). You can adjust this value to your own environment. The key point of this is that SCA Request node constructs an MQ message that contains the service request and sends it to a local queue, called RECEIVE, that is owned by the MB7QMR queue manager. The process that is running in WebSphere Process Server retrieves the message from the RECEIVE queue and performs the processing logic that is required. It subsequently constructs an MQ message that contains the response and puts it in the Reply-to queue, called SEND. At the same time, it blocks (for example, 120 seconds in this case) until it receives a response; otherwise, it will get time-out.

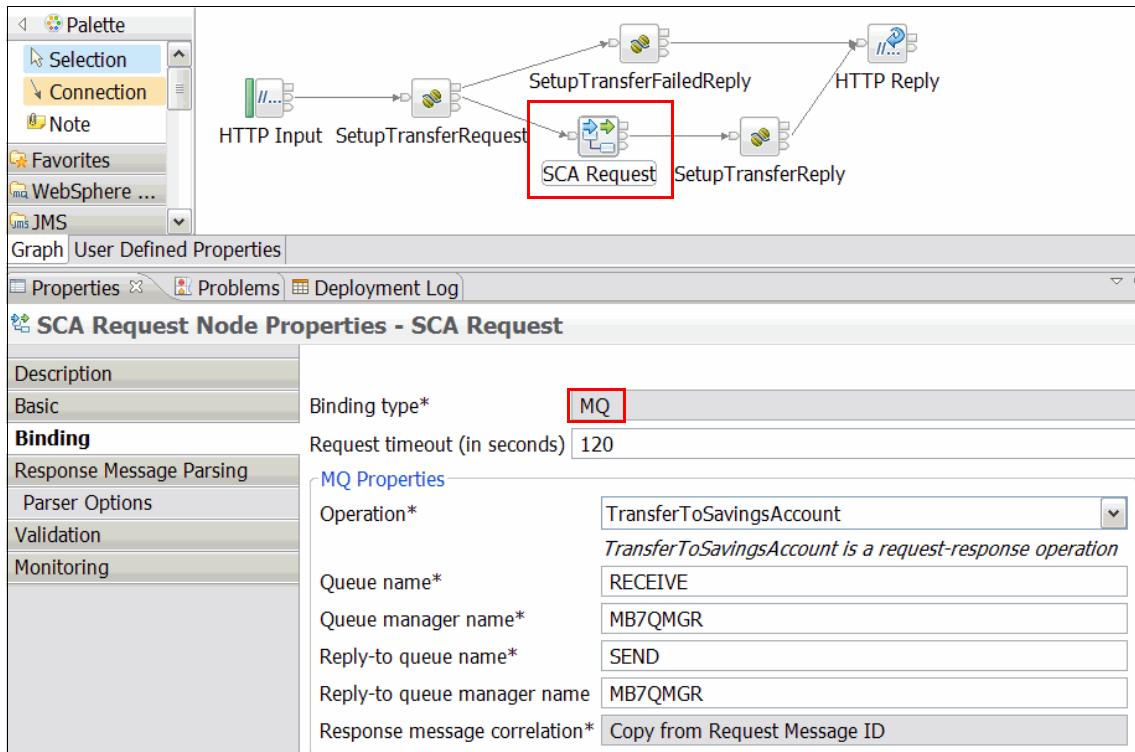


Figure 5-49 Select SCA Request node to verify MQ binding

8. Because the SCA Asynchronous nodes store the request identifiers in the user context in the Local Environment, the SCARequest node does not need to do this step. Therefore, the Java code must be updated in the JavaCompute nodes accordingly. Delete the Java code in the SetupTransferRequest JavaCompute Node, and copy and paste the Java code in Example 5-1 to this JavaCompute node. Select **Ctrl-S** to save.

Appendix A, “Additional material” on page 533 gives you the Java code in Example 5-1 in a zip file called *SetupTransferRequest_SCARequest.zip* for the SCA Request node example.

Example 5-1 Sample Java code in the SetupTransferRequest JavaCompute node for SCA Request

```

package sca.broker.sample.banktransfer;

import java.math.BigDecimal;
import com.ibm.broker.javacompute.MbJavaComputeNode;

```

```

import com.ibm.broker.plugin.*;

public class TransferRequest_SetupTransferRequest extends
MbJavaComputeNode {
    public void evaluate(MbMessageAssembly inAssembly) throws
MbException {
        MbOutputTerminal out = getOutputTerminal("out");
        MbMessage inMessage = inAssembly.getMessage();
        // create new message
        MbMessage outMessage = new MbMessage();
        try {
            // optionally copy message headers
            copyMessageHeaders(inMessage, outMessage);
            // -----
            // Add user code below
            // Get the incoming MIME parts
            MbElement inRoot = inMessage.getRootElement();
            MbElement inMime = inRoot.getFirstElementByPath("MIME");
            MbElement inParts = inMime.getFirstElementByPath("Parts");
            // Traverse the local environment for property overrides
            MbElement outRoot = outMessage.getRootElement();
            MbMessage locEnv = inAssembly.getLocalEnvironment();
            MbMessage newLocEnv = new MbMessage(locEnv);
            MbElement destination =
newLocEnv.getRootElement().getFirstElementByPath("Destination");
            // MbElement http = destination.getFirstElementByPath("HTTP");
            // MbElement requestIdentifier =
http.getFirstElementByPath("RequestIdentifier");
            MbElement sca =
destination.createElementAsLastChild(MbElement.TYPE_NAME, "SCA", null);
            MbElement request =
sca.createElementAsLastChild(MbElement.TYPE_NAME, "Request", null);

            // request.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
            "UserContext", requestIdentifier.getValue());

            // Operation
            String operation = null;

            // Transfer amount
            String amount = null;

            // Get the first part
            MbElement part = inParts.getFirstChild();

```

```

        // Iterate through all of the parts dynamically setting sca
overrides
        while (part != null) {
            // Strip out the property
            String content = (String)part.getFirstChild().getValue();
            int start = content.indexOf("\"") + 1;
            int end = content.indexOf("\"", start);
            String property = content.substring(start, end);
            // Get the corresponding property data
            MbElement data =
            part.getLastChild().getFirstElementByPath("BLOB");
            // If there is a data part then setup the relevant property
overrides
            if (data != null) {
                // Obtain the byte data
                byte[] blobData =
                (byte[])data.getFirstElementByPath("BLOB").getValue();
                // Convert to a string
                String dataString = new String(blobData);
                // Set LE properties
                if (property.equals("Operation")) {

request.createElementAsLastChild(MbElement.TYPE_NAME_VALUE, property,
dataString);
                    operation = dataString;
                } else if (property.equals("Amount")) {
                    amount = dataString;
                    try {
                        BigDecimal bigDecimal = new BigDecimal(amount);
                        if (bigDecimal.scale() > 2 || bigDecimal.scale() <
0) {
                            throw new Exception ("Invalid currency format
specified!");
                        }
                    } catch (NumberFormatException nfe) {
                        throw new Exception ("Invalid amount specified!");
                    }
                } else {
                    throw new Exception("No amount specified!");
                }
                // Get the next part
                part = part.getNextSibling();
            }
            // Delete the HTTPInputHeader since we don't need this in our
MQ message request

```

```

        MbElement httpInputHeader =
outRoot.getFirstElementByPath("HTTPInputHeader");
httpInputHeader.detach();

        // Create the Broker XMLNSC Parser element
        MbElement outParser =
outRoot.createElementAsLastChild(MbXMLNSC.PARSER_NAME);
        MbElement service =
outParser.createElementAsLastChild(MbElement.TYPE_NAME, operation,
null);

        service.setNamespace("http://SavingsAccount/SavingsAccountInterface");
        MbElement transferin =
service.createElementAsLastChild(MbElement.TYPE_NAME, "transferin",
null);
        transferin.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
"amount", amount);

        // End of user code
        // -----
        // Create the new assembly with the new property overrides
MbMessageAssembly outAssembly = new MbMessageAssembly(
        inAssembly,
        newLocEnv,
        inAssembly.getExceptionList(),
        outMessage);

        // The following should only be changed
        // if not propagating message to the 'out' terminal
out.propagate(outAssembly);

} catch (Exception e) {
    throw new MbUserException(
        TransferRequest_SetupTransferRequest.class.getName(),
        "evaluate()", "",
        "",
        e.getMessage(),
        null);
} finally {

        // clear the outMessage even if there's an exception
outMessage.clearMessage();
}

```

```

    }

    public void copyMessageHeaders(MbMessage inMessage, MbMessage
outMessage)
        throws MbException {
    MbElement outRoot = outMessage.getRootElement();

        // iterate though the headers starting with the first child of
the root
        // element
        MbElement header = inMessage.getRootElement().getFirstChild();
        while (header != null && header.getNextSibling() != null) // stop
before
            // the last
            // child
            // (body)
            {
                // copy the header and add it to the out message
                outRoot.addAsLastChild(header.copy());
                // move along to next header
                header = header.getNextSibling();
            }
        }

    }

```

9. Delete the Java code in SetupTransferReply JavaComputeNode, and copy and past the Java code in Example 5-2 to this JavaCompute node. Select **Ctrl-S** to save.

For the SCA Request sample, Appendix A, “Additional material” on page 533, gives you the Java code for Example 5-2 in a zip file called SetupTransferReply.zip.

Example 5-2 Sample Java code in the SetupTransferReply JavaCompute node for SCA Request node

```

package sca.broker.sample.banktransfer;

import java.math.BigDecimal;
import com.ibm.broker.javacompute.MbJavaComputeNode;
import com.ibm.broker.plugin.*;

public class TransferRequest_SetupTransferReply extends
MbJavaComputeNode {

```

```

    public void evaluate(MbMessageAssembly inAssembly) throws
MbException {
    MbOutputTerminal out = getOutputTerminal("out");

    MbMessage inMessage = inAssembly.getMessage();

    // create new message
    MbMessage outMessage = new MbMessage();

    try {
        // optionally copy message headers
        copyMessageHeaders(inMessage, outMessage);

        // -----
        // Add user code below

        // Traverse the local environment for property overrides
        MbMessage locEnv = inAssembly.getLocalEnvironment();
        MbMessage newLocEnv = new MbMessage(locEnv);
        MbElement locEnvRoot = newLocEnv.getRootElement();
        // MbElement soap = locEnvRoot.getFirstElementByPath("SCA");
        // MbElement response =
        soap.getFirstElementByPath("Response");
        // MbElement userContext =
        response.getFirstElementByPath("UserContext");
        // MbElement destination =
        locEnvRoot.createElementAsLastChild(MbElement.TYPE_NAME, "Destination",
        null);
        // MbElement http =
        destination.createElementAsLastChild(MbElement.TYPE_NAME, "HTTP",
        null);
        // http.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
        "RequestIdentifier", userContext.getValue());
        MbElement inRoot = inMessage.getRootElement();
        MbElement XMLNSC = inRoot.getFirstElementByPath("XMLNSC");
        MbElement transferResponse = XMLNSC.getLastChild();
        MbElement transferOut =
        transferResponse.getFirstElementByPath("transferout");
        String amount = new
        BigDecimal(transferOut.getFirstElementByPath("amount").getValue().toString())
        .setScale(2, BigDecimal.ROUND_HALF_UP).toString();
        String oldBalance = new
        BigDecimal(transferOut.getFirstElementByPath("oldbalance").getValue().toString())
        .setScale(2, BigDecimal.ROUND_HALF_UP).toString();
    }
}

```

```

        String newBalance = new
BigDecimal(transferOut.getFirstElementByPath("newbalance").getValue().t
oString()).setScale(2, BigDecimal.ROUND_HALF_UP).toString();
        String accept =
transferOut.getFirstElementByPath("accept").getValue().toString();
        // Confirmation message
        StringBuffer html = new StringBuffer();
        html.append("<HTML><HEAD>");
        html.append("<META http-equiv='Content-Type'
content='text/html; charset=ISO-8859-1'></HEAD>");
        html.append("<BODY><form action='/TransferRequestForm'
method=post>");
        if (accept.equals("yes")) {
            html.append("<h1>Your transfer of " + amount + " has been
successful!</h1>");
            html.append("<h3>Your old savings balance was " +
oldBalance + "</h3>");
            html.append("<h3>Your new savings balance is " + newBalance
+ "</h3>");
        } else {
            html.append("<h1><font color=\"#ff0000\">Your transfer of "
+ amount + " has been unsuccessful due to insufficient
funds!</font></h1>");
            html.append("<h3>Your savings balance remains unchanged at
" + newBalance + "</h3>");
        }
        html.append("<tr><td><input type='submit' name='OK'
value='OK'></td></tr>");
        html.append("</form></BODY></HTML>");
        // Set the content type to be html so that it may be viewed by
a browser
        MbElement outRoot = outMessage.getRootElement();
        MbElement outProperties =
outRoot.getFirstElementByPath("Properties");
        MbElement contentType =
outProperties.getFirstElementByPath("ContentType");
        contentType.setValue("text/html");
        // Delete the MQMD header since we don't need this in our HTTP
reply
        MbElement mqmd = outRoot.getFirstElementByPath("MQMD");
        mqmd.detach();
        // Create the Broker Blob Parser element
        MbElement outParser =
outRoot.createElementAsLastChild(MbBLOB.PARSER_NAME);

```

```

        // Create the BLOB element in the Blob parser domain with the
        required text
        outParser.createElementAsLastChild(MbElement.TYPE_NAME_VALUE,
        "BLOB", html.toString().getBytes());
        // End of user code
        // -----
        // Create the new assembly with the new property overrides
        MbMessageAssembly outAssembly = new MbMessageAssembly(
            inAssembly,
            newLocEnv,
            inAssembly.getExceptionList(),
            outMessage);
        // The following should only be changed
        // if not propagating message to the 'out' terminal
        out.propagate(outAssembly);
    } catch (Exception e) {
        throw new MbUserException(
            TransferRequest_SetupTransferReply.class.getName(),
            "evaluate()", "",
            "",
            e.getMessage(),
            null);
    } finally {
        // clear the outMessage even if there's an exception
        outMessage.clearMessage();
    }
}

public void copyMessageHeaders(MbMessage inMessage, MbMessage
outMessage)
    throws MbException {
    MbElement outRoot = outMessage.getRootElement();
    // iterate though the headers starting with the first child of
    the root
    // element
    MbElement header = inMessage.getRootElement().getFirstChild();
    while (header != null && header.getNextSibling() != null) // stop
before
    // the last
    // child
    // (body)
    {
        // copy the header and add it to the out message
        outRoot.addAsLastChild(header.copy());
    }
}

```

```

        // move along to next header
        header = header.getNextSibling();
    }
}

}

```

- 10.Verify and deploy the message flows and message sets to the Message Broker runtime in the SCANodesSample Execution Group (same as step 10 on page 106 to 14 on page 108 in Part 1a).
- 11.Ensure that the deployment is successful.
- 12.Run the Part 1b Scenario (SCA Request) by repeating step 2 on page 113 to step 1 on page 114.

5.4.5 SCA Nodes scenario part 2: A Message Broker flow implements a service using SCA Input and SCA Reply

In this section, we discuss the WebSphere Message Broker flow that implements a service using SCA Input and an SCA Reply.

Background information

In this part of the scenario, a message broker is called from a process that is running within WebSphere Process Server. You build up the top-level flow that implements the service; however, you use pre-built subflows that contain the processing logic.

Figure 5-50 shows the main message flow structure.

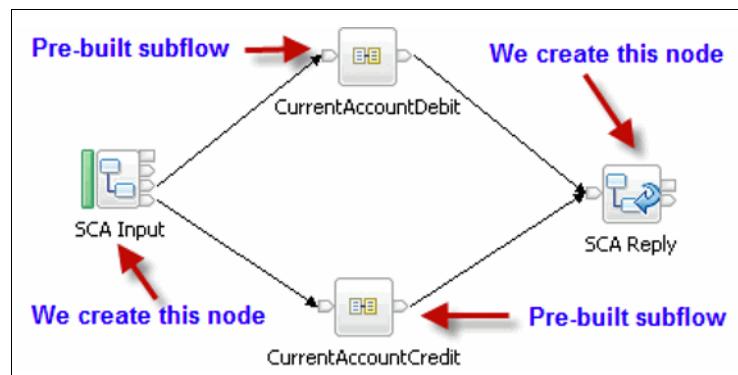


Figure 5-50 Current Account business logic flow

The SCA Input node receives the service request using the binding that is specified for the interface, which in this case uses HTTP transport. The message is propagated from one of two terminals to the CurrentAccountDebit subflow or the CurrentAccountCredit subflow depending on the operation that was specified. The processing logic is completed by JavaCompute nodes that are located in the subflows.

The starting point for this part of the scenario is a deployable WSDL file, CurrentAccountRequestService.wsdl, which was created from the message set CurrentAccountMS. We already completed this step, although it is a straightforward matter to create the deployable WSDL file from the message set. The WSDL file specifies the operations, inputs, and outputs, as shown in Figure 5-51. This file is opened in the graphical WSDL editor.

CurrentAccountMSPortType		
CurrentAccountDebit		
input	CurrentAccountDebit	CurrentAccountDebit
output	CurrentAccountDebitResponse	CurrentAccountDebitResponse
CurrentAccountCredit		
input	CurrentAccountCredit	CurrentAccountCredit
output	CurrentAccountCreditResponse	CurrentAccountCreditResponse

Figure 5-51 Current Account Port type

Building and deploying the resources on the WebSphere Message Broker toolkit

To build and deploy the resources on the WebSphere Message Broker toolkit:

1. Locate the message set CurrentAccountMS, and note that it includes the deployable WSDL file CurrentAccountRequestService.wsdl. Select **Generate → Broker SCA Definition**.
2. You are creating an inbound Broker SCA definition (the request for the service is being sent to Broker). The Binding Type is Web Service, which uses HTTP transport. The alternative choice at this point is MQ transport. Give the Broker SCA definition the name CurrentAccountRequest, as shown in Figure 5-52 on page 129, and click **Next**.

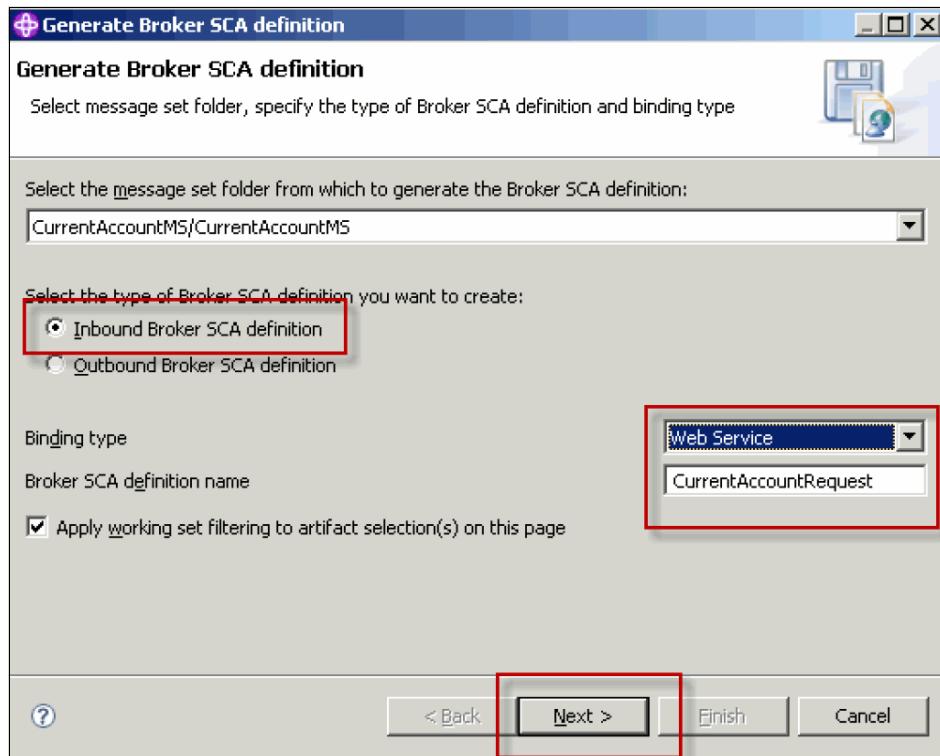


Figure 5-52 Generating new inbound Broker SCA with Web Service binding type

3. Select **Use an existing deployable WSDL from the workspace**. Select **CurrentAccountRequestService.wsdl**, and click **Next**. Confirm that the details are correct, and click **Finish**, as shown in Figure 5-53 on page 130.

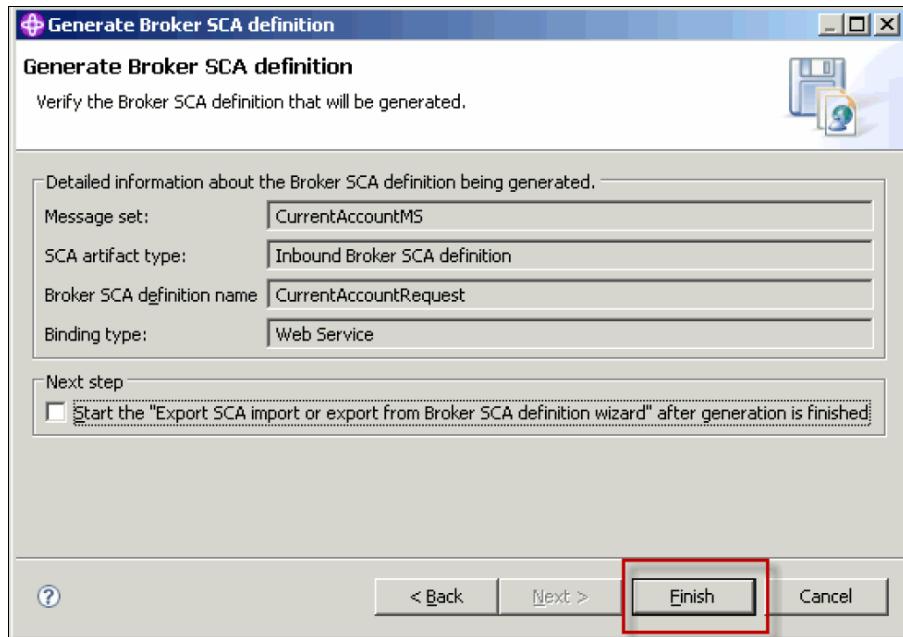


Figure 5-53 Generate CurrentAccountRequest SCA definition

Note that in the preceding step you did not select the option **Start the Export SCA import or export from Broker SCA definition wizard after generation is finished** because this option creates a set of components that you can import into WID to help build a project, which invokes the service that is represented by the Broker SCA definition. We already completed this step for you in the sample that is in the resultant project being encapsulated in project interchange file WIDSavingsAccountExtend.zip. You will use this PI file later.

4. The next step is to create the message flow that implements the CurrentAccountDebit and CurrentAccountCredit operations, shown in Figure 5-54 on page 131. Expand the Message Flow Project SCANodesSample, and note the two flows that were already created for you in the currentaccount schema called CurrentAccountDebit.msgflow and CurrentAccountCredit.msgflow. Use these as subflows in the top-level flow that you will create.

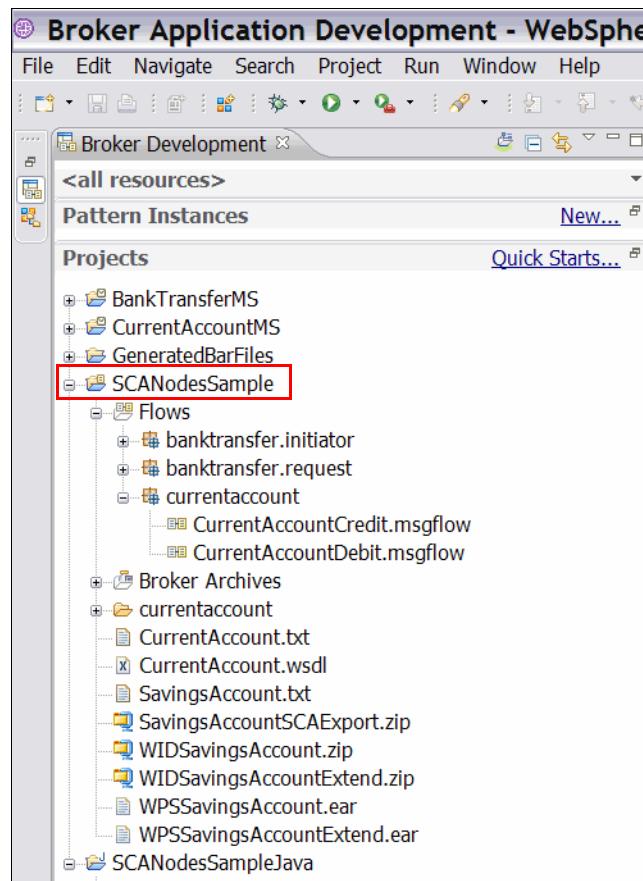


Figure 5-54 Two subflows for Credit and Debit

5. Click the **currentaccount** schema, and then select **New → Message Flow**. Call the new message flow **CurrentAccountRequest**, locate it in the **currentaccount** schema, and click **Finish**, as shown in Figure 5-55 on page 132.

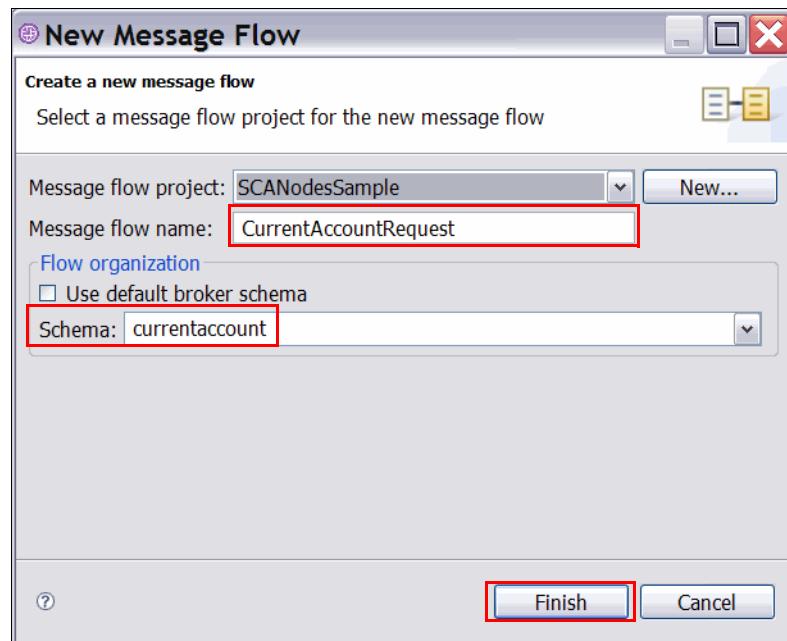


Figure 5-55 CurrentAccountRequest message flow

6. Locate the Broker Inbound SCA definition `CurrentAccountRequest.insca` that you created in the previous steps, and drag-and-drop it onto the canvas that opens for new message flow `CurrentAccountRequest`, as shown in Figure 5-56 on page 133.

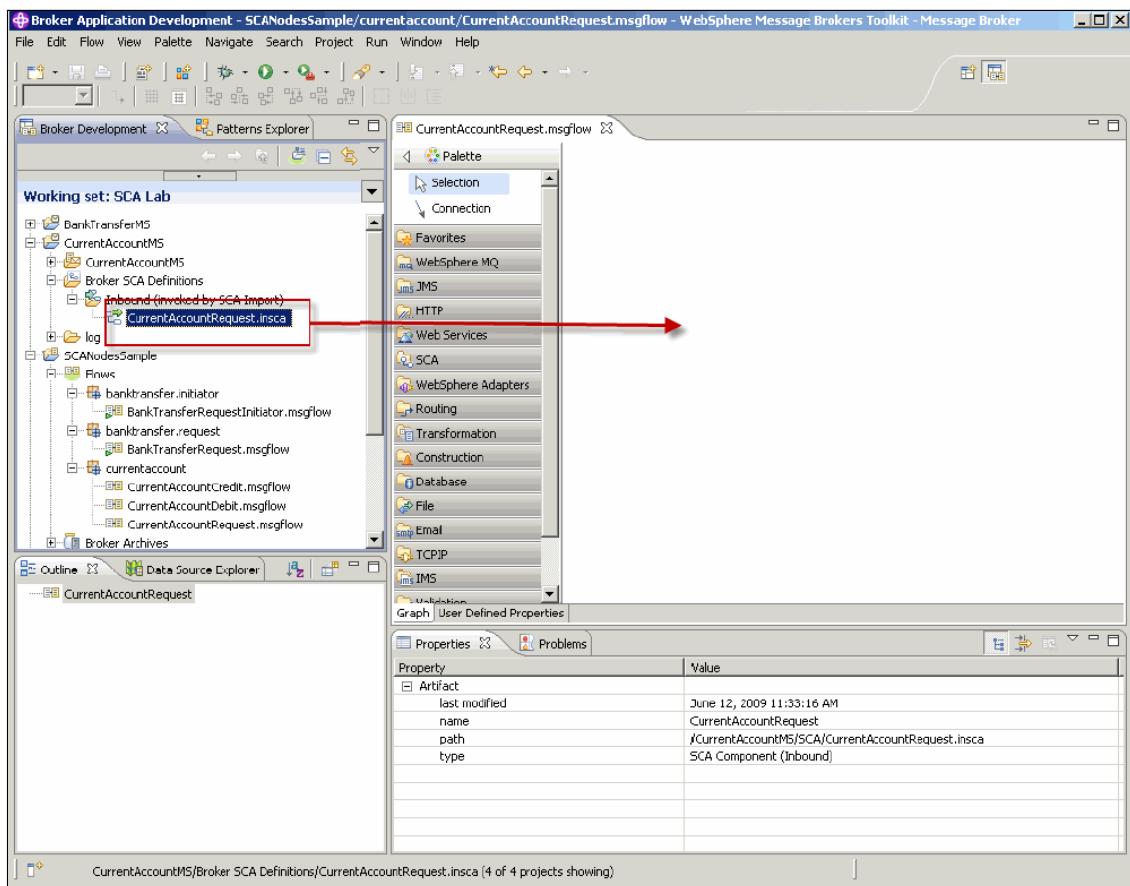


Figure 5-56 Drag Inbound SCA definition to create message flow

7. A pair of properly configured SCACInput and SCACReply nodes are created. Left-click **SCA Input node**, and examine its properties, as shown in Figure 5-57 on page 134, which are derived from the WSDL file.

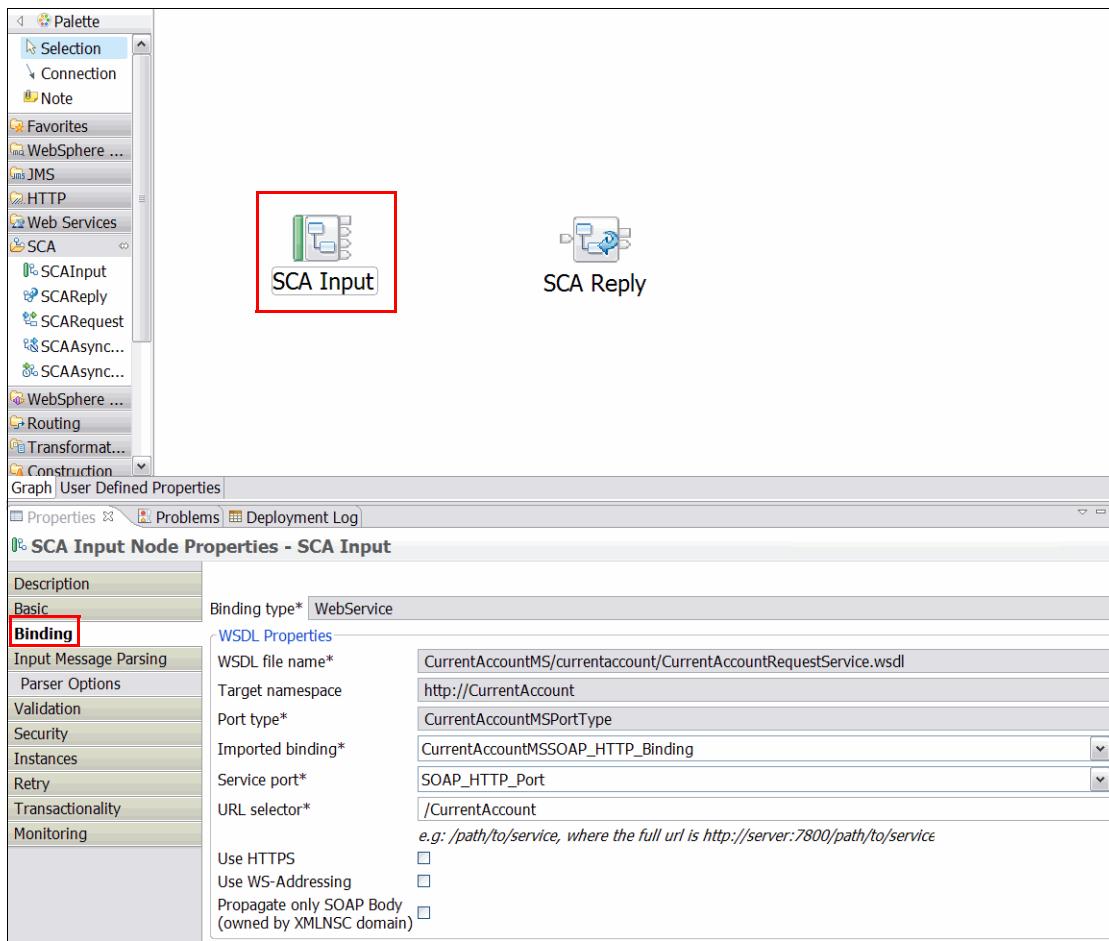


Figure 5-57 Display SCAInput binding properties

The SCA Input node provides key features (applicable only when the Binding type is Web Services) to generate dynamic terminals. A dynamic terminal is generated for each operation that is supported by the port type and implemented by the imported binding. The dynamic operation terminal to which the SCA message is routed depends on the operation that is defined in the SCA message when it is received. Note that a terminal is created on the SCA Input node for each of the operations that are supported by the interface for this message flow called CurrentAccountCredit and CurrentAccountDebit. Figure 5-58 on page 135 shows the SCA Input node with credit and debit terminals.

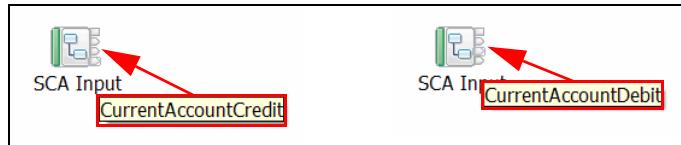


Figure 5-58 SCA Input node with credit and debit terminals

8. Add the subflows that implement the business logic for the services that this message flow will provide. Right-click anywhere on the canvas for the message flow, and select **Add Subflow**. Select **CurrentAccountCredit.msgflow**, and click **OK**, as shown in Figure 5-59.

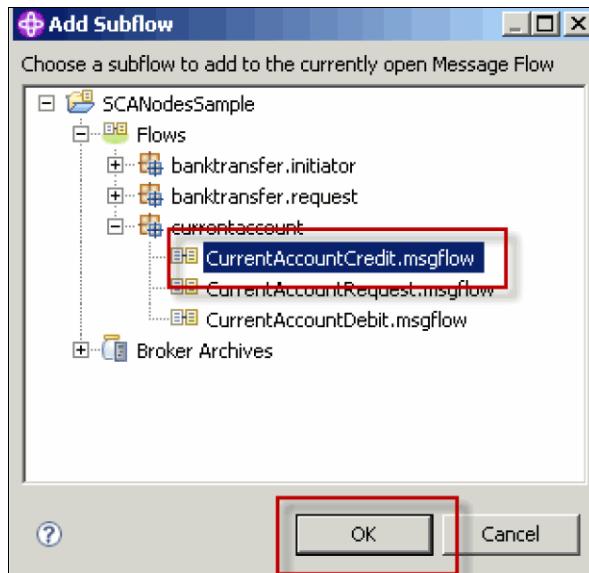


Figure 5-59 Add Credit subflow

9. Drag the subflow that you added to a convenient point on the canvas, as shown in Figure 5-60 on page 136, then:
 - a. Wire SCAInput node terminal CurrentAccountCredit to its input terminal.
 - b. Wire its output terminal to the input terminal of the SCAResponse node.

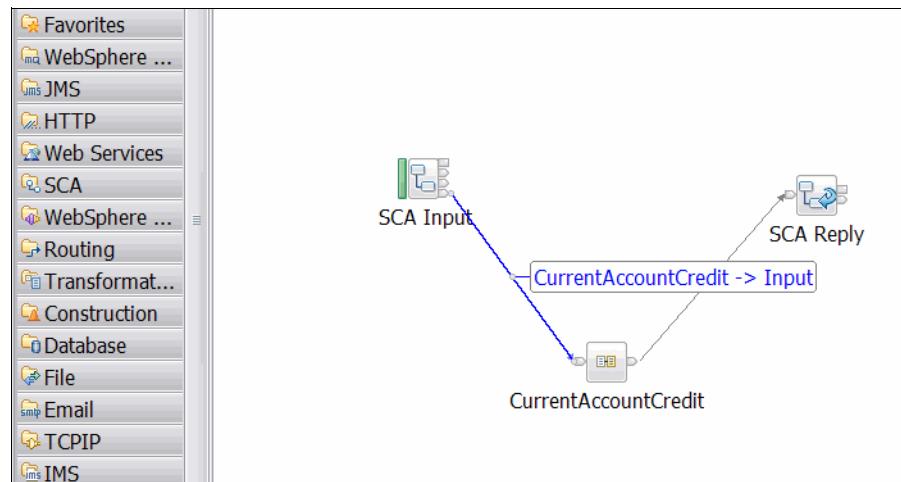


Figure 5-60 Current Account Credit flow

10. Repeat the same steps for CurrentAccountDebit message flow. The final diagram looks similar to Figure 5-61.

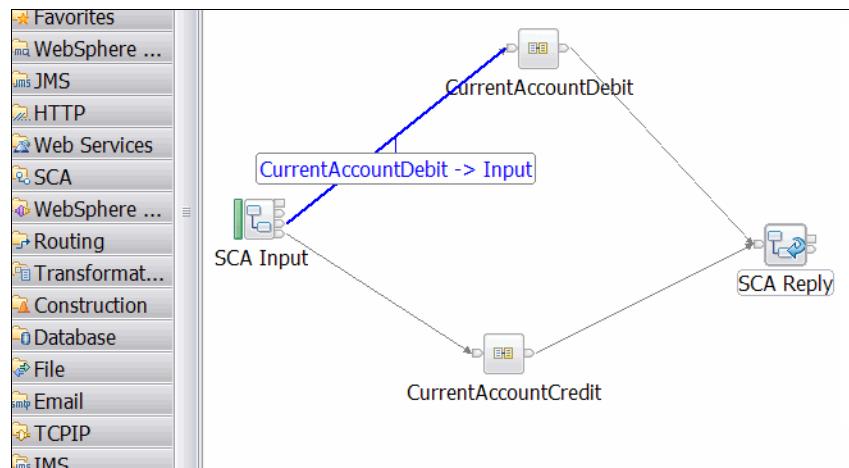


Figure 5-61 Current Account Request Message flow

11. The next step is to build a broker archive file with the new resources and deploy to the message broker. In the Message Broker toolkit, locate and double-click the Broker Archive file SCANodesSample.bar to open it in the broker archive file editor. Check all Message Flows and all Message Sets, as shown in Figure 5-62 on page 137. Select the **Remove contents of Broker**

Archive before building and **Override configurable property values** options, and then click **Build broker archive**.

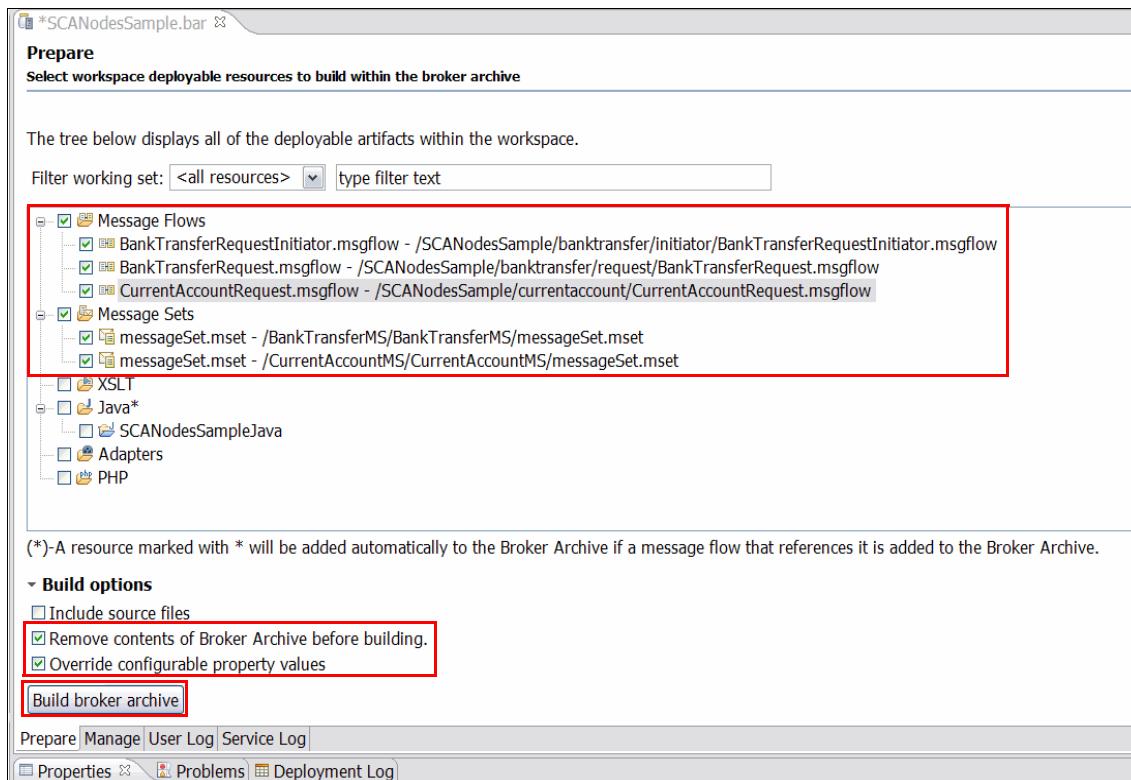


Figure 5-62 Deploy message flows and message sets

12. Deploy the Broker Archive file, and press **Ctl-S** to save it. Drag-and-drop the Broker Archive file from the Message Broker toolkit onto the Message Broker Execution Group called SCANodeSample in MQ Explorer. Ensure that the deployment is successful.
13. Make the extended SavingsAccount project available to WID. Ensure that WID is started with the Business Integration perspective selected. Click **File**, and select **Import**.
14. Select **Project Interchange** under the Other folder, and select **Next**. Browse to locate the Project Interchange file in C:\ ...(your) directory ...\\SCANodeSample\\.
15. Highlight **WIDSavingsAccountExtend.zip**, and click **Open**.
16. Click **Select All** and **Finish**, as shown in Figure 5-63 on page 138. Confirm that you want to overwrite the existing SavingsAccount project.

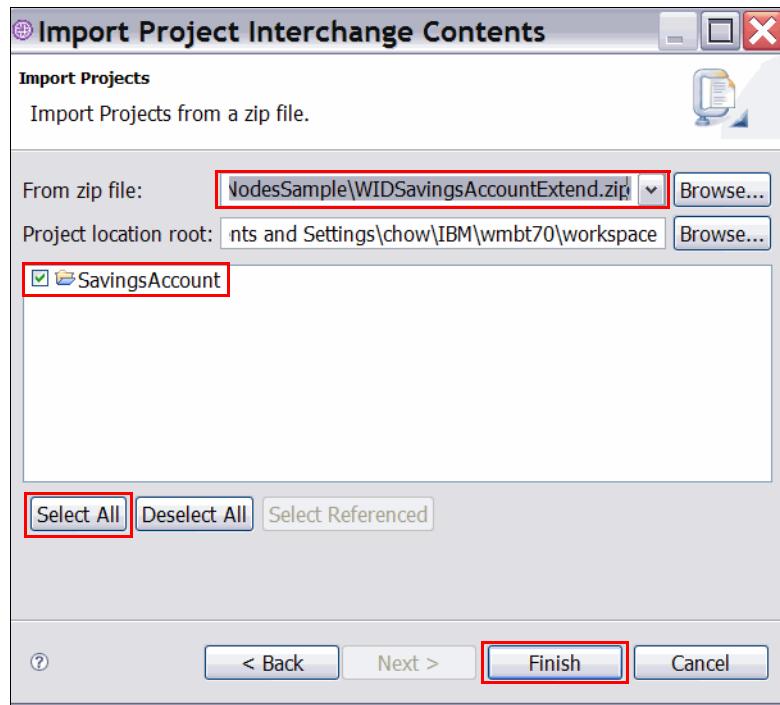


Figure 5-63 Import WIDSavingsAccountExtend.zip

17. Open the assemble diagram for SavingsAccount, and observe that it now includes the service CurrentAccountRequest that is implemented in the Message Broker message flow of the same name, as shown in Figure 5-64 on page 139.

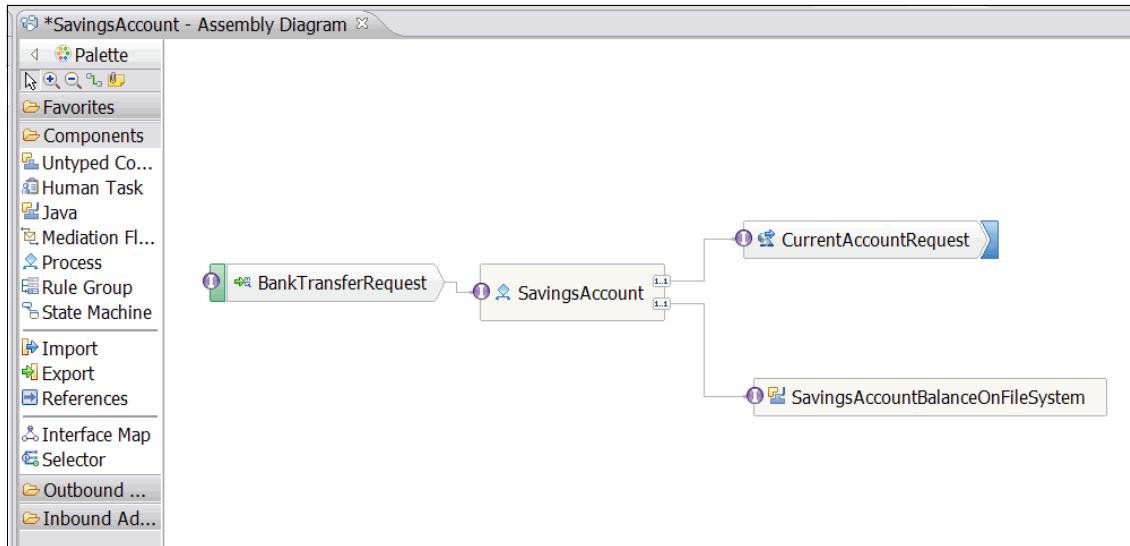


Figure 5-64 CurrentAccountRequest is linked between WebSphere Process Server and WebSphere Message Broker

18. Re-deploy the new version of SavingsAccount onto WebSphere Process Server. Right-click **WPS**, and select **Add and Remove Project**. Highlight **SavingsAccountApp**, and click **Remove** → **Finish**. Add the new version back to WebSphere Process Server by clicking **WPS** and selecting **Add and Remove Projects**. Add **SavingsAccountApp**, and click **Finish**. Wait until the resources are shown to be started and synchronized.

Run the Part 2 scenario: SCA Input and SCA Reply

To run the Part 2 scenario:

1. The application in the second SCA scenario is different from the first scenario because transfers to and from the Savings account are linked to the Current Account. In a Web browser session, open location <http://localhost:7080/TransferRequestForm>.
2. From the pull-down menu, select **Transfer To Savings Account**. In the Amount Entry field, type an amount of 50, and click **Submit**, as shown in Figure 5-65 on page 140.

Bank Transfer Form

Request: Transfer To Savings Account

Amount: 50

Submit

You have 100.00 in your Savings Account

You have 100.00 in your Current Account

Figure 5-65 Transfer money to Savings Account

Figure 5-66 shows the results that are returned to the browser session.

Your transfer of 50.00 has been successful!

Your old savings balance was 100.00

Your new savings balance is 150.00

OK

Figure 5-66 Result of the transfer to Savings Account

3. Select **OK** to return to the Bank Transfer Form entry display.
4. At this point, you can see that the Current Account balance was debited by the amount that was credited to the Savings Account, as shown in Figure 5-67 on page 141.

Bank Transfer Form

Request Transfer To Savings Account

Amount

Submit

You have 150.00 in your Savings Account

You have 50.00 in your Current Account

Figure 5-67 Increase amount to Savings

5. As a final test, from the pull-down menu, select **Transfer To Current Account**, enter any amount (say 50), and click **Submit**, as shown in Figure 5-68.

Bank Transfer Form

Request Transfer To Current Account

Amount 50

Submit

You have 150.00 in your Savings Account

You have 50.00 in your Current Account

Figure 5-68 Transfer \$50 to Current Account

6. The response indicates success. Select **OK** to return. You should see that the Current Account was credited by the amount you specified, as shown in Figure 5-69 on page 142.

7. If users try to transfer the amount more than the current account has, you will see that the transaction is unsuccessful due to a fund shortage, as shown in Figure 5-70.

Bank Transfer Form

Request Transfer To Savings Account

Amount 101

Submit

You have 100.00 in your Savings Account

You have 100.00 in your Current Account

Figure 5-69 Transfer more than the current account balance

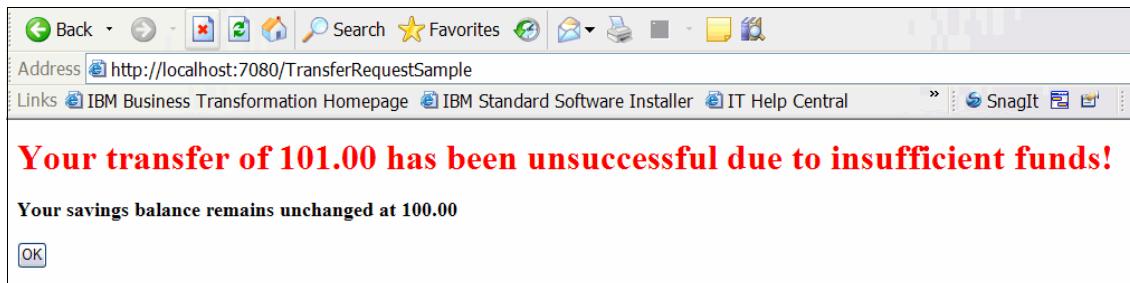


Figure 5-70 Transaction is unsuccessful due to overdraft

Your test is complete.



Enterprise Resource Planning integration with WebSphere Message Broker

In this chapter, we introduce the WebSphere Adapters nodes and describe how to configure and deploy these nodes. We also discuss the required resources that are necessary to enable communication between the WebSphere Message Broker and an enterprise information system (EIS), such as SAP, Siebel, and PeopleSoft.

In this chapter, we explain the terms that are associated with the WebSphere Adapters, describe the different resources that are required and how to generate these, and provide scenarios that illustrate how the nodes can be used for both inbound and outbound communication with the EIS.

Specifically, we cover the following topics:

- ▶ “Terms associated with WebSphere Adapters” on page 144
- ▶ “WebSphere Message Broker V7.0 enhancements to the WebSphere Adapters” on page 145
- ▶ “Using WebSphere Adapters in a message flow” on page 147
- ▶ “Scenarios” on page 154

6.1 Terms associated with WebSphere Adapters

In this section, we discuss the terms that are associated with WebSphere Adapters:

- ▶ Enterprise information system (EIS)

Enterprise information system describes the applications that the enterprise uses for handling company-wide information. An EIS provides a well-defined set of services that are exposed as local or remote interfaces or both.

Enterprise resource planning (ERP) and customer relationship management (CRM) are examples of typical enterprise information systems.

- ▶ Enterprise Metadata Discovery (EMD)

Enterprise Metadata Discovery is a specification that can be used to examine an EIS and determine the details of the business object data structures and APIs that exist. An EMD stores definitions as XML schemas, by default, and builds components that can access the EIS. WebSphere Message Broker uses the Adapter Connection wizard to examine an EIS.

- ▶ Business object

A business object is a structure that consists of data, the action to be performed on the data, and additional instructions, if any, for processing the data. The data can represent either a business entity, such as an invoice or an employee record, or unstructured text.

- ▶ Inbound and Outbound communication

The WebSphere Adapters support two modes of communication:

- Inbound: An event is generated on the EIS and the adapter responds to the event by sending a message to the message broker. The WebSphere Adapter's input nodes support inbound communication.
- Outbound: The message broker uses the adapter to send a request to the EIS. The WebSphere Adapter's request nodes support outbound communication.

6.2 EIS connectivity with JCA adapters

Prior to WebSphere Message Broker V6.1, connectivity to an EIS systems was accomplished using IBM WebSphere Business Integration adapters. These adapters were not integrated into the message flow; instead, they operated as stand-alone adapters that ran outside of the broker runtime. Message flows connected to these adapters using a JMS binding.

With WebSphere Message Broker V6.1, JCA-based WebSphere Adapters are delivered as built-in message flow nodes to provide connectivity to three major EIS systems: SAP, Siebel, and PeopleSoft systems. Providing this connectivity as nodes simplifies management and improves performance for key integration scenarios. Support is provided for inbound and outbound scenarios.

Using the adapter nodes allows you to integrate EIS systems into your business solutions through your ESB. The message flows provide the transitional capabilities to allow communication between the EIS systems and other components of the solution.

6.3 WebSphere Message Broker V7.0 enhancements to the WebSphere Adapters

WebSphere Message Broker V7.0 provides significant extensions for the supported EIS applications:

- ▶ Siebel and PeopleSoft operational reconfiguration enhancements:
 - Eases promotion of Siebel and PeopleSoft message flows through the Test, QA, and Production life cycle.
 - New configurable service provides reconfiguration for key adapter node properties, for example, Hostname, Client ID, System number, UserID, Password.
 - Also supports the wholesale replacement of the adapter connection.
- ▶ Specific SAP enhancements:
 - A Single Program ID allows multiple IDocs to be handled by different flows without disruption.
 - Synchronous Callback Interface (SCI) support with the SAPReply node. The SAPReply node can be used to send a reply to an SAP synchronous callout.

Note: For additional information about this subject, refer to the topic SAPReply node in the WebSphere Message Broker V7.0 Information Center:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/bc22000_.htm

- High availability (HA) for SAP Input nodes allows multi-broker failover with MQ HA store.

Note: For additional information about this subject, refer to the topic SAP high availability in the WebSphere Message Broker V7.0 Information Center:
http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/bc22070_.htm

- The Use wait parameter before calling BAPI commit parameter can be used to control whether the SAP adapter waits for SAP to commit the updates synchronously or issues a commit and returns while the SAP commit occurs asynchronously.

Note: For additional information about this subject, refer to the topic SAP BAPI transaction commit in the WebSphere Message Broker V7.0 Information Center:
http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac66390_.htm

- The Adapter Connection wizard includes *Iterative Discovery* to allow the addition of newly discovered objects into an existing adapter component without modifying any existing objects. You can rediscover certain objects for an inbound or outbound adapter.

Note: For additional information about this subject, refer to the topic SAP options for rediscovery in the WebSphere Message Broker V7.0 Information Center:
http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/bc22220_.htm

- ▶ General enhancements:
 - Incremental Deployment to ease the addition of new definitions to existing deployments:
 - This is very important for the addition of new artifacts in Siebel and SAP scenarios.
 - Multiple adapters can use the same connection configuration.
 - User defined operations are supported, in addition to the existing Create, Read, Update, and Delete operations:
 - More expressive operations can be declared in addition to the standard suite.

6.4 WebSphere Adapter overview

Using the WebSphere Adapters for SAP, Siebel, and PeopleSoft you can create integrated processes that exchange information with the EIS server through a standard interface. This interface shields the message flow developer from having to understand the lower-level details regarding the implementation of the application or the data structures that are involved.

By using the WebSphere Adapter, an application component that performs a specific business function can send requests to the EIS server, such as, to query a customer record, update an order document, receive events from the server, or be notified that a customer record is updated.

WebSphere Adapters for SAP Software, Siebel Business Applications and PeopleSoft Enterprise complies with the Java Connector Architecture (JCA) and standardizes the way application components, application servers, and enterprise information systems interact with each other.

The adapter configuration, which you generate with the Adapter Connection wizard, uses a standard interface and standard data objects. The adapter takes the standard data object that the application component sends and calls the EIS function. The adapter then returns a standard data object to the application component. The application component does not have to deal directly with the EIS function because it is the EIS-specific adapter that calls the function and returns the results, for example, an application component can request a list of customers and sends a standard business object with the range of customer IDs to the EIS-specific adapter. The application component receives, in return, the results in the form of a standard business object that contains the list of customers. The adapter completes all of the interactions directly with the EIS function.

Similarly, the message flow might want to know about a change to the data on the EIS server, for example, a change to a particular customer. You can generate an adapter component that listens for such events on the EIS server and notifies message flows with the update. In this case, the interaction begins on the EIS server.

6.5 Using WebSphere Adapters in a message flow

In this section, we provide the required steps to incorporate the WebSphere Adapters into a message broker message flow.

6.5.1 Step 1: Preparing the environment for WebSphere Adapters nodes

Before you can develop message flows that use WebSphere Adapters nodes, you must set up the broker runtime environment so that it can access the EIS.

To enable the WebSphere Adapters nodes in the broker runtime environment, configure the broker with the location of the EIS provider JAR files and native libraries. On Windows, the directory cannot be a mapped network drive on a remote Windows computer; the directory must be local or on a Storage Area Network (SAN) disk:

1. The WebSphere Adapters nodes require libraries from the EIS provider. For more information about how to obtain and use these libraries, see “Developing message flow applications that use WebSphere Adapters” in the WebSphere Message Broker V7.0 Information Center:
<http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp>
2. Use the **mqsichangeproperties** command to make the JAR files and shared libraries available to the WebSphere Adapters nodes. Use the following commands to set up the dependencies:
 - **mqsichangeproperties broker_name -c EISProviders -o EIS_type -n jarsURL -v jar_directory**
 - **mqsichangeproperties broker_name -c EISProviders -o EIS_type -n nativeLibs -v bin_directory**

6.5.2 Step 2: Discovering the EIS objects and services

The Adapter Connection wizard is used to create the resources that enable the WebSphere Adapters to connect to an Enterprise Information System.

A message flow application that uses one of the WebSphere Adapters requires the following resources:

- ▶ One or more message flows that contain one or more WebSphere Adapter nodes
- ▶ A message set that contains the XML Schema Definitions (XSD) for the business objects in the Enterprise Information System
- ▶ An adapter component file for the WebSphere Adapter that is being used (inbound or outbound)

The following information is provided to the Adapter Connection wizard, which automatically creates the required resources:

- ▶ The communication mode (inbound or outbound) for connection
- ▶ The connection properties for the EIS server, for example, host name, userID, and password
- ▶ The interface for use, which is specific to the type of adapter
- ▶ The business objects or services in the EIS system. The Adapter Connection wizard discovers the objects and services that are available and allows you to select from a list
- ▶ The operations to perform on the selected objects of services, for example, update, create, and delete operations. The operations that are available depends on the type of adapter

6.5.3 Step 3: Adding the WebSphere Adapters node to the message flow

The adapter component, which the Adapter Connection wizard generates, can be dragged and dropped into the message flow, which creates the appropriate WebSphere Adapters input or request node and associates the message set and adapter component with the node.

When you create a new input node in this manner, the node is automatically wired to a subflow node. The subflow contains a RouteToLabel node, which forwards the incoming message to the appropriate label node. The number of label nodes that are available in the subflow depend on the number of operations you selected when configuring the adapter.

The configuration parameters for the WebSphere Adapters node can be changed in the message flow. The EIS method to invoke and the output data location can be changed in the request node. For input nodes, you can set the \$LocalEnvironment/Adapter/MethodName variable to the name of the business method that corresponds to the EIS event that triggered the message delivery.

6.5.4 Step 4: Creating and deploying the broker archive file to the broker

The resources that the message flow requires are added to a broker archive (BAR) file and deployed to the broker. These resources include the message flow, the message set, and an adapter component.

6.6 The supported SAP adapter interfaces

The SAP adapter supports ALE and BAPI flows on inbound and outbound connections. An Application Link Enabling (ALE) interface is used to exchange data between two or more SAP systems or between SAP and an external system. Business Application Programming (BAPI) is an interface that provides access to processes and data in business application systems, such as R/3.

6.6.1 SAP inbound operations

The SAP adapter can connect to an SAP EIS and listen for events through the use of the SAP Java Connector (SAP JCo) API. After SAP sends an event, the SAP adapter can read the event and convert it from native SAP EIS format to the format that is required for processing in the message flow.

The Adapter Connection wizard is used to discover the EIS objects in SAP and generate message sets based on the structure of the objects. The SAP Inbound Adapter that is generated from the Adapter Connection wizard can be used to connect to SAP and retrieve IDocs, extract data from the IDocs, and populate a message to pass on to the next node in the flow.

The SAPInput node reads the data that is received from SAP. The SAPInput node supports the following interfaces:

- ▶ Advanced Event Processing (AEP)

For inbound processing, the AEP interface polling mechanism retrieves events in SAP, converts them into business objects, and sends the event data as business objects to WebSphere Message Broker.

- ▶ Application Link Enabling (ALE)

For inbound processing, SAP pushes the data in IDocs to the SAP adapter, which then converts the IDocs to business objects and delivers them to the endpoint.

- ▶ ALE pass through IDOC

The inbound processing is the same as with the ALE interface. However, in this mode, the bit stream for the IDoc is provided without any form of parsing. The business object that is created contains one field, which is the bit stream of the IDoc.

- ▶ Business Application Programming Interface (BAPI)

For inbound processing, the SAP server sends an RFC-enabled function (such as a BAPI function) through the adapter to the endpoint. The SAP

adapter supports inbound processing (from the SAP server to the adapter) for simple BAPIs only, that is, a BAPI that performs a single operation.

6.6.2 SAP outbound operations

For all outbound requests, the SAP adapter receives data from an external source and then updates or retrieves the data into the SAP system. The adapter transforms the message tree into the JCA CCI record that it requires to perform the outbound operation. The format of the message is determined as part of the discovery process when the Adapter Connection wizard is run.

The SAPRequest node supports the following interfaces:

- ▶ Advanced Event Processing (AEP)

For outbound processing, the SAP adapter processes events that are sent from an application to retrieve data from, or update in, the SAP server.

- ▶ Application Link Enabling (ALE)

For outbound processing, the SAP adapter converts the business object to an IDoc and delivers it to the SAP system.

- ▶ ALE pass through IDOC

The outbound processing is the same as with the ALE interface except that the business object data is in the form of a bit stream, rather than structured.

- ▶ Business Application Programming Interface (simple BAPIs)

For outbound processing, message flows call BAPIs and other RFC-enabled functions on the SAP server. The adapter supports simple BAPI calls by representing each with a single business object schema.

- ▶ BAPI work unit interface

A BAPI work unit consists of a set of BAPIs that are processed in sequence to complete a task. In this mode, the message flow can request that a sequence of separate functions be executed with a single call and ensure that the functions are executed in the specified sequence.

- ▶ BAPI result set interface

BAPI result sets use the GetList and GetDetail functions to retrieve an array of data from the SAP server. The information that is returned from the GetList function is used as input to the GetDetail function.

- ▶ Query interface for SAP Software (QISS)

The QISS interface provides the means to retrieve data from application tables on the SAP server or to query SAP application tables for the existence of data. The adapter can perform hierarchical data retrieval from the SAP

application tables. QISS supports outbound interactions for read operations (RetrieveAll and Exists) only.

6.7 Application Integration Patterns

A pattern can be used to generate customized solutions to a recurring problem in an efficient way. Patterns are provided to encourage the adoption of best practice in message flow design. WebSphere products provide tools in support of patterns that:

- ▶ Give guidance for the implementation of solutions
- ▶ Increase development efficiency because resources are generated from a set of predefined templates
- ▶ Results in higher quality solutions through reuse of assets and common implementation of aspects, such as error handling and logging

Application integration is a collection of technologies and services that form a middleware to enable integration of systems and applications across the enterprise. The Patterns Explorer tab in the Broker Application Development perspective includes an Application Integration Pattern that describes the problem and gives selection guidance and solution for this type of pattern specification, for example:

The Problem: Different types of enterprise applications typically cannot communicate with one another to efficiently share data.

Selection guidance: Use this pattern when:

- ▶ One or more systems must be connected in a structured way
- ▶ Data passing between the systems requires transforming or routing
- ▶ Data in multiple systems must be maintained and synchronized
- ▶ Connections between systems are likely to change and evolve over time

The Solution: Implement a message flow that mediates between the providing and requesting applications.

The Application Integration Pattern is extended to describe a specific SAP Application Integration Pattern. SAP provides a range of business applications, which includes customer relationship management (CRM), supply chain management (SCM), and product life cycle management (PLM).

The problem is described as SAP Integration enables the integration of SAP systems and applications across the enterprise. The pattern can be used to process various types of IDocs that have a single program identifier without you

being required to redeploy or rediscover existing message sets and adapters, even when you are adding different types of IDocs.

The solution is to implement an IDoc routing message flow that processes IDocs of all types by using the generic pass-through message set. Partial parsing allows the IDoc routing message flow to determine the IDoc type and put the IDoc, as a bit stream, on an appropriate WebSphere MQ queue. A single message flow is required for each IDoc type that gets this message and processes the IDoc as though the flow started with a SAPInput node. To implement these message flows, you must run the Adapter Connection wizard for the IDoc that the message flow is going to process, and then you must use the DataObject domain and set the message format to IDoc. The adapter component that is created by the Adapter Connection wizard is not required when you are using the message set with an MQInput node. Subsequent IDoc types can be supported by creating a new message flow and a new queue.

Note: For additional information about how to implement the IDoc routing message flow, refer to topic “Routing IDocs to separate message flows” in the WebSphere Message Broker V7.0 Information Center:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/bc22060_.htm

6.8 WebSphere Message Broker scenario environment

We developed the WebSphere Message Broker scenarios in a Windows environment using the following components:

- ▶ WebSphere Message Broker 7.0.0.0
- ▶ WebSphere Message Broker Toolkit 7.0.0
- ▶ WebSphere MQ 7.0.1.0
- ▶ WebSphere MQ Explorer 7.0.0
- ▶ SAP ERP 2005 ECC 6.0 instance using a Unicode enabled SAP kernel on a DB2® v8.1 database at Fixpack 11. The SAP instance is running at BASIS and ABAP Support Package 7
- ▶ SAP GUI for Windows, version 7.10 Patch Level 14
- ▶ Siebel version 8.0

6.9 Scenarios

The first scenario describes how you configure a WebSphere Message Broker to communicate with an SAP Server using the SAPInput node.

The following SAP components, which are needed for the interaction with WebSphere Message Broker in the Inbound scenario, were configured on the SAP server:

- ▶ RFC Destination with a connection type of T (TCP/IP Connection) and RFC program ID of BETA01
- ▶ Logical systems:
 - CLNT001 for client 001
 - BETA01
- ▶ Distribution Model configures the flow between the WebSphere Message Broker adapter and the SAP system:
 - The message type of the distribution model is a MATMAS (material master) IDoc
 - Sending partner: CLNT001
 - Receiving partner: BETA01
- ▶ Partner Profile:
 - Partner System BETA01:
 - Port A000000093
 - Message type MATMAS MATMAS05
 - Partner System CLNT001

Note: For instructions about how to configure the SAP server to work with the SAP adapter, refer to the topic *Configuring the SAP server to work with the adapter* in the WebSphere Message Broker V7.0 Information Center:

<http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp>

The second scenario uses the SAPRequest node and describes an SAP outbound operation that makes use of a built-in function (BAPI).

The third scenario describes a master data synchronization between an SAP EIS system and a Siebel EIS system. We use the SAPReply node to illustrate the SAP adapter support for inbound processing with the Synchronous callback interface (SCI).

6.9.1 Scenario 1: SAP Inbound

The objective of this scenario is to demonstrate the functionality of the SAPInput node. We build a simple message flow that receives an IDoc from an SAP transaction into a MQOutput queue.

The broker runtime for this scenario consists of a broker called MQV7BROKER, which is connected to a queue manager called MQV7QMGR.

We first enable the broker runtime to work with the SAP system. WebSphere Message Broker V7.0 supports SAP Java Connector (SAP JCo) version 3.0.1; however, it does not support SAP JCo version 2.

Step 1: Preparing the environment for WebSphere Adapters nodes

The sapjco3.jar JAR and sapjco3.dll native library files were downloaded and saved to a directory on the local broker machine, in this case, C:\AdapterFiles\SAPFiles.

We are going to use the IBM WebSphere MQ Explorer to generate and issue the mqsichangeproperties commands to set up the dependencies:

1. Open the IBM WebSphere MQ Explorer.
2. In the MQ Explorer, locate the Navigator panel, right-click the **Configurable Services** folder for the MQV7BROKER, and select **Show IBM Predefined Templates**. A + sign is added to the Configurable Services folder.
3. Click the + sign beside Configurable Services to expand the services, as shown in Figure 6-1 on page 156.

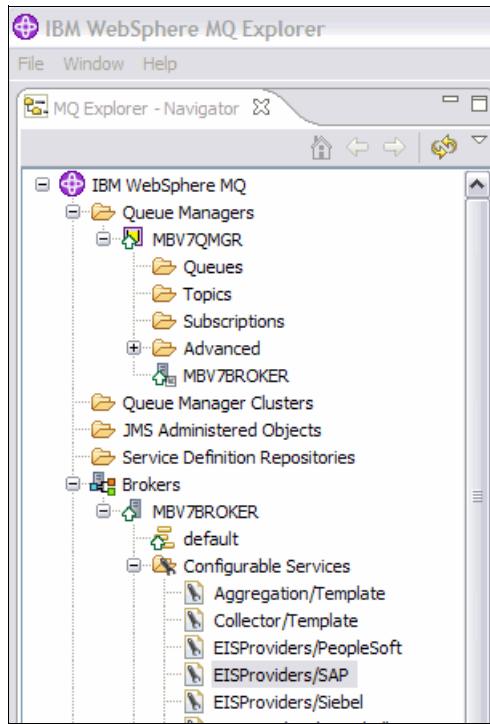


Figure 6-1 The Expanded Configurable Services folder

4. Right-click **EISProviders/SAP**, and select **Properties**. A panel opens, which displays both jarsURL and nativeLibs with the default values.
5. In the Value column for each of the jarsURL and nativeLibs Keys, type C:\AdapterFiles\SAPFiles, as shown in Figure 6-2 on page 157, and click **Finish**. The panel displays the mqsichangeproperties command that is submitted to the broker.

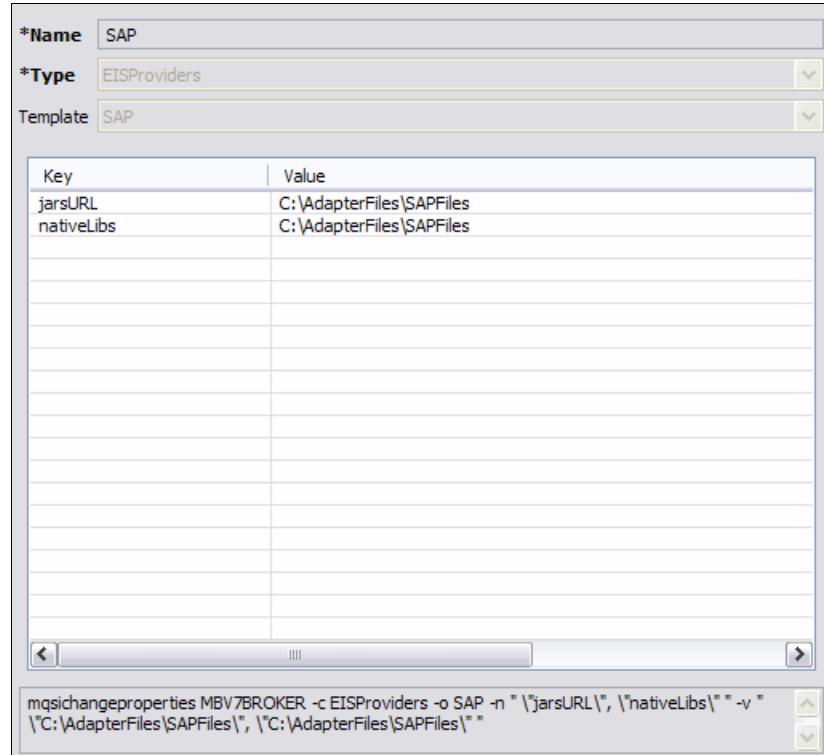


Figure 6-2 *mqsischangeproperties* command

6. The Broker Administration log should contain a BIP2880I message for each of the mqsischangeproperties commands, which indicates that they were successfully completed, as shown in Figure 6-3.

	Message	Source	Timestamp	Message Detail
i	BIP2880I	Change Notification	Nov 24, 2009 2:18:41 PM EST	The property 'EISProviders/SAP'
i	BIP2880I	Change Notification	Nov 24, 2009 2:18:41 PM EST	The property 'EISProviders/SAP'

Figure 6-3 *Broker Administration log*

7. Stop and restart the Broker for the changes to take effect.

Step 2: Discovering the EIS objects and services

To discover the EIS objects and services:

1. Open the WebSphere Message Broker Toolkit, and click the **Quick Starts** link in the Projects section of the Broker Development perspective.

2. From the Adapter Connection, click **Start**, and select **IBM WebSphere Adapter for SAP Software with transaction support (IBM: 7.0.0.0)**. Click **Next**.
3. Leave the default value (CWYAP_SAPAdapter_Tx), and click **Next**.
4. Specify the location of the required SAP sapjco3.jar and System library files, as shown in Figure 6-4. For each option, click **Browse**, and navigate to C:\AdapterFiles\SAPFiles. Select the appropriate file, and click **Open**. Click **Next**.

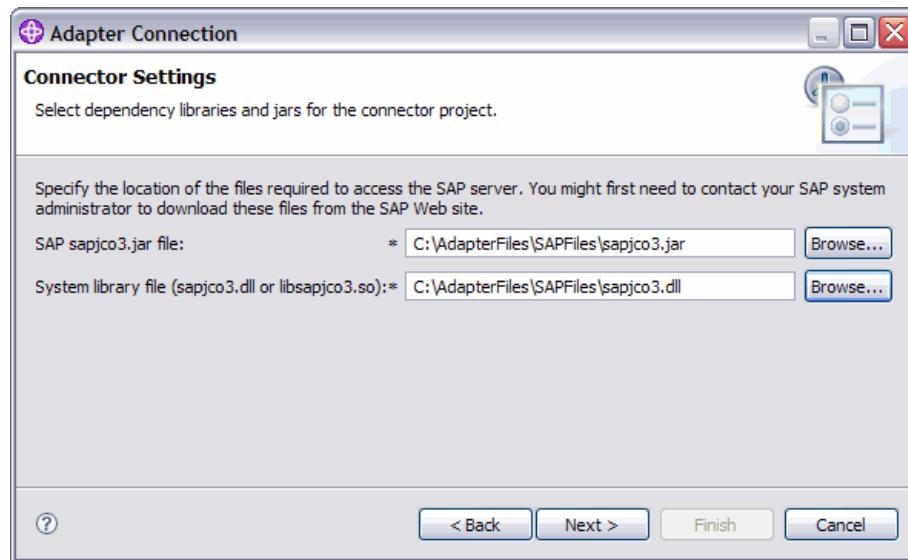


Figure 6-4 Adapter Connection wizard: Connector Settings

5. Because we intend to receive an IDoc from SAP, we must select **Inbound**, as shown in Figure 6-5 on page 159. Click **Next**.

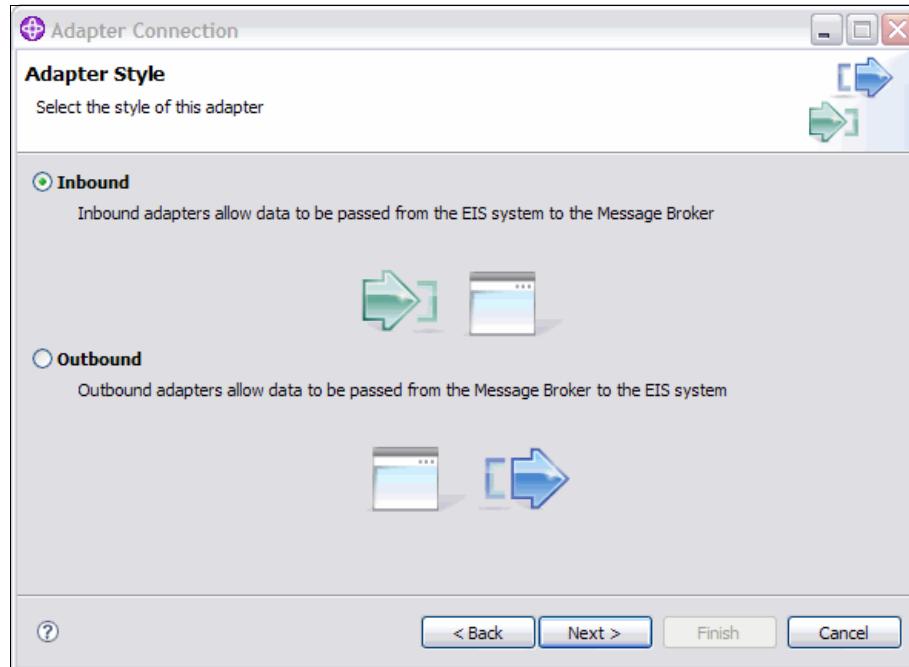


Figure 6-5 Adapter Connection wizard: Inbound

6. In the Figure 6-6 on page 160, we configure the Discovery Agent. This agent is used to search inside SAP for the business object that we want to transform into Message Definition files inside a Message Set. Complete the Host name, System number, Client, User name, Password, and SAP interface name fields with the appropriate values. Leave the default values in the remaining fields. Click **Next**.

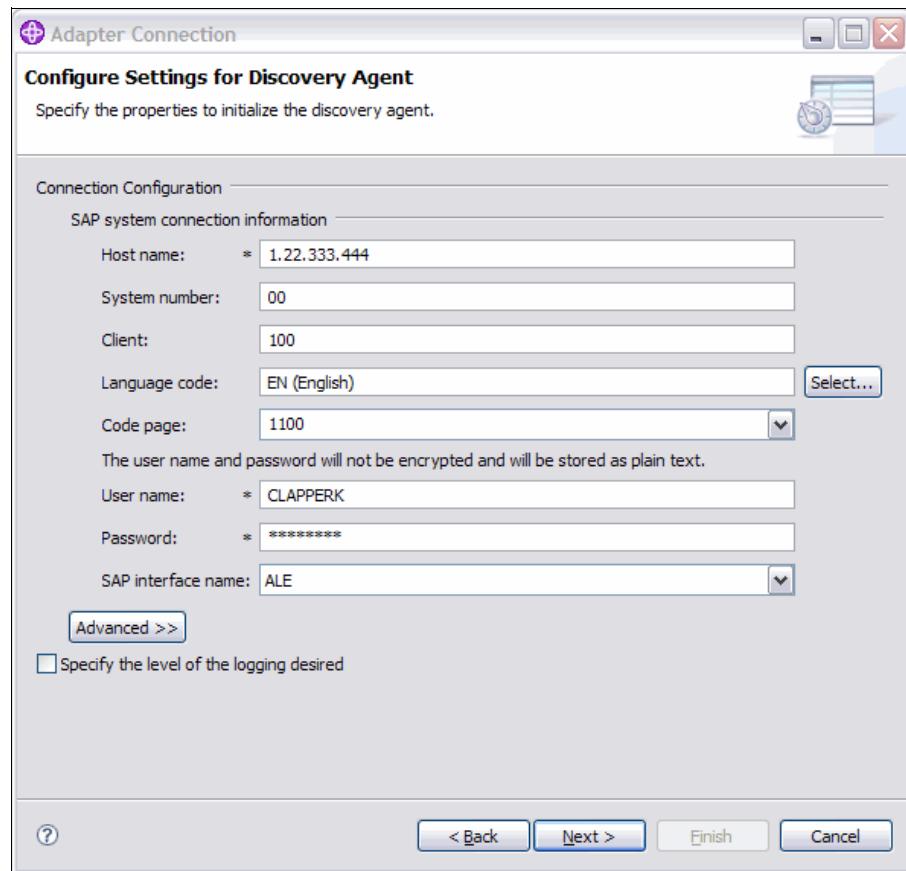


Figure 6-6 Adapter Connection wizard: Configure Settings for Discovery Agent

7. In the Find and Discover Services panel, expand the ALE menu by clicking the + sign. Select **Discover IDoc From System**, but do not expand it at this point. We must first define a query.
8. In the top-left corner of the Objects discovered by query pane, click **Create or edit filter** to define your search criteria.
9. We are going to look for the IDoc type, MATMAS05, which was the message type defined on the SAP system for the BETA01 Receiving Partner System. Type MATMAS* in the Find objects with this pattern: field, and click **OK**, as shown in Figure 6-7 on page 161.

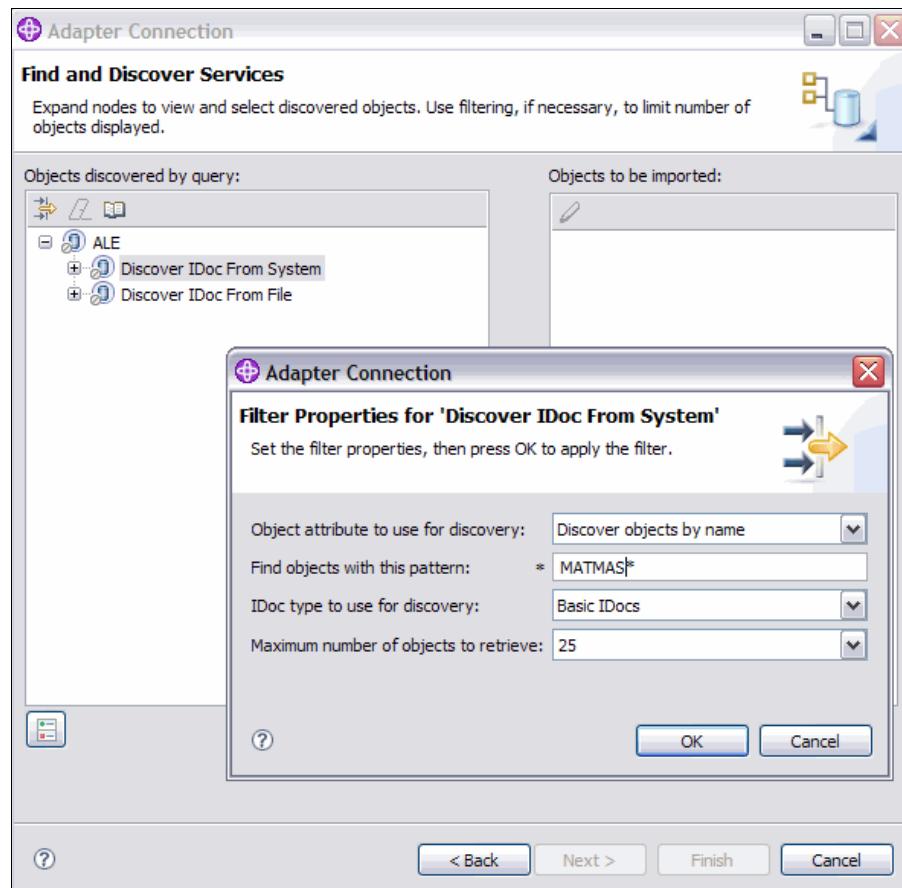


Figure 6-7 Adapter Connection wizard: IDOC lookup

10. Now expand the Discover IDoc From System (filtered) by clicking the + sign next to it. The IDocs matching the specified criteria are shown.
11. Click **MATMAS05**, and move it to Objects to be imported by clicking in the icon with the > sign on the center of the window, as shown in Figure 6-8 on page 162.

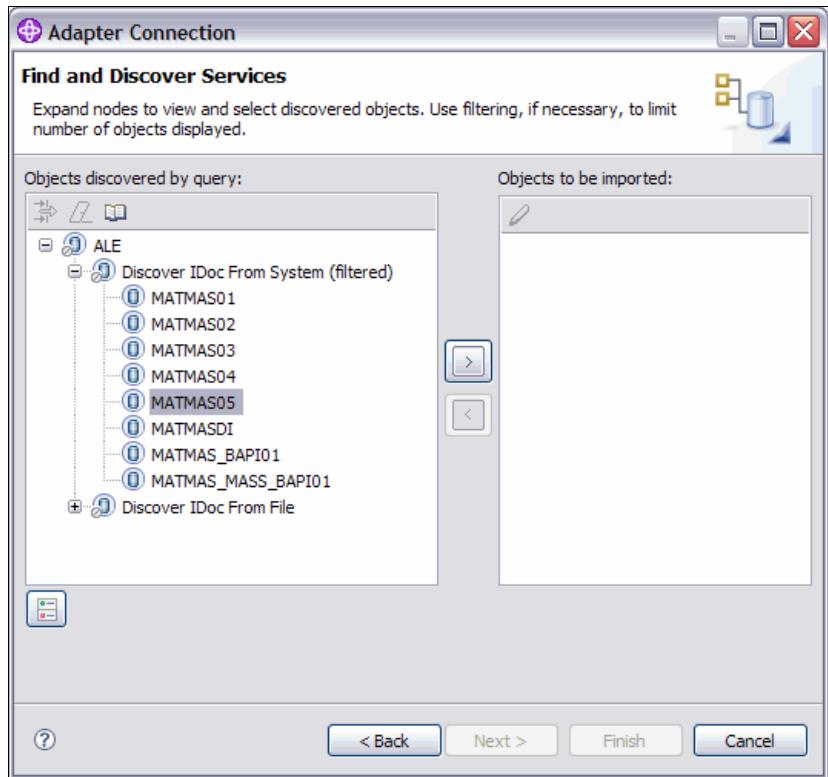


Figure 6-8 Adapter Connection wizard: Import MATMAS05 IDOC

12. On Figure 6-9 on page 163, select the following options, and click **OK**.
- Use SAP field names to generate attribute names for Control Record.
 - Use SAP-original casing for Control Record business object attribute names.
 - Use SAP field names to generate attribute names for Data Record.
 - Use SAP-original casing for Data Record business object attribute names.

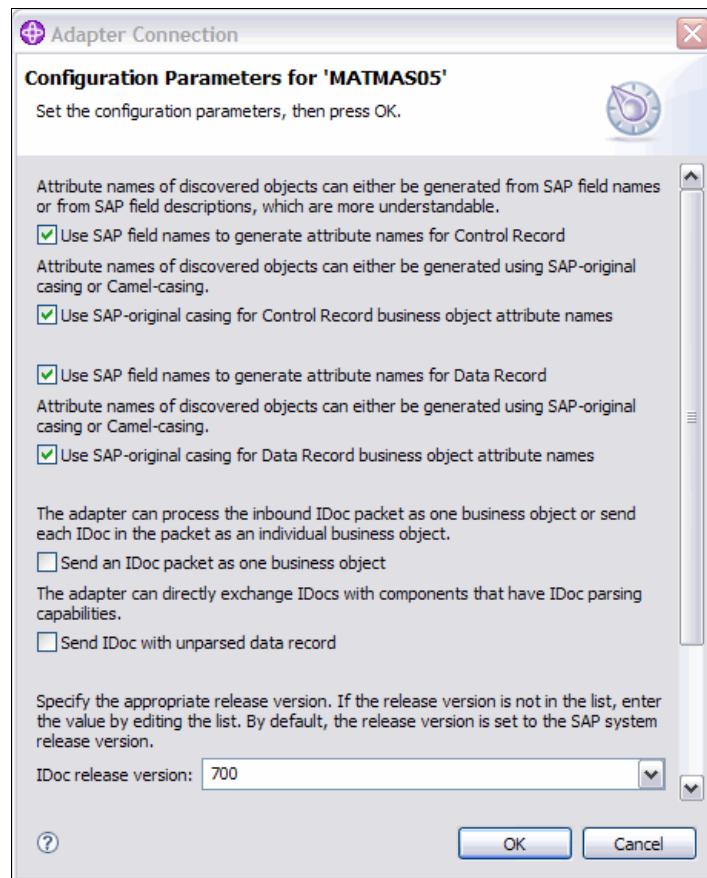


Figure 6-9 Adapter Connection wizard: Configure MATMAS05

13. In the Find and Discover Services panel, click **Next**.
14. From the pull-down list, select **Create** in the Service operation:* field. Click **Add**.
15. In the Add Value panel, Figure 6-10 on page 164, select the value. For our purposes, we selected:
`MessageType=MATMAS;MessageCode=;MessageFunction=;`
Click **OK**, and then click **Next**.

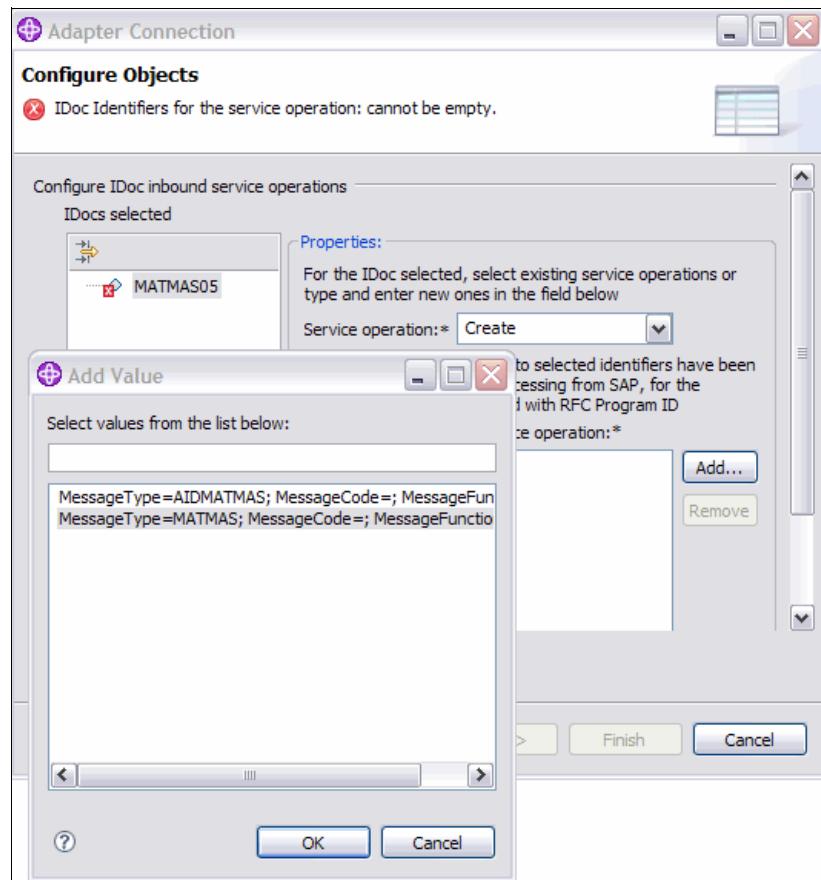


Figure 6-10 Adapter Connection wizard: Select service operation for IDOC

16. Next, we configure the Connection Properties, Figure 6-11 on page 165, which the broker uses at runtime to connect to the SAP instance. We set the Inbound Connection Properties with the SAP system values and the RFC program ID already created on the SAP system.

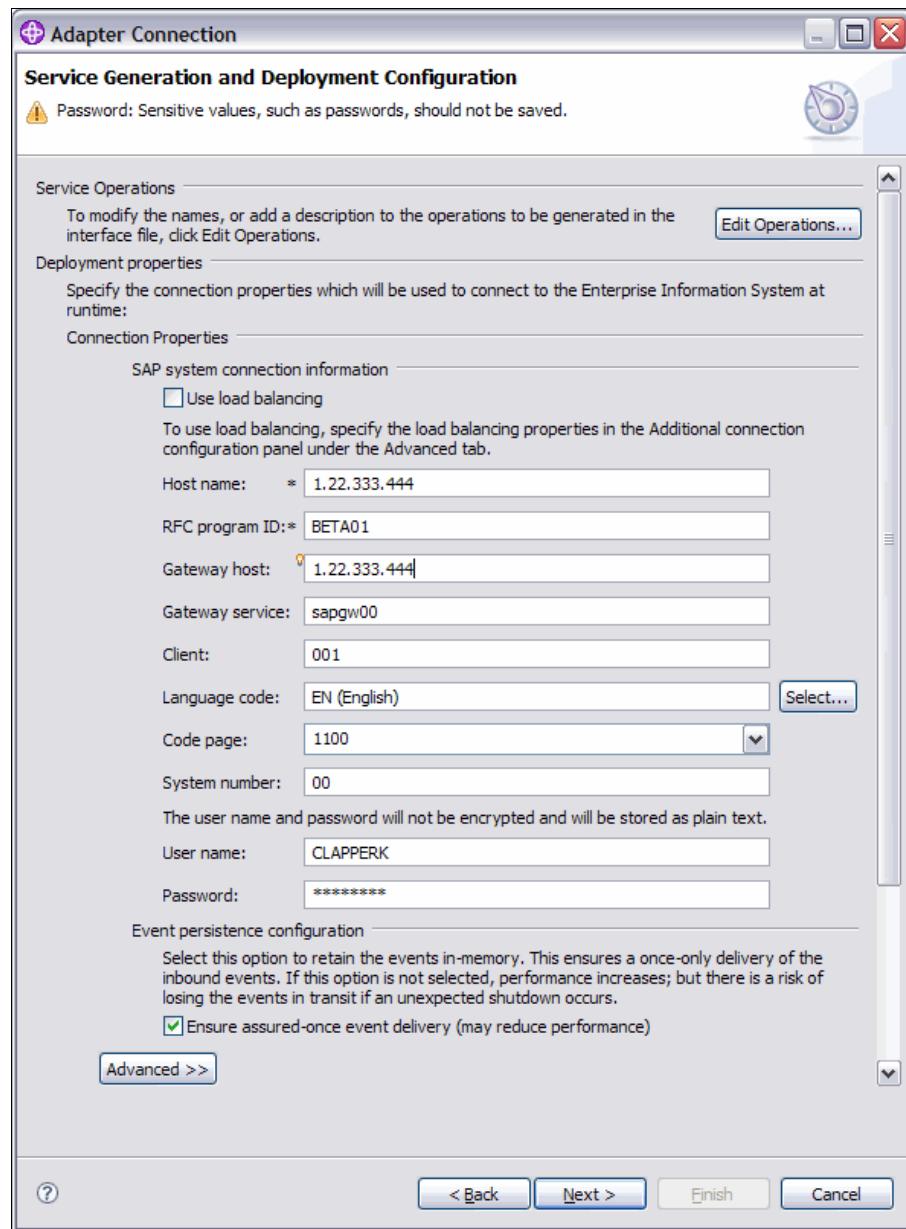


Figure 6-11 Adapter Connection wizard: Configure connection properties

17. In the Publishing Properties panel, Figure 6-12, we provide the names for the resources that will be generated:

- In the Message flow project name: * field, enter SAPInbound.
- In the field for the Adapter component name, enter SAPIdocInbound.
- All of the fields, apart from Adapter component name: *, are completed for you.

Click **Finish**.

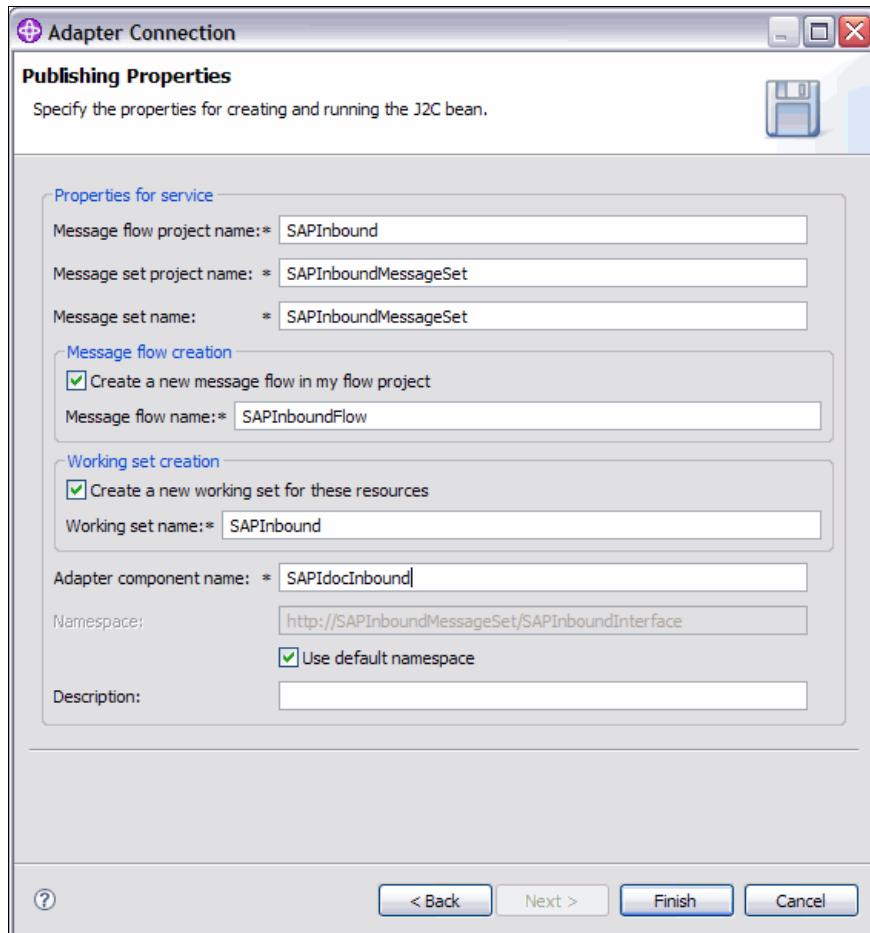


Figure 6-12 Adapter Connection wizard: Publishing properties Inbound

18. The generated resources are now listed in the Projects section of the Broker Development perspective, Figure 6-13, in the WebSphere Message Broker Toolkit, which include:

- Message flow: SAPInboundFlow.msgflow
- Message set: SAPInboundMessageSet
- Adapter component: SAPIdocInbound.inadapter

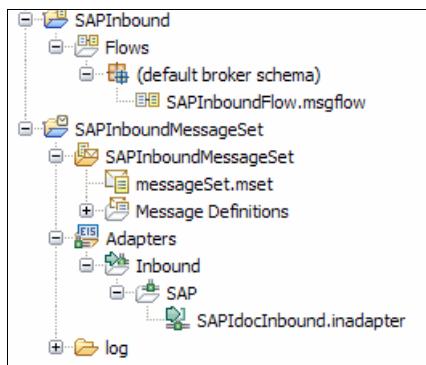


Figure 6-13 The Inbound resources generated by the Adapter Connection wizard

19. The adapter component values can be changed without having to run the Adapter Connection wizard again, as shown in Figure 6-14. Double-click the **SAPIdocInbound.inadapter** component to see the details.

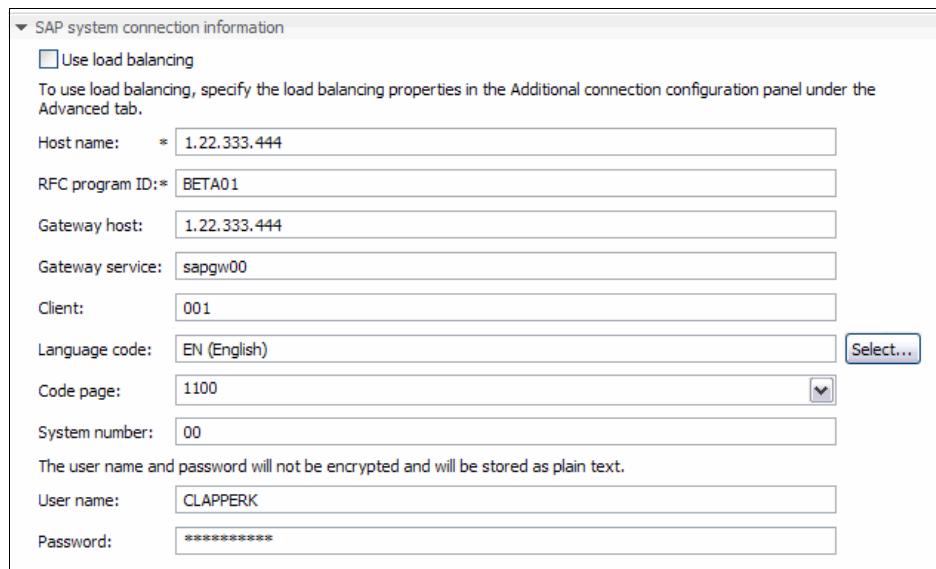


Figure 6-14 SAPIdocInbound.inadapter component details

Step 3: Adding the WebSphere Adapters node into the message flow

To add the WebSphere Adapters node into the message flow:

1. Under the SAPInbound folder, double-click **SAPInboundFlow.msgflow** to open the flow in the flow editor.
2. Click the **SAPIdocInbound.inadapter** component and drag-and-drop it onto the flow editor to create a message flow with a SAPInput node and a subflow, as shown in Figure 6-15.

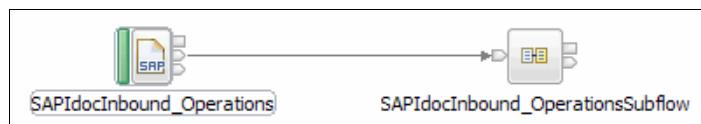


Figure 6-15 SAPInboundFlow message flow

3. The subflow provides a template for building different operations for the IDoc. Double-click the subflow to open it. The operations that were selected during the Adapter Connection wizard process are represented in the subflow, as shown in Figure 6-16.

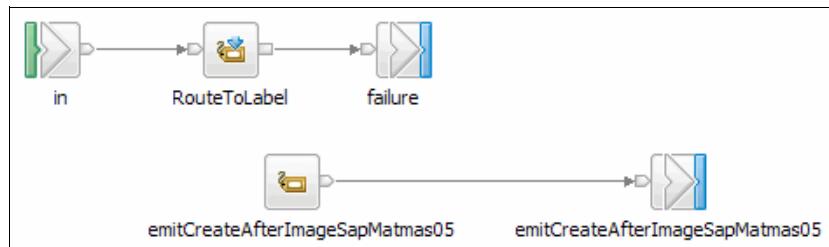


Figure 6-16 SAPInboundFlow message flow: Subflow

4. From the flow editor node palette, select **MQOutput node** and drag-and-drop it onto the main flow.
5. In the Basic tab of the MQOutput Node Properties to SAP_INBOUND_OUT, update the Queue name, as shown in Figure 6-17 on page 169.

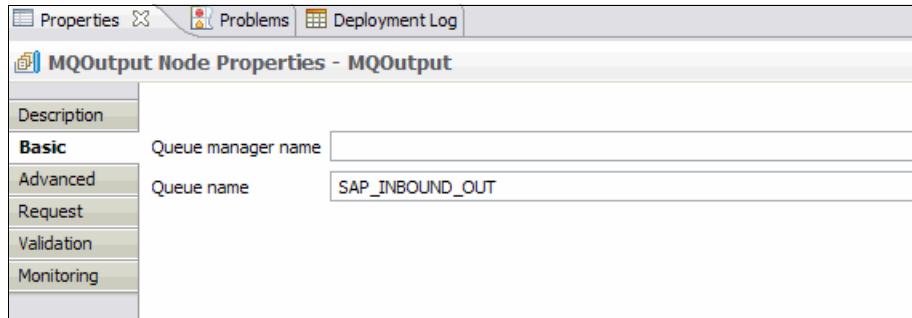


Figure 6-17 MQOutput Queue name property

6. Connect the out terminal of the SAPIdocInbound_OperationsSubflow to the in terminal of the MQOutput node, Figure 6-18, and save the message flow.

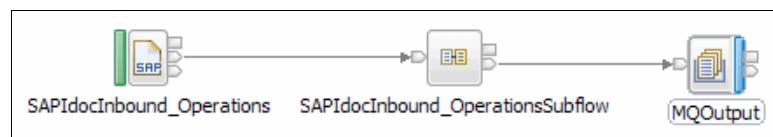


Figure 6-18 SAPInboundFlow message flow with MQOutput node connected

It is now possible to use this simple message flow to receive an IDoc from the SAP system.

Step 4: Creating and deploying the broker archive file to the broker

To create and deploy the BAR file to the broker:

1. Right-click **SAPInbound folder**, and select **New → Message Broker Archive**. In the Name field, type SAPInbound, and click **Finish**. The Prepare BAR file panel opens.
2. Select all of the components that were created for the SAPInbound scenario, and click Build broker archive, as shown in Figure 6-19 on page 170.

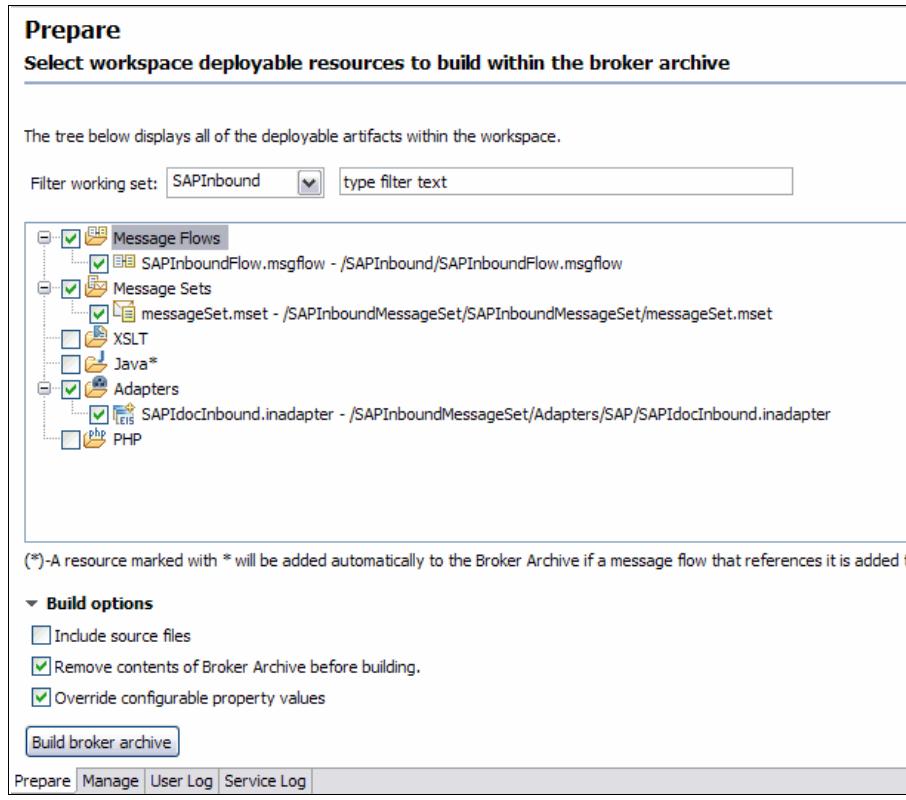


Figure 6-19 Prepare Inbound Broker Archive file

3. When the operation successfully completes, click **OK** on the pop-up message, and save the BAR file.
4. Click the **SAPInbound.bar** file, and drag-and-drop it into the WebSphere MQ Explorer taskbar. When the MQ Explorer opens up, drag-and-drop the BAR file onto the brokers execution group.
5. After the components are successfully deployed to the execution group, Figure 6-20 on page 171, you are ready to test the message flow.

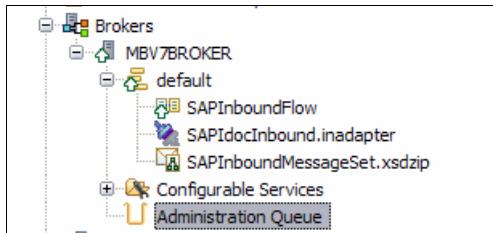


Figure 6-20 Successfully deployed components

Testing the SAPInbound message flow

We use the SAP GUI to send an IDoc type MATMAS05 containing the material information to the Logical system that is defined in the adapter component deployed to the broker. To test the SAPInbound message flow:

1. In the SAP main menu, issue transaction code bd10 (Send Material).
2. In the Material field, type THINGS. In the Logical system field, type BETA01, as shown in Figure 6-21.

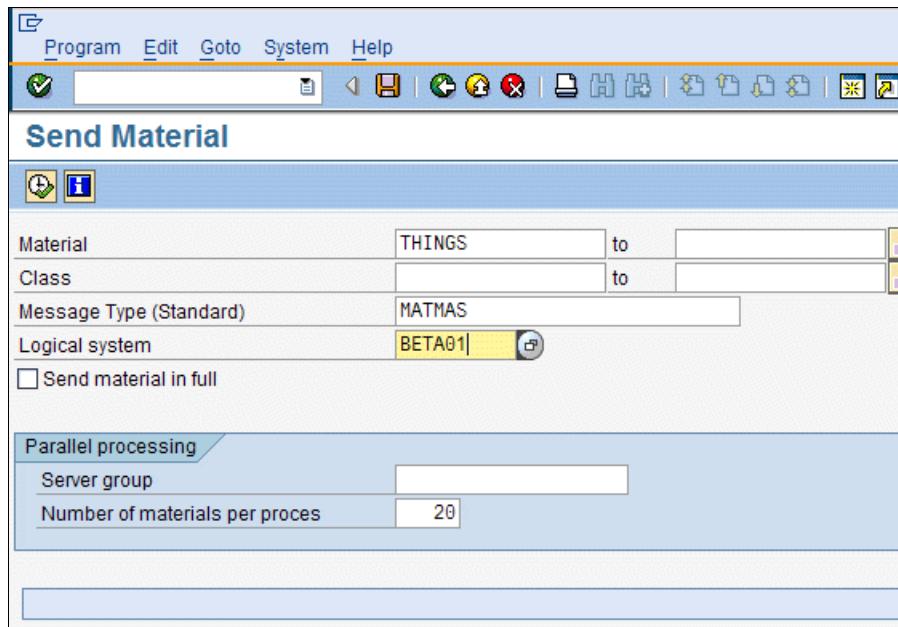


Figure 6-21 SAP Send Material panel

The IDoc MATMAS05 is processed by the SAPInboundFlow and arrives in the SAP_INBOUND_OUT output queue. Example 6-1 on page 172 shows

snippets of the relevant fields taken from the message received on the output queue.

Example 6-1 Snippets of fields from received message on output queue

```
<NS1:SapMatmas05
xmlns:NS1="http://www.ibm.com/xmlns/prod/websphere/j2ca/sap/sapmatmas05">
  <SAPTransactionID>091ECC9F0ECC4B22A0FB001F</SAPTransactionID>
  <SapMatmas05IDocB0>
    <SapIDocControlRecord>
      <RCVPOR>A000000093</RCVPOR>
      <SNDPRT>LS</SNDPRT>
      ...
      <IDOCTYP>MATMAS05</IDOCTYP>
      <SNDPRN>CLNT001</SNDPRN>
      ...
      <TABNAM>EDI_DC40</TABNAM>
      <CREDAT>20091211</CREDAT>
      ...
      <MESTYP>MATMAS</MESTYP>
      <RCVPRN>BETA01</RCVPRN>
      ...
      <SERIAL>2009121114349</SERIAL>
      <SNDPOR>SAPWLI</SNDPOR>
      <DOCNUM>0000000002144034</DOCNUM>
      ...
    </SapIDocControlRecord>
    <SapMatmas05DataRecord>
      ...
  </SapMatmas05IDocB0>
</NS1:SapMatmas05>
```

6.9.2 Scenario 2: SAP Outbound

The objective of this scenario is to demonstrate the functionality of the SAPRequest node. In this scenario, we modify the XML file NewCustomer.xml with a first and last name and put the XML message to an MQInput queue. This message is then mapped to a BAPI structure using a mapping node before being passed to the SAPRequest node.

Figure 6-22 on page 173 shows the contents of the newCustomer XML message file.

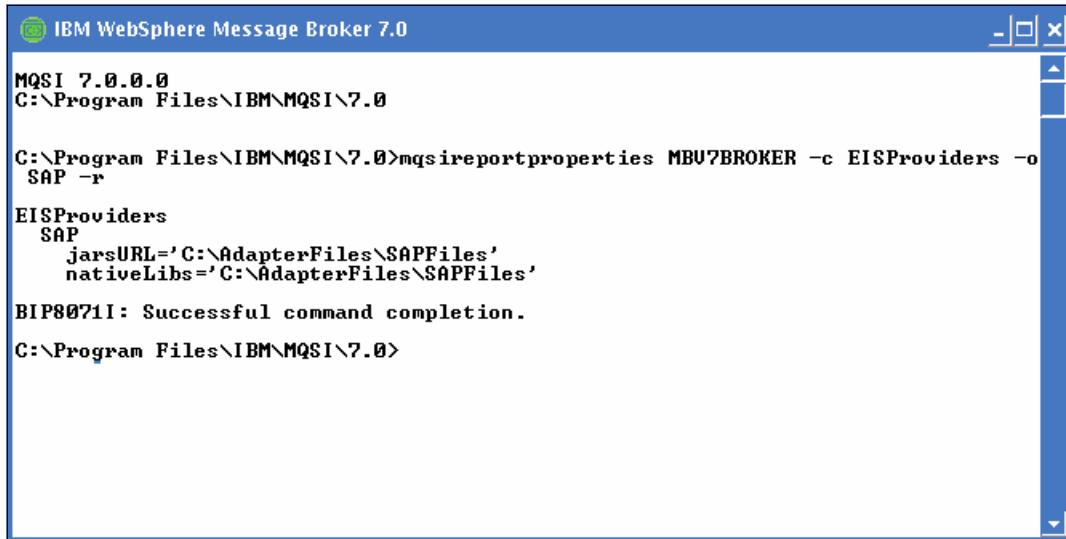
```
<Customer>
  <FirstName>Kirstine</FirstName>
  <LastName>Clapperton</LastName>
  <City>Raleigh</City>
  <PostCode>27713</PostCode>
  <Country>US</Country>
  <Language>US</Language>
  <Currency>USD</Currency>
  <SalesOrg>0001</SalesOrg>
  <Division>01</Division>
  <DistribChannel>01</DistribChannel>
  <Ref_Customer>000000099</Ref_Customer>
</Customer>
```

Figure 6-22 The NewCustomer.xml file

The SAPRequest node executes the BAPI, which results in the creation of a new customer inside the SAP system. The output from this node is the new customer number that is generated. We then verify that the customer number in SAP returns the name that we provided.

Step 1: Preparing the environment for WebSphere Adapters nodes

The environment that we prepared for the WebSphere Adapters nodes in scenario 1 still exists. The **mqsi reportproperties** command confirms the current values that are set for the jarsURL and nativeLibs parameters of the SAP EISProviders object, as shown in Figure 6-23 on page 174.



IBM WebSphere Message Broker 7.0

```
MQSI 7.0.0.0
C:\Program Files\IBM\MQSI\7.0

C:\Program Files\IBM\MQSI\7.0>mqsi reportproperties MBV7BROKER -c EISProviders -o SAP -r

EISProviders
SAP
    jarsURL='C:\AdapterFiles\SAPFiles'
    nativeLibs='C:\AdapterFiles\SAPFiles'

BIP8071I: Successful command completion.

C:\Program Files\IBM\MQSI\7.0>
```

Figure 6-23 *mqsi reportproperties MBV7BROKER -c EISProviders -o SAP -r*

Step 2: Discovering the EIS objects and services

To discover the EIS objects and services:

1. Open the WebSphere Message Broker Toolkit, and in the Projects section of the Broker Development perspective, click **Quick Starts**.
2. From adapter connection, click **Start**. Expand the components under CWYAP_SAPAdapter_Tx, and select the previous connection settings that we configured in the previous scenario, as shown in Figure 6-24 on page 175. Click **Next**.

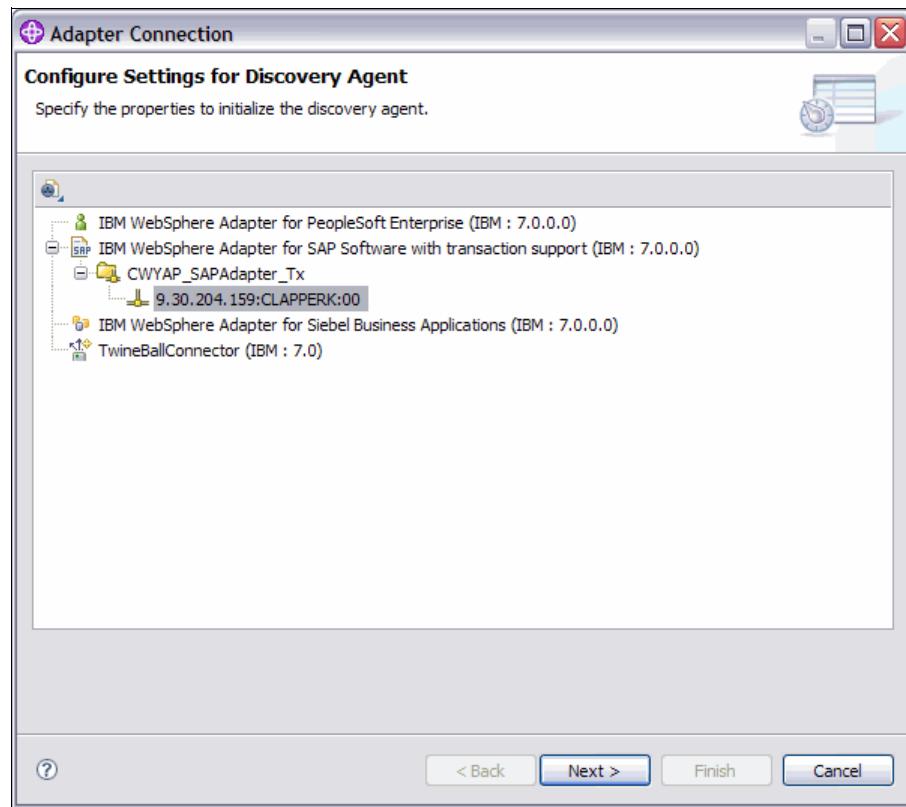


Figure 6-24 Adapter Connection wizard: Configure Settings for Discovery Agent

3. Select **Outbound**, and click **Next**, as shown in Figure 6-25 on page 176.

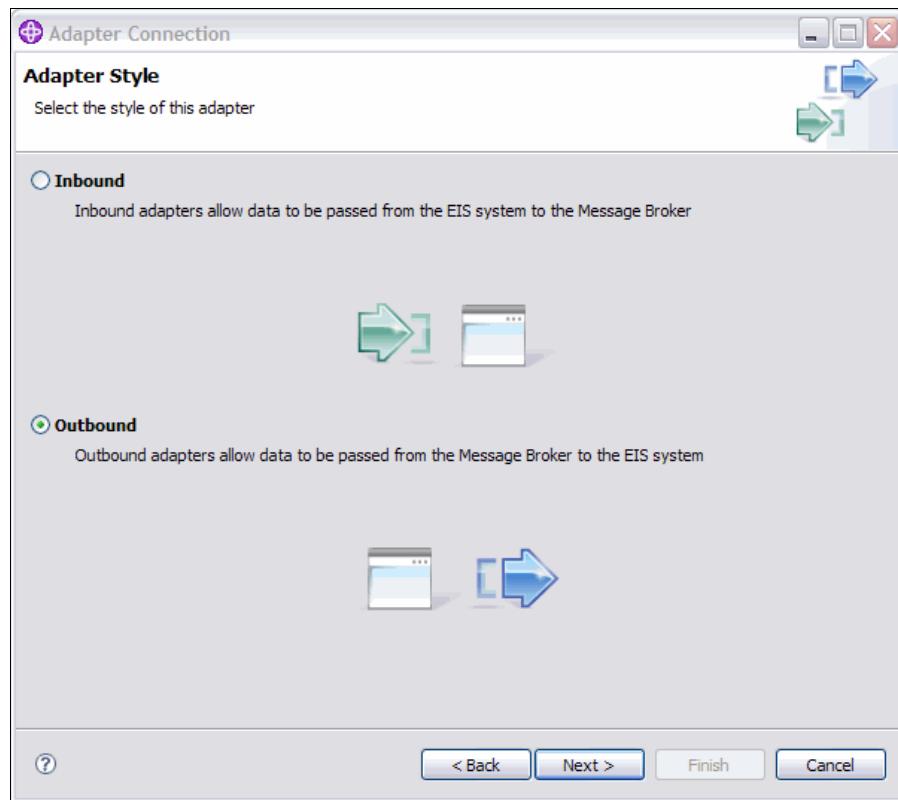


Figure 6-25 Adapter Connection wizard: Outbound

4. We are going to use the Discovery Agent to get the business object from SAP to build our Message Set. Most of the fields are already completed. You should only need to enter the password for the specified user, and select BAPI from the pull-down list for the SAP interface name field, as shown in Figure 6-26 on page 177. Click **Next**.

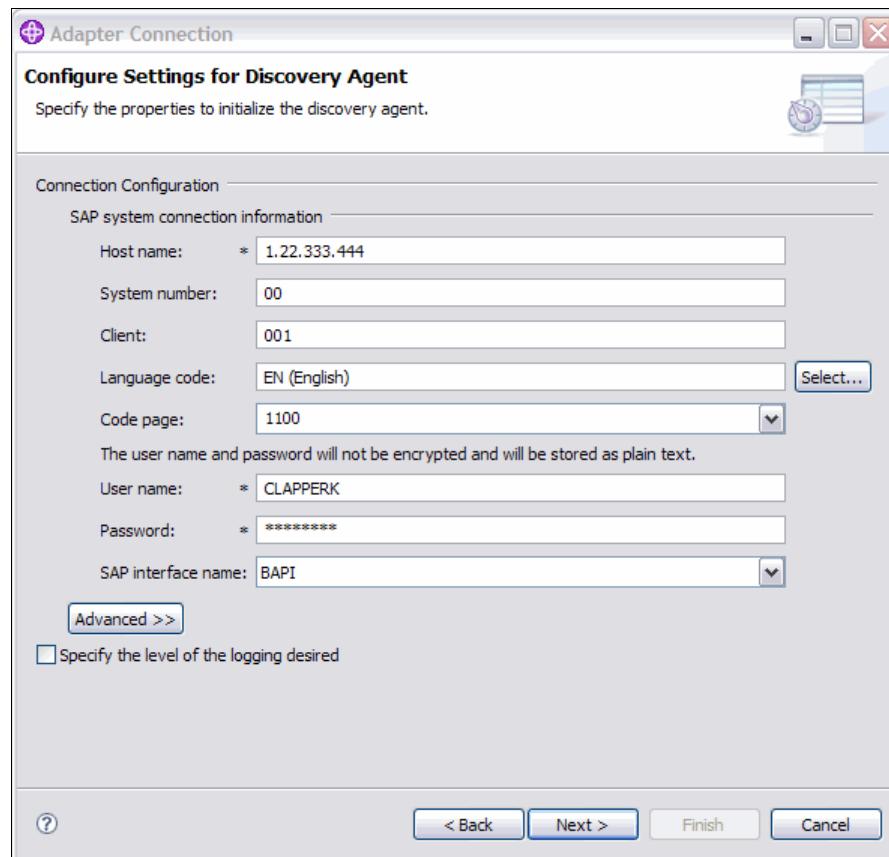


Figure 6-26 Adapter Connection wizard: Configure connection

5. In the Find and Discover Services panel, click **RFC** to highlight it. Do not expand RFC at this point because we must first define a query.
6. To enable you to define your search criteria, in the top-left corner of the Objects discovered by the query panel, click **Create or edit filter**.
7. This scenario is going to use a BAPI called BAPI_CUSTOMER_CREATEFROMDATA1, so enter *CREATEFROMDATA* in the Find objects with this pattern field, as shown in Figure 6-27 on page 178, and click **OK**.

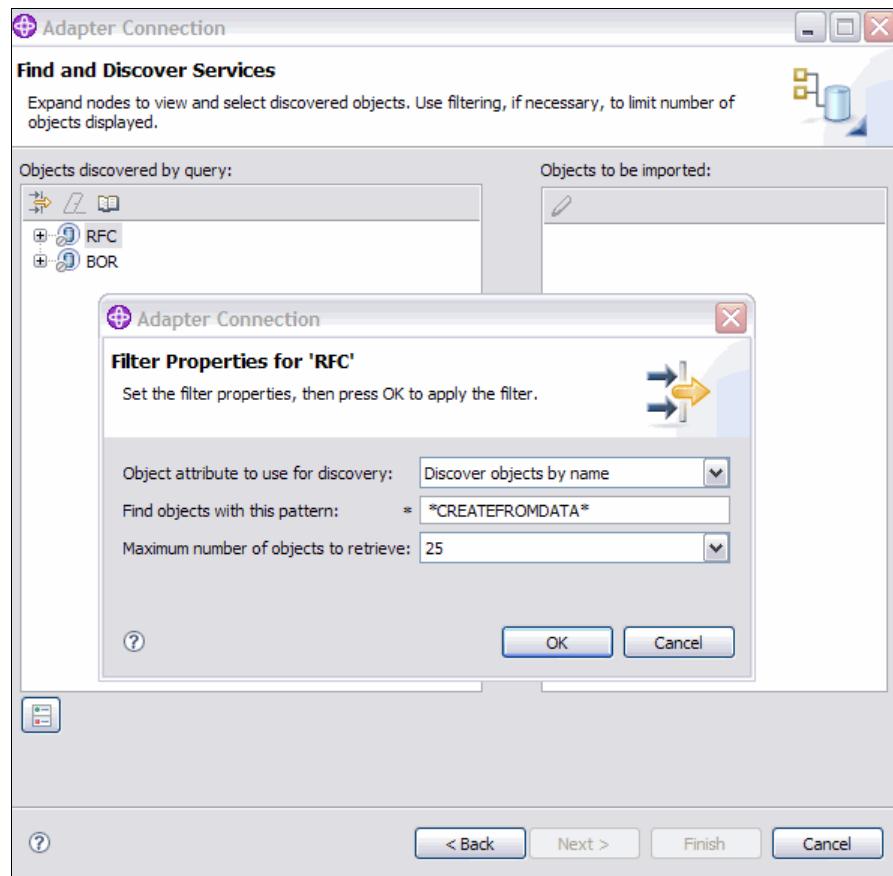


Figure 6-27 Adapter Connection wizard: Filter properties for RFC

8. Expand the components under RFC (filtered) by clicking the + sign. Select **BAPI_CUSTOMER_CREATEFROMDATA1**, and click the > sign to add the selected object to the right panel.
9. Select **Use SAP field names** and **Use the original character case of SAP field names**, as shown in Figure 6-28 on page 179, and click **OK**.

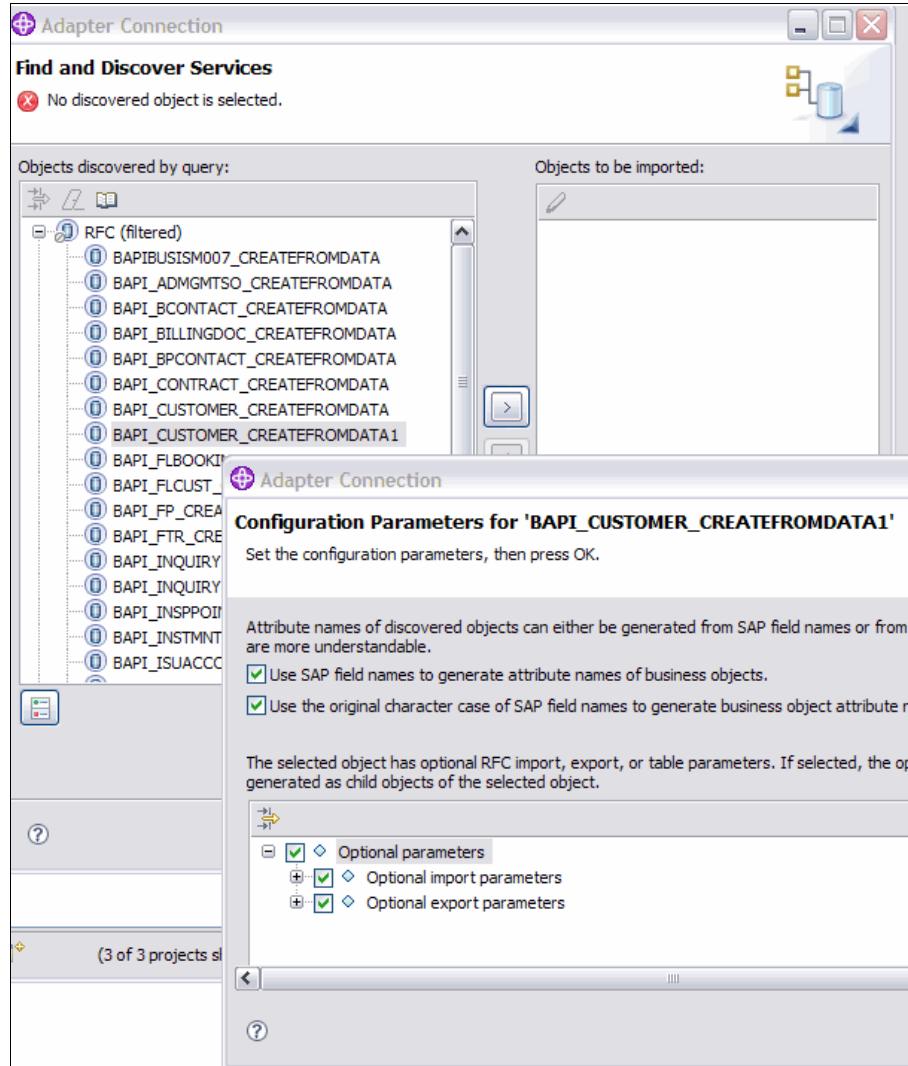


Figure 6-28 Adapter Connection wizard: Configure parameters for BAPI

10. After BAPI_CUSTOMER_CREATEFROMDATA1 is added to the Objects to be imported, click **Next**.
11. Select the option for generating BAPI business objects within a wrapper, and type Customer in the Business object name for the service operations option. Click **Add**.
12. Select **Create**, and click **OK**, as shown in Figure 6-29 on page 180. Click **Next**.

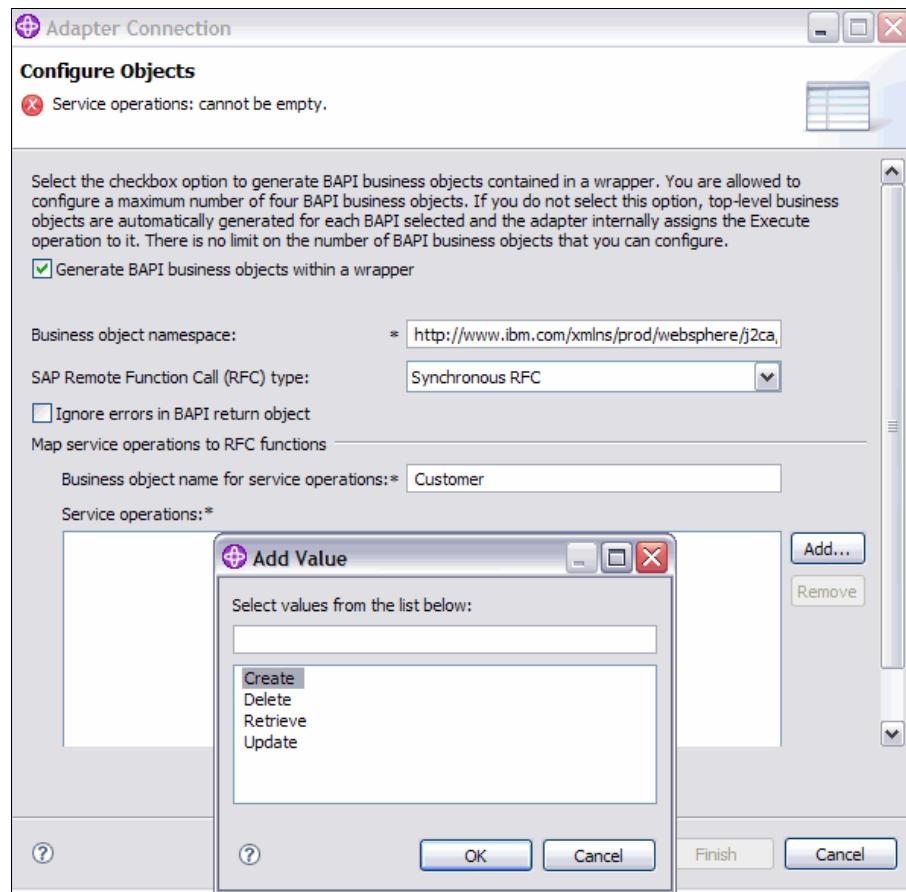


Figure 6-29 Adapter Connection wizard: Add service operations

13. We now must configure the adapter connection properties for the broker runtime. Most of the values should already be provided. You should only need to enter the password for the specified user, as shown in Figure 6-30 on page 181. Click **Next**.

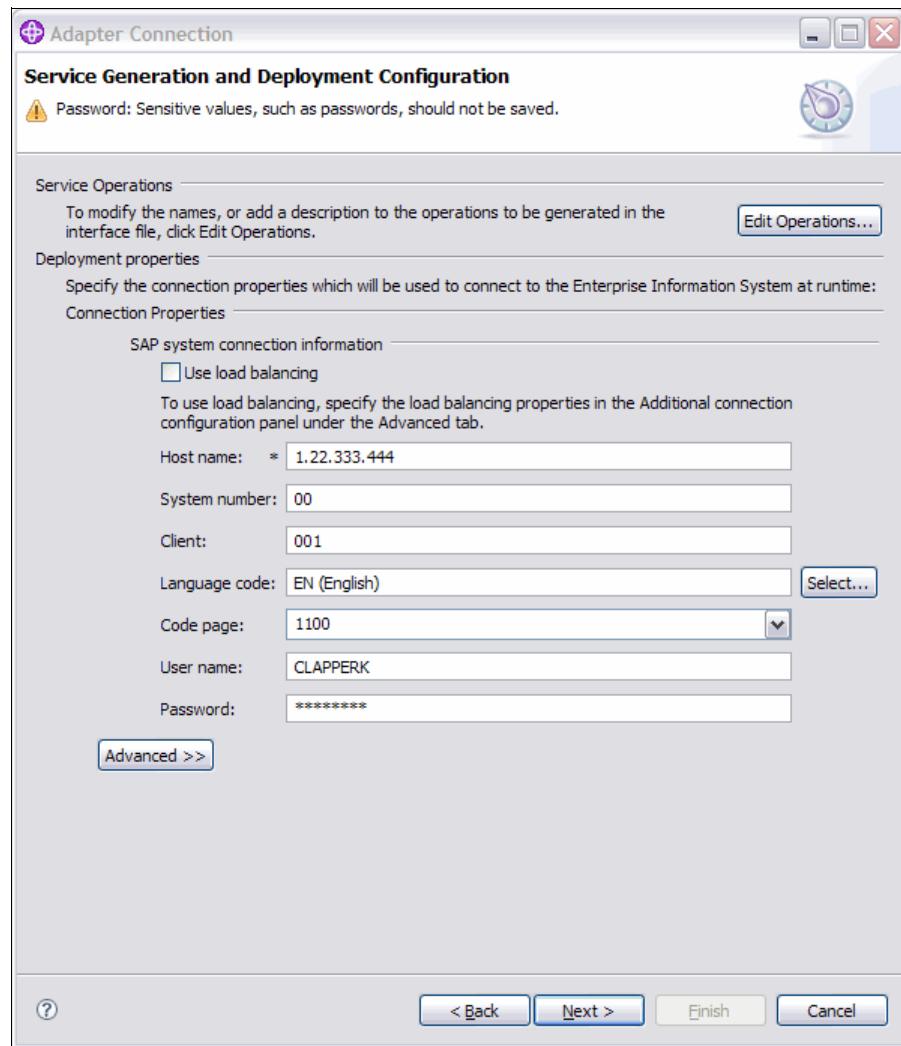


Figure 6-30 Adapter Connection wizard: Configure broker runtime connection

14. In the Publishing Properties panel, we provide the names for the resources that will be generated. In the Message flow project name: * field, type SAP0utbound. All of the other fields, apart from Adapter component name: *, are completed for you. Type SAPBAPI0utbound in the field for the Adapter component name, and click **Finish**, as shown in Figure 6-31 on page 182.

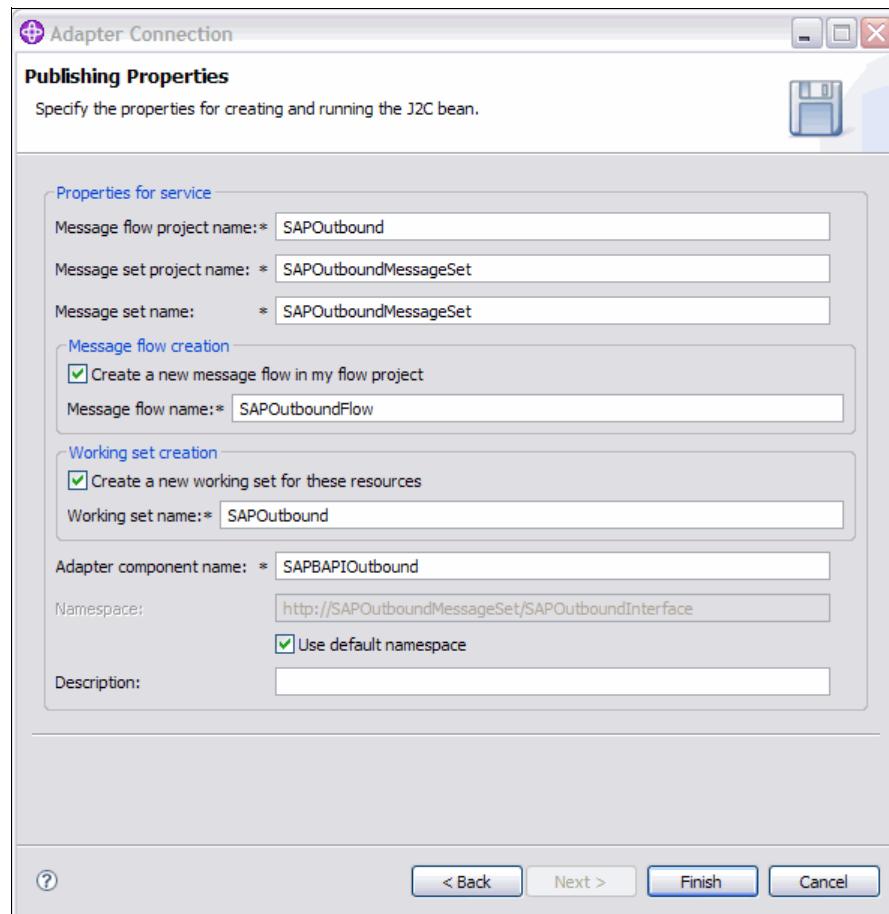


Figure 6-31 Adapter Connection wizard: Publishing properties Outbound

The generated resources should now be listed in the Projects section of the Broker Development perspective in the WebSphere Message Broker Toolkit, as shown in Figure 6-32 on page 183. These should include the following:

- Message flow: SAPOutboundFlow.msgflow
- Message set: SAPOutboundMessageSet
- Adapter component: SAPBAPIOOutbound.outadapter

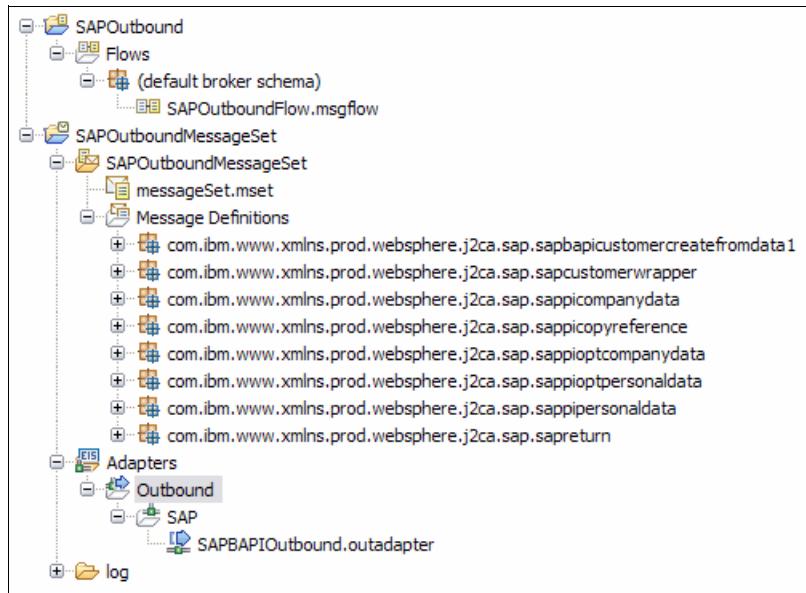


Figure 6-32 The Outbound resources generated by the Adapter Connection wizard

15. You can change the adapter component values without running the Adapter Connection wizard again. Double-click the **SAPBAPIOOutbound.outadapter** component to see the details, as shown in Figure 6-33.

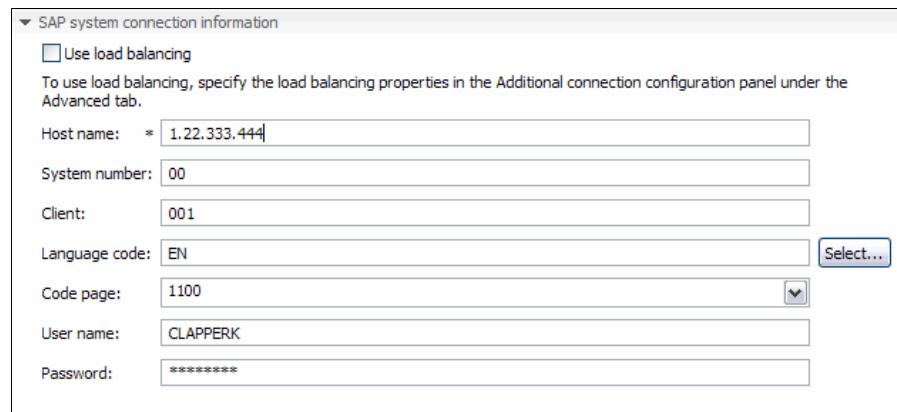


Figure 6-33 SAPBAPIOOutbound.outadapter component details

Step 3: Adding the WebSphere Adapters node into the message flow

To add the WebSphere Adapters node into the message flow:

1. We now have the format for the SAP request. We must create the input format from the NewCustomer.xml input file that we are going to use to test the flow. In the Projects panel of the WebSphere Message Broker Toolkit, right-click **SAPOutboundMessageSet**, and select **Import**.
2. Select **General → File System**, and click **Next**.
3. Click **Browse**, and select the directory path to the location of the NewCustomer.xml file. Select the file, as shown in Figure 6-34. Click **Finish**.

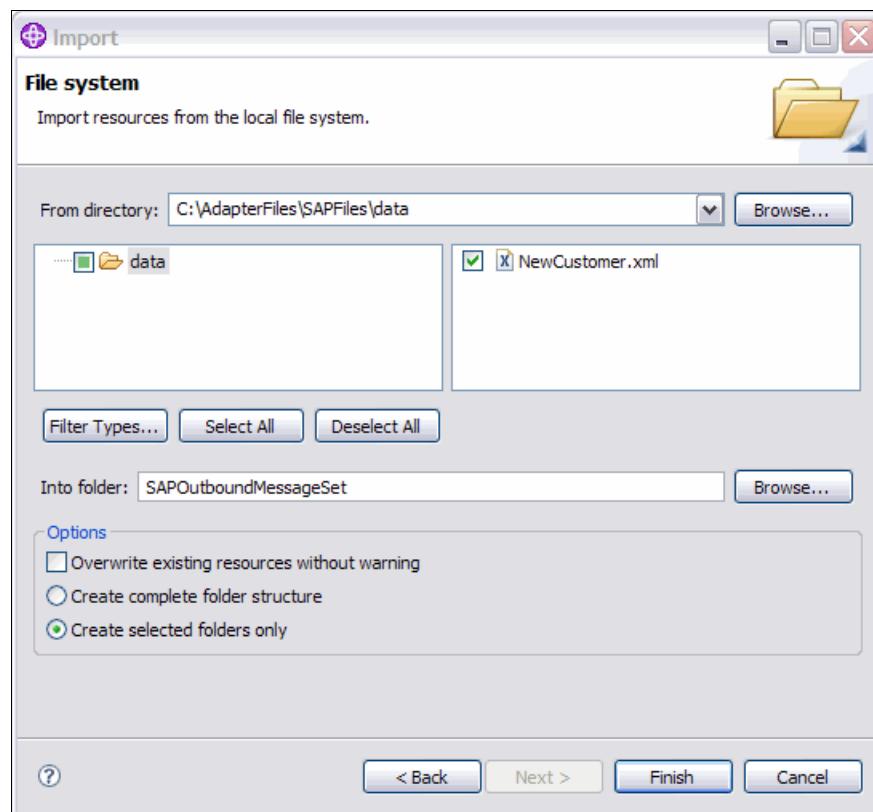


Figure 6-34 Select the NewCustomer.xml input file

4. Right-click the **NewCustomer.xml** file that was just imported into the Projects panel of the WebSphere Message Broker Toolkit. Select **Generate → XML Schema**.

5. If the SAPOutboundMessageSet is not already highlighted, select it, and click **Finish** to generate the schema, as shown in Figure 6-35.



Figure 6-35 Generate the new XML schema

6. Click **OK** to close the XML Schema Generated window.
7. Right-click the newly generated newCustomer.xsd file under the SAPOutboundMessageSet folder, and select **New → Message Definition File From → XML Schema File**.
8. Click **Next** to create a new message definition file, as shown in Figure 6-36 on page 186.

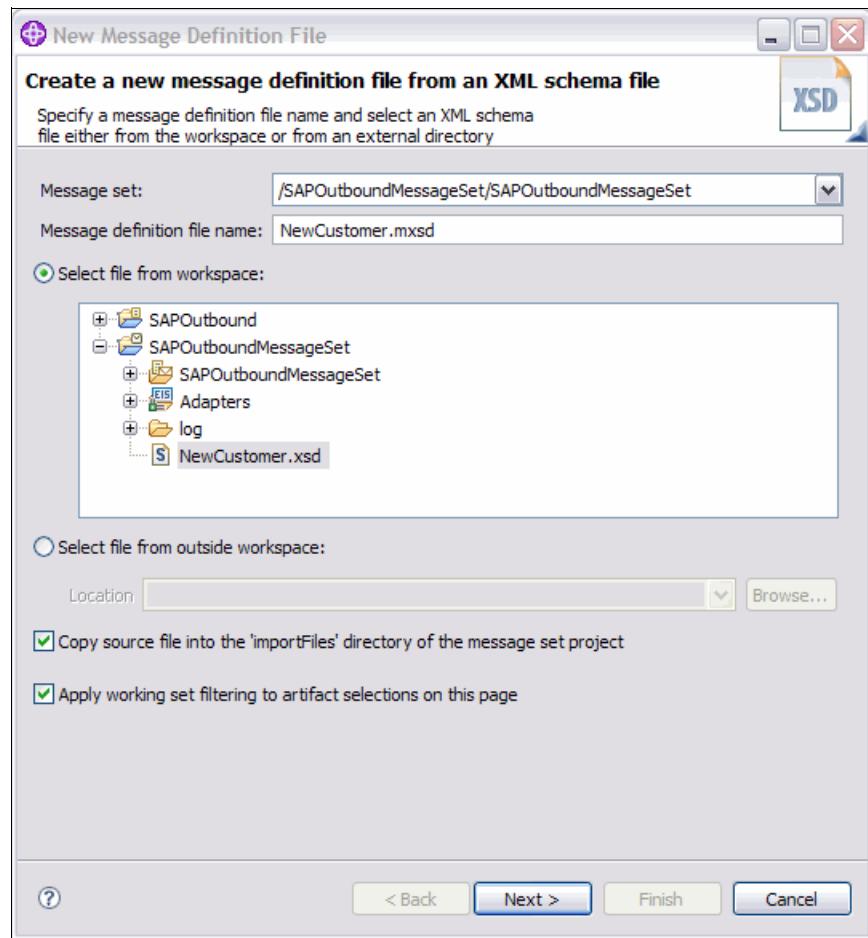


Figure 6-36 Create a new message definition file

9. Click **Select All** so that both the Elements and Customer options are selected, as shown in Figure 6-37 on page 187, and click **Finish**.

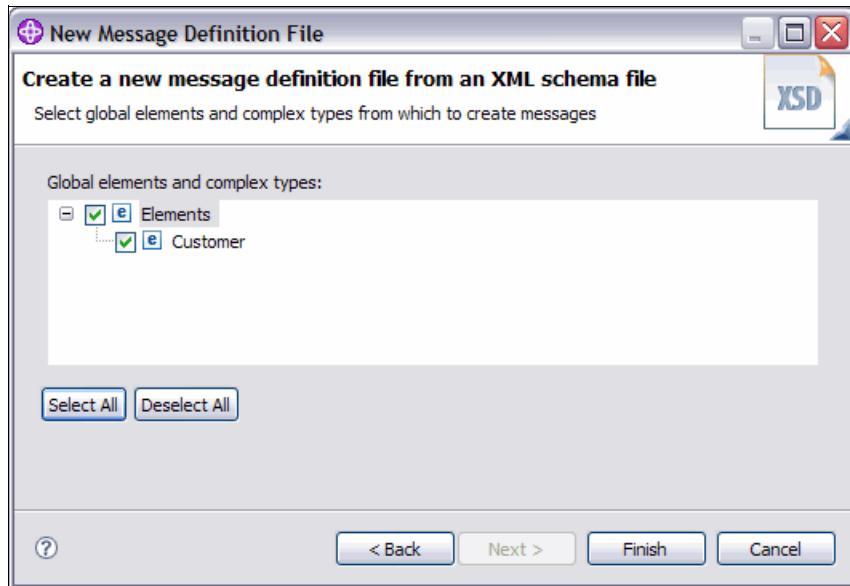


Figure 6-37 Select the global elements and complex types

10. To open the flow in the flow editor, under the SAPOutbound folder, double-click **SAPOutboundFlow.msgflow**.
11. Click **SAPBAPIOutbound.outadapter** component, and drag-and-drop it onto the flow editor. The message flow only contains the SAPRequest node at this point. We must build the rest of the message flow.
12. From the WebSphere MQ folder of the node Palette, select an MQInput node, and drag-and-drop it onto the message flow. Click **MQInput node**, and in the MQInput Node Properties:
 - a. Click the **Basic** tab:
 - i. In the Queue name field, type SAP_OUTBOUND_IN.
 - b. Click the **Input Message Parsing** tab:
 - i. In the Message domain field, select **XMLNSC** from the pull-down list.
 - ii. In the Message set field, select **SAPOutboundMessageSet** from the pull-down list.
13. The next step is to create the mapping between the input format and the BAPI format. We use a Mapping node to achieve this. Select a Mapping node from the Transformation folder of the node Palette, and drag-and-drop it onto the message flow. Double-click **Mapping node** to start the Mapping wizard:
 - a. Expand the Messages folder for both the source and target.

- b. From the source panel, select **Customer**, as shown in Figure 6-38.
- c. From the target panel, Figure 6-38, select **SapCustomerWrapper**. The SapCustomerWrapper contains all of the definitions that are required to call the BAPI.

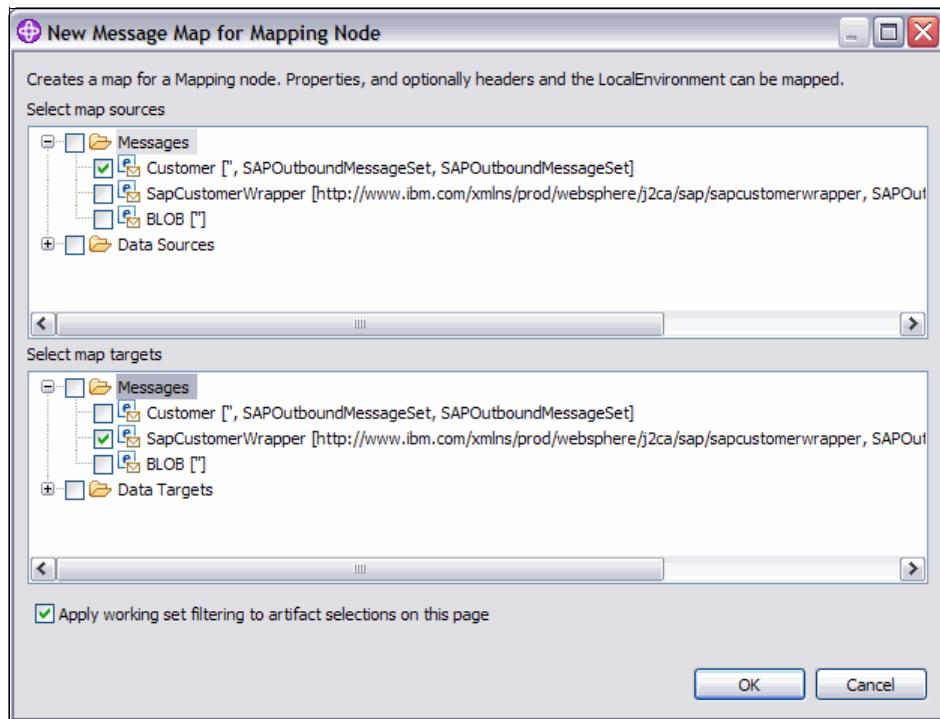


Figure 6-38 Mapping between the input format and the BAPI format

- d. Click **OK**.
 - e. Click **OK** again.
14. The mapping tool is automatically opened in the WebSphere Message Brokers Toolkit. The Properties folders in the source and target panels are mapped by default. Perform the following tasks:
- a. In the source panel, expand Customer.
 - b. In the target panel, expand sapcustomerwrapper and then SapBapiCustomerCreatefromdata.
 - c. Right-click **SapPiPersonaldata**, and click **Map by Name**, as shown in Figure 6-39 on page 189.

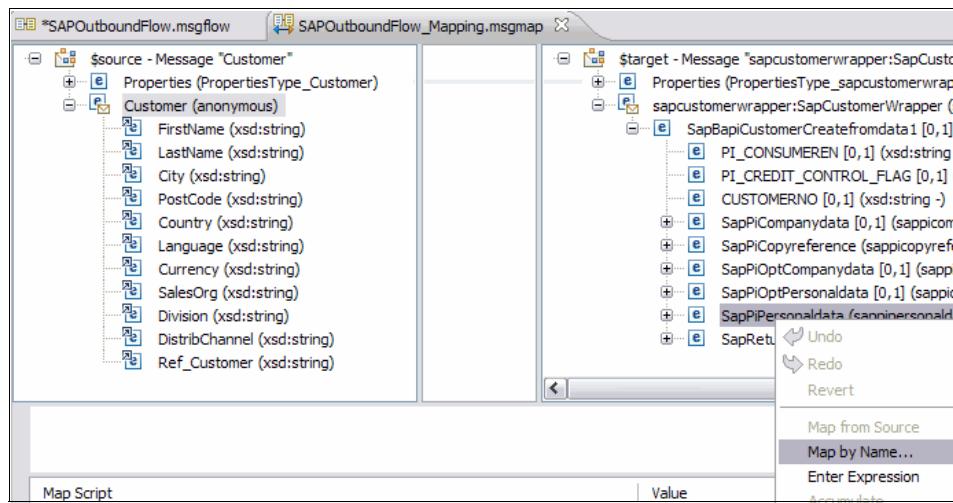


Figure 6-39 Mapping SapPiPersonaldata

The following values are selected by default:

- Mapping Scope: Map leaves of the selected nodes
- Name Matching Options: Alphanumeric characters
- Mapping Criteria: Create mappings between sources and targets with the same name

Leave these defaults. Click **Next**, which results in the automatic mappings of similar values in the source and target panels.

- In the scenario example, the target mapping has an additional post code element called POSTL_COD2. The source mapping does not require this, so it is deselected, as shown in Figure 6-40 on page 190. Click **Finish**.

Map By Name	
Select mappings to be created	
All source and target names satisfying previously chosen options are listed. De-select a target by unchecking the box. De-select a source by clicking the button.	
Selectable Mapping Targets	Selectable Mapping Sources for Target
<input checked="" type="checkbox"/> FIRSTNAME	FirstName
<input checked="" type="checkbox"/> LASTNAME	LastName
<input checked="" type="checkbox"/> CITY	City
<input checked="" type="checkbox"/> POSTL_COD1	PostCode
<input type="checkbox"/> POSTL_COD2	
<input checked="" type="checkbox"/> COUNTRY	Country
<input checked="" type="checkbox"/> LANGU_P	Language
<input checked="" type="checkbox"/> CURRENCY	Currency

Figure 6-40 Mapping similar elements in the source and target panels

- e. In the source panel, highlight **Customer**. In the target panel, right-click **SapPiCopyreference**, and click **Map by Name**.
 - f. Leave the default values that are already selected, and select **Create mappings when source and target names are more similar than**. Click **Next**.
 - g. The mappings are made automatically for you. Click **Finish**.
15. The final node we must add is an MQOutput node. Select an MQOutput node from the WebSphere MQ folder of the node Palette and drag-and-drop it onto the message flow. Click **MQOutput node**, and in the MQOutput Node Properties:
- a. Select the **Basic** tab.
 - b. In the Queue name field, type SAP_OUTBOUND_OUT.
16. Connect the nodes in the message flow, as shown in Figure 6-41:
- MQInput node (out) → Mapping node (in)
 - Mapping node (out) → SAPRequest node (in)
 - SAPRequest node (out) → MQOutput node (in)

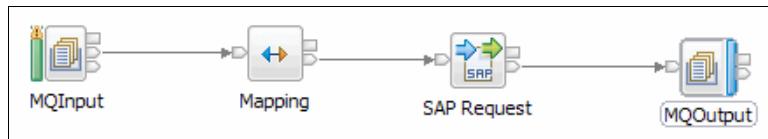


Figure 6-41 The SAPOutbound message flow

17. Save the message flow.

Step 4: Creating and deploying the BAR file to the broker

To create and deploy the BAR file to the broker:

1. Right-click the **SAPOutbound** folder, and select **New → Message Broker Archive**. Type SAPOutbound in the Name field, and click **Finish**.
2. The Prepare BAR file panel, Figure 6-42, is displayed. Select all of the components that were created for the SAPOutbound scenario, and click **Build broker archive**.

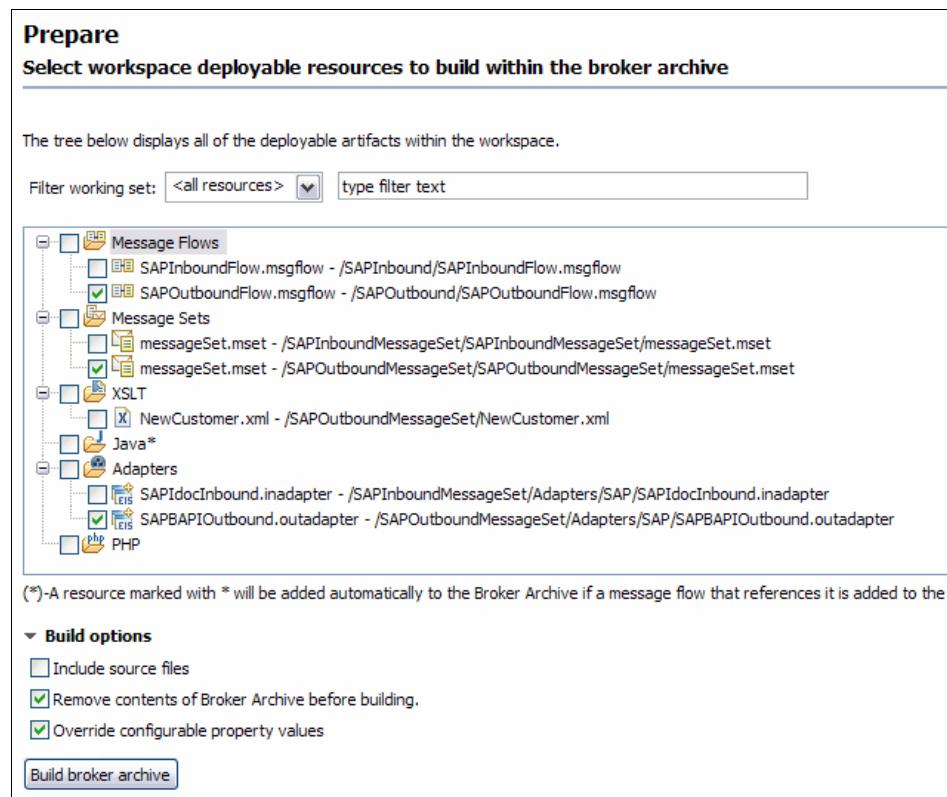


Figure 6-42 Prepare Outbound Broker Archive file

3. When the operation successfully completes, click **OK**, and save the BAR file.
4. Click the **SAPOutbound.bar** file, and drag it into the WebSphere MQ Explorer taskbar. When the MQ Explorer opens up, drop the BAR file onto the brokers execution group.
5. After the components are successfully deployed to the execution group, as shown in Figure 6-43 on page 192, you can test the message flow.

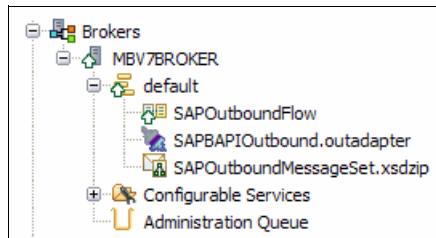


Figure 6-43 Successfully deployed components

Testing the SAPOutbound message flow

To test the SAPOutbound message flow:

1. In the WebSphere Message Broker Toolkit, open the SAPOutboundFlow message flow.
2. Right-click **MQInput node**, and select **Test**.
3. In the SAPOutboundFlow.mbtest tool, at the bottom right side of the mbtest tool, select the **Configuration** tab.
4. Select **I will deploy the specified Broker Archive manually**, and click **Browse**.
5. Select **SAPOutbound.bar**, and click **OK**.
6. At the bottom right side of the mbtest tool, select the **Events** tab.
7. Select **Edit** as text from the pull-down list in the Body textbox of the Message panel.
8. Click **Import Source**, and navigate to the directory that contains the NewCustomer.xml file (C:\AdapterFiles\SAPFiles\data in this scenario example). Select the **NewCustomer.xml** file, and click **Open**.

The input message body in the mbtest tool opens, as shown in Figure 6-44 on page 193.

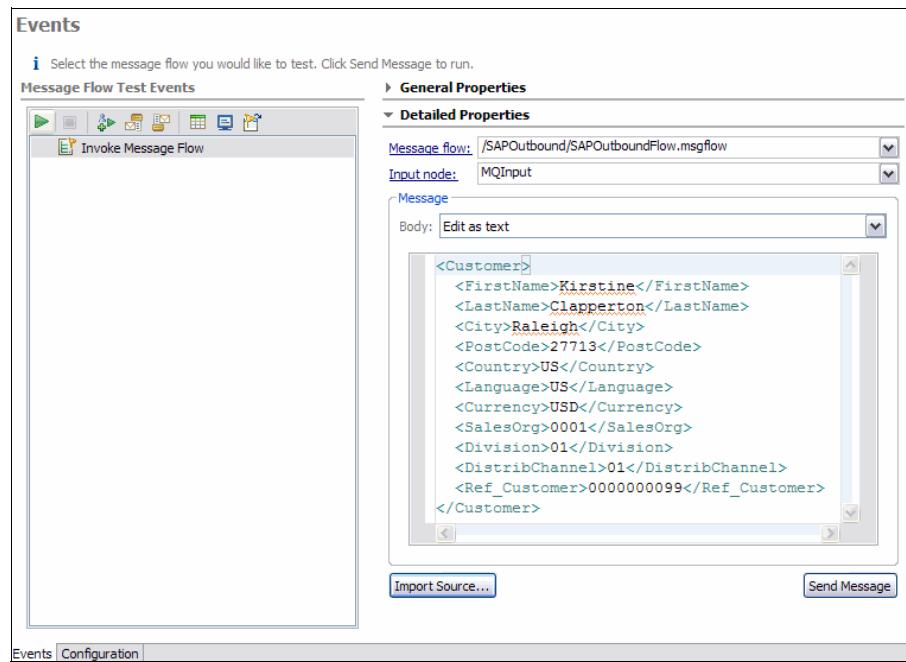


Figure 6-44 Message body opened in mbtest tool

9. Click **Send Message**.
10. In the Select Deployment Location window, Figure 6-45 on page 194, select the execution group, and click **Finish**.

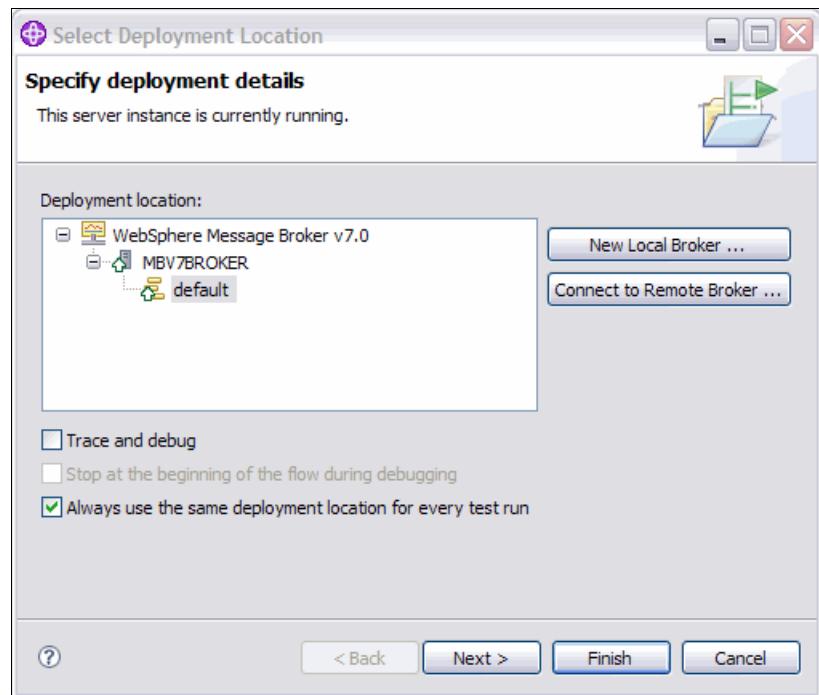


Figure 6-45 Select deployment location

11. After the message flow successfully processes the message, the result is displayed in the mbtest tool. We can see that SAP generated a value of 0000033696 in the CUSTOMERNO field, shown in Figure 6-46 on page 195.

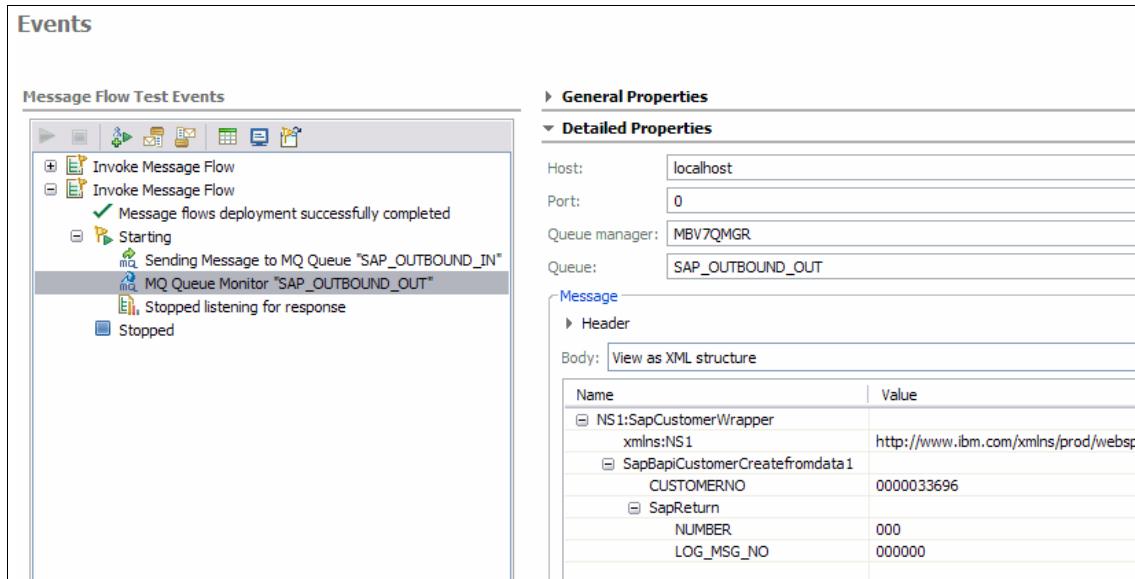


Figure 6-46 The mbtest tool result

12. We can verify that customer Kirstine Clapperton was created with customer number 0000033696 in the SAP system. We use the SAP Easy Access menu to look up this information:
- Log onto the SAP system, expand the SAP menu, and navigate to the Customer Display menu.
 - Enter the customer number that was received in the output message in the Customer field, and click **Enter**, as shown in Figure 6-47 on page 196.

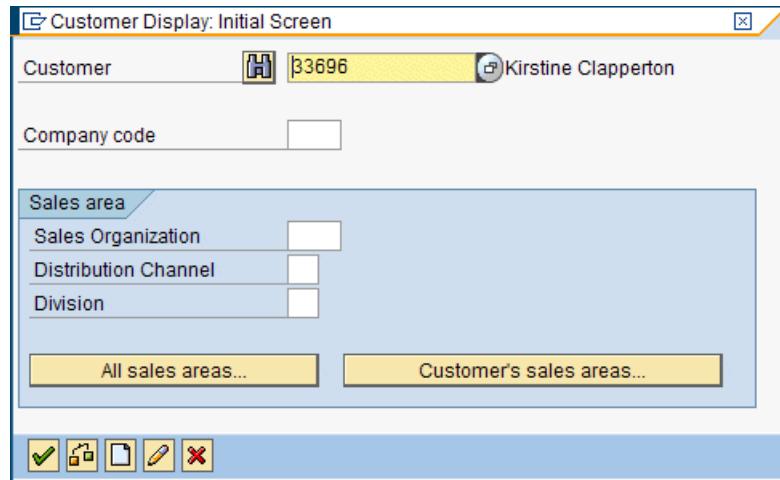


Figure 6-47 Customer Display: Initial panel

The customer that is associated with customer number 33696 is displayed. We can see that this is the information that we are expecting, as shown in Figure 6-48 on page 197.

The screenshot shows the SAP Customer Display: General Data interface. At the top, the menu bar includes Customer, Edit, Goto, Extras, Environment, System, and Help. Below the menu is a toolbar with various icons. The main title is "Display Customer: General Data". A sub-header indicates "Other Customer" and "Data for Invoice Summary (Japan)". The customer number is 33696, and the name is Kirstine Clapperton from Raleigh. The interface is divided into several tabs: Address, Control Data, Payment Transactions, Marketing, and Contact Person. The Address tab is active. It contains sections for Name, Search Terms, Street Address, PO Box Address, and Communication. The Name section shows Title (empty) and Name (Kirstine Clapperton). The Search Terms section has a search term of CLAPPERTON. The Street Address section shows Street/House number (empty), Postal Code/City (27713, Raleigh), Country (US, USA), and Region (empty). The PO Box Address section shows PO Box (empty) and Postal code (empty). The Communication section shows Language (Finnish), Telephone (empty), Mobile Phone (empty), Extension (empty), and Other communication... (with a yellow background).

Figure 6-48 Customer Display: General Data

6.9.3 Scenario 3: Simulated master data synchronization scenario between SAP and Siebel enterprise information systems

In scenario 3, we simulate a master data synchronization between an SAP EIS system and a Siebel EIS system. Ideally, the scenario includes an Advanced Business Application Programming (ABAP) program that, after creating a

Business Partner (BP) in the SAP system, sends an event to a WebSphere Message Broker flow that is acting as an RFC-enabled program. This scenario simulates the ABAP program using the BAPI Test tool (se37) and assume that the create action took place, after which, the event was sent to WebSphere Message Broker.

The WebSphere Message Broker flow queries Siebel to determine if the specified BP already exists as an account. If it does exist, the flow returns data about the account and confirmation that it exists to SAP. If the account does not already exist, Siebel creates an account based on the data that is used to create the same account in SAP and returns this data and confirmation that the account was created to SAP.

The message flow makes use of the SAPInput node, the SiebelRequest node, and the SAPReply node. The SAP adapter supports inbound processing for the synchronous callback interface (SCI). An RFC-enabled function call is sent to an endpoint by way of the adapter, and the response from the endpoint is returned to the SAP server. The following list describes the sequence of processing actions that result from an inbound request using the Synchronous callback interface:

1. The adapter starts event listeners, which listen for the RFC-enabled function event (which was specified with the RFC ProgramID property) on the SAP server.
2. When an RFC-enabled function call is invoked from SAP, the RFC-enabled function event is pushed to the adapter by way of the event listeners.
3. The adapter converts the RFC-enabled function event to a business object.
4. The adapter sends the business object to an endpoint (Siebel) in a synchronous manner:
 - The adapter generates the business object name using the received RFC-enabled function name.
5. The adapter receives the response business object from the endpoint.
6. The adapter maps the response business object to an RFC-enabled function and returns it to the SAP server.

The assumptions are:

- ▶ A company is using a Siebel system for business-partner related operations. Each business partner is an account in the system.
- ▶ The same company is using an SAP system for customer and business partner, and Siebel must be updated each time a business partner is created in SAP.

- ▶ The SAP Test Function Module simulates the creation of a new business partner in SAP, but nothing is actually created in the SAP system. The creation of the business partner in SAP is simulated and an event is sent to the Siebel adapter to create the same business partner in Siebel.

The following SAP components, which are needed for the interaction with WebSphere Message Broker in the Inbound scenario, were configured on the SAP server:

- ▶ RFC Destination with a connection type of T (TCP/IP Connection) and RFC program ID of BETA02.

Step 1: Preparing the environment for WebSphere Adapters nodes

The sapjco3.jar JAR and sapjco3.dll native library files are downloaded and saved to a directory on the local broker machine, in this case, C:\AdapterFiles\SAPFiles.

The Siebel.jar and SiebelJI_1language_code.jar (SiebelJI_enu.jar) files are downloaded and saved to a directory on the local broker machine, in this case, C:\AdapterFiles\SiebelFiles.

We use the IBM WebSphere MQ Explorer to generate and issue the **mqsiclangeproperties** commands to set up the dependencies.

1. Open the IBM WebSphere MQ Explorer.
2. In the MQ Explorer Navigator panel, right-click the **Configurable Services** folder for the MQV7BROKER, and select **Show IBM Predefined Templates**. This step adds a + sign to the Configurable Services folder.
3. Click the + sign that is next to Configurable Services to expand the services, as shown in Figure 6-49 on page 200.

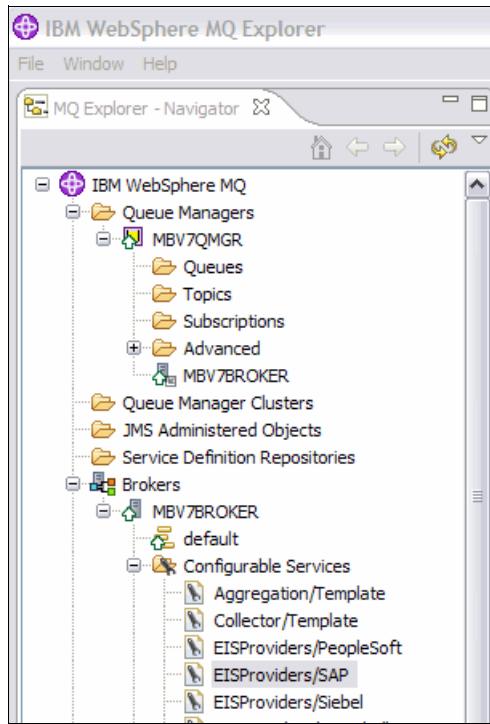


Figure 6-49 The Expanded Configurable Services folder

4. Right-click **EISProviders/SAP**, and select **Properties**. A panel opens that displays both jarsURL and nativeLibs with the default values.
5. In the Value column for each of the jarsURL and nativeLibs Keys, type C:\AdapterFiles\SAPFiles, and click **Finish**. The panel displays the **mqsicchangeproperties** command that is submitted to the broker, as shown in Figure 6-50 on page 201.

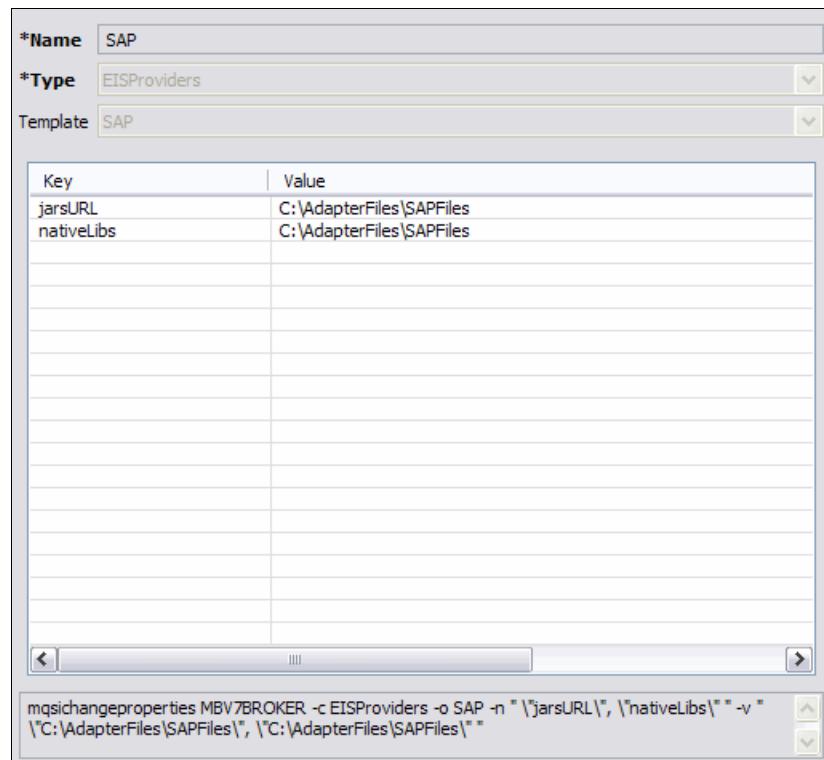


Figure 6-50 The `mqsischangeproperties` command

6. The Broker Administration log should contain a BIP2880I message for each of the `mqsischangeproperties` commands, indicating that they were successfully completed.
7. Right-click the **EISProviders/Siebel** in the **Configurable Services** folder, and select **Properties**, which opens a panel that displays both jarsURL and nativeLibs with the default values.
8. In the Value column for each of the jarsURL and nativeLibs Keys, type C:\AdapterFiles\SiebelFiles, and click **Finish**. The panel displays the `mqsischangeproperties` command that are submitted to the broker, as shown in Figure 6-51 on page 202.

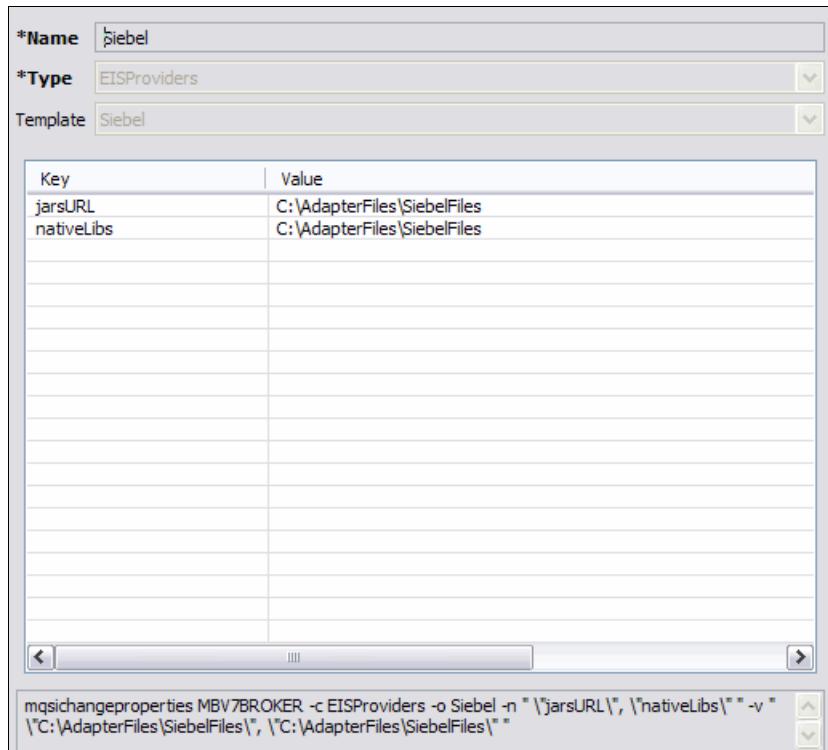


Figure 6-51 The `mqsicchangeproperties` command

The Broker Administration log should contain a BIP2880I message for each of the `mqsicchangeproperties` commands, indicating they were successfully completed.

9. Stop and restart the Broker for the changes to take effect.

Step 2a: Discovering the SAP objects and services

To discover the SAP objects and services:

1. Open the WebSphere Message Broker Toolkit. In the Projects section of the Broker Development perspective, click **Quick Starts**.
2. From the adapter connection, click **Start**, and select **IBM WebSphere Adapter for SAP Software with transaction support (IBM: 7.0.0.0)**. Click **Next**.
3. Leave the default value (CWFAP_SAPAdapter_Tx), and click **Next**.

4. Specify the location of the required SAP sapjco3.jar and System library files. For each option, click **Browse**, and navigate to C:\AdapterFiles\SAPFiles, as shown in Figure 6-52. Select the appropriate file, and click **Open**. Click **Next**.

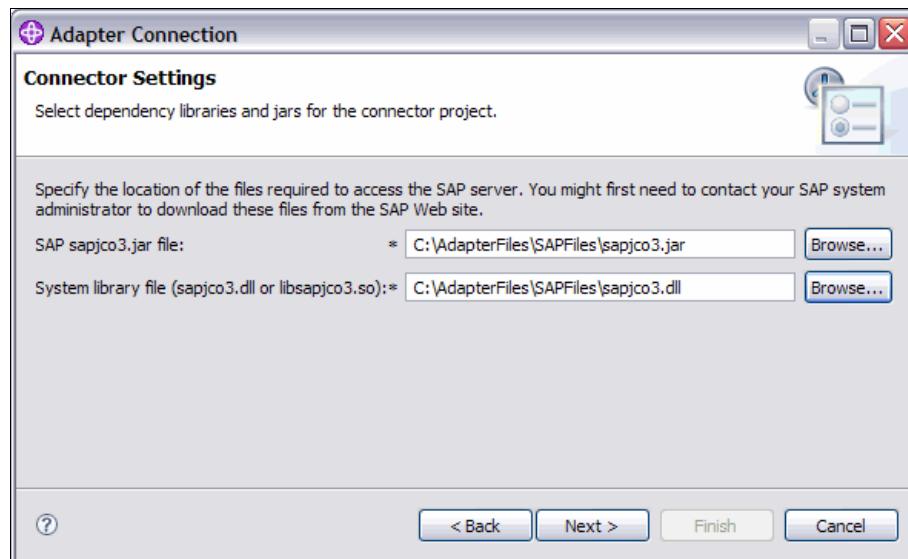


Figure 6-52 Adapter Connection wizard: Connector Settings

5. Select **Inbound**, Figure 6-53 on page 204, and then click **Next**.

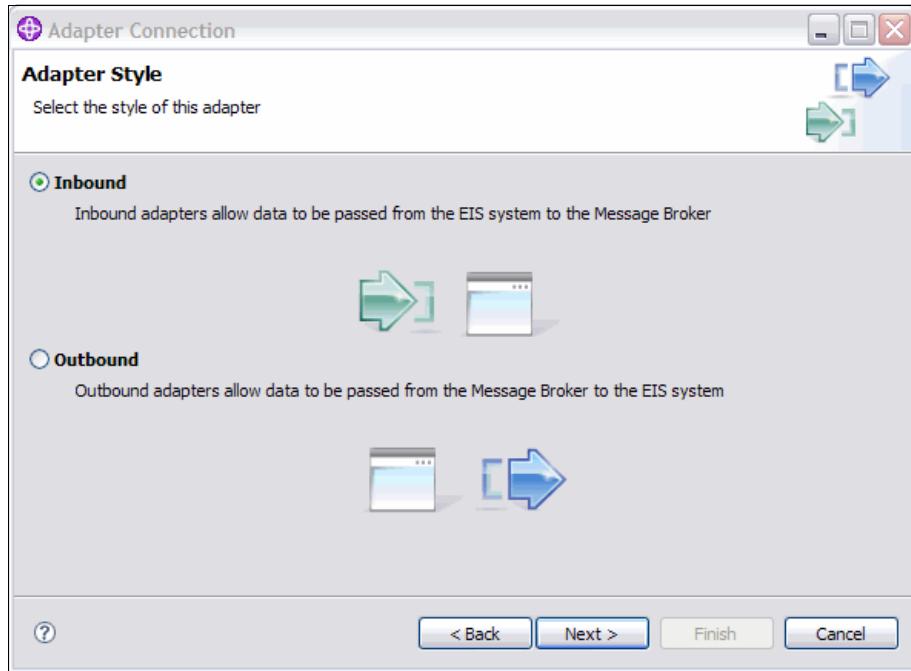


Figure 6-53 Adapter Connection wizard: Inbound

6. In the next panel, we configure the Discovery Agent. This agent is used to search inside SAP for the business object that we want to transform into Message Definition files inside a Message Set. Complete the Host name, System number, Client, User name, Password, and SAP interface name fields with the appropriate values. Select **BAPI** from the pull-down list for the SAP interface name, as shown in Figure 6-54 on page 205. Click **Next**.

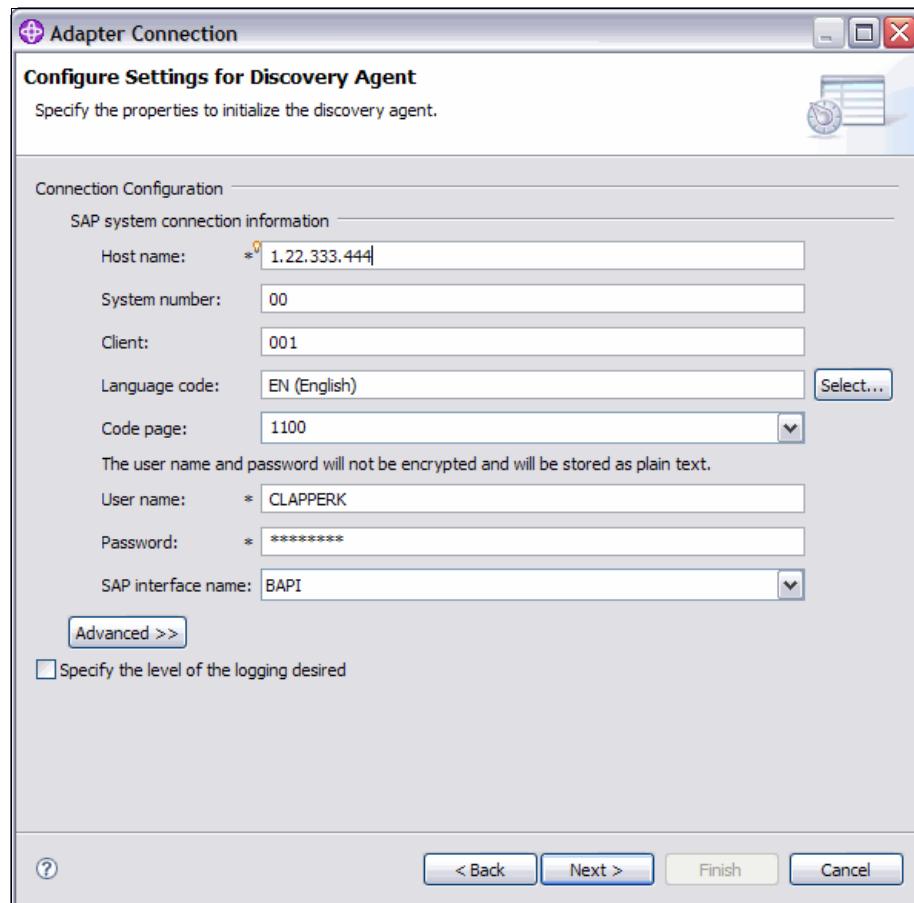


Figure 6-54 Adapter Connection wizard: Configure SAP connection

7. In the Find and Discover Services panel, click **RFC** to highlight it. Do not expand RFC at this point, we first must define a query.
8. In the top-left corner of the Objects discovered by query panel, click **Create or edit filter** to define your search criteria.
9. This scenario is going to use a BAPI called BAPI_BUPA_CREATE_FROM_DATA, so in the Find objects with this pattern field, Figure 6-55 on page 206, type BAPI*BUPA*CREATE*. Click **OK**.

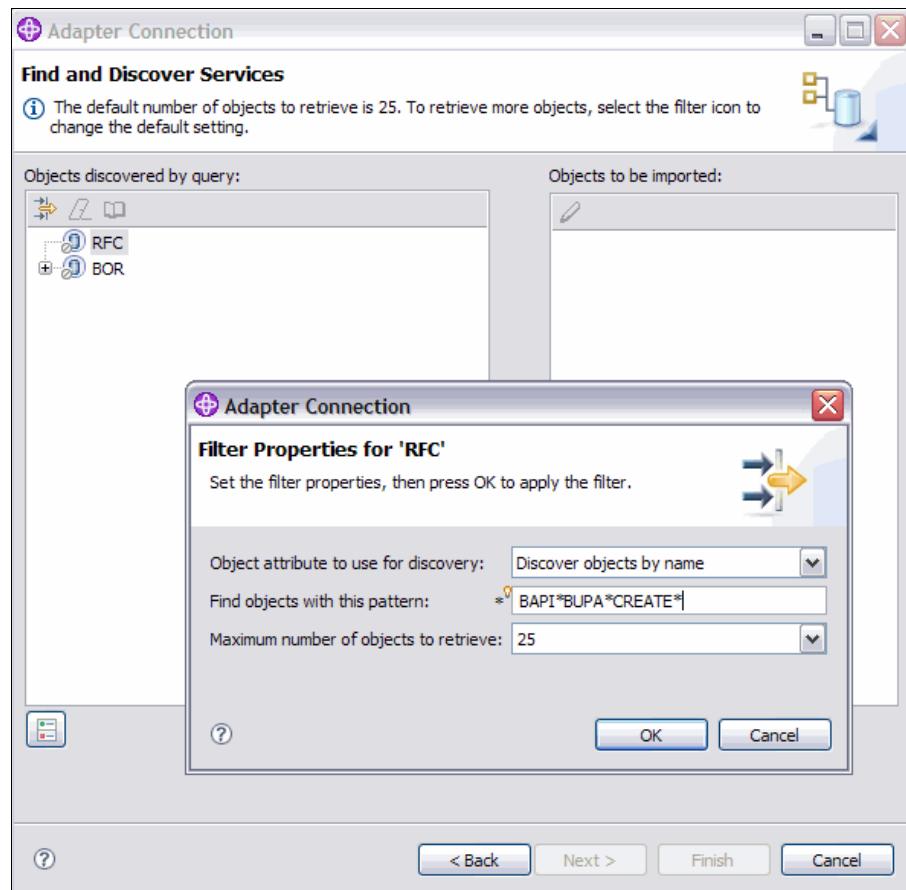


Figure 6-55 Adapter Connection wizard: Filter properties for RFC

10. Now expand the components under RFC (filtered) by clicking the + sign. Select **BAPI_BUPA_CREATE_FROM_DATA**, and click the > sign to add the selected object to the right panel.
11. Leave the default values, as shown in Figure 6-56 on page 207. Click **OK**.

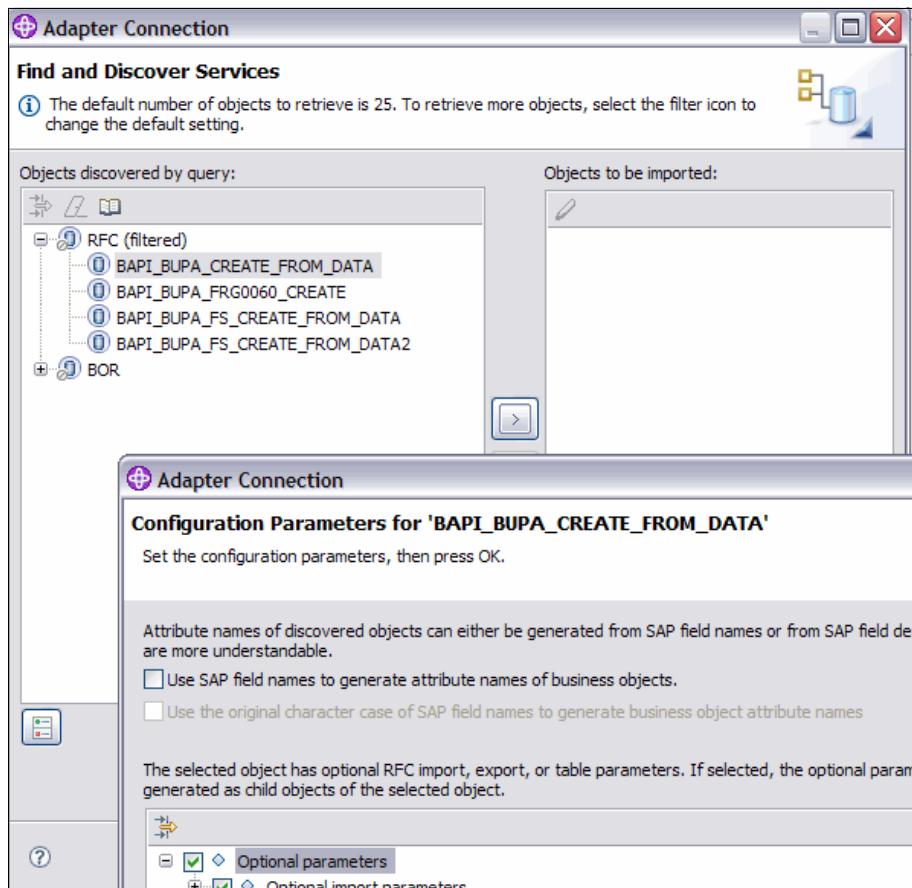


Figure 6-56 Adapter Connection wizard: Configure parameters for BAPI

12. After BAPI_BUPA_CREATE_FROM_DATA is added to the Objects to be imported, click **Next**.
13. Leave the default values, and click **Next**, as shown in Figure 6-57 on page 208.

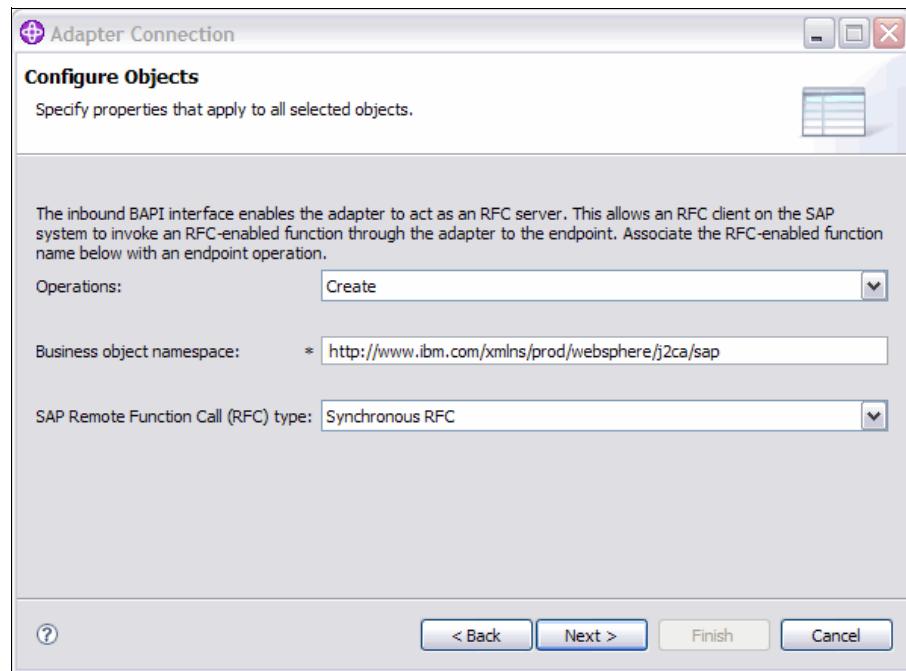


Figure 6-57 Adapter Connection wizard: Configure objects

14. Now we must configure the adapter connection properties for the broker runtime. Most of the values should already be provided, and you should only need to enter the RFC program ID and the password for the specified user, as shown in Figure 6-58 on page 209. Click **Next**.

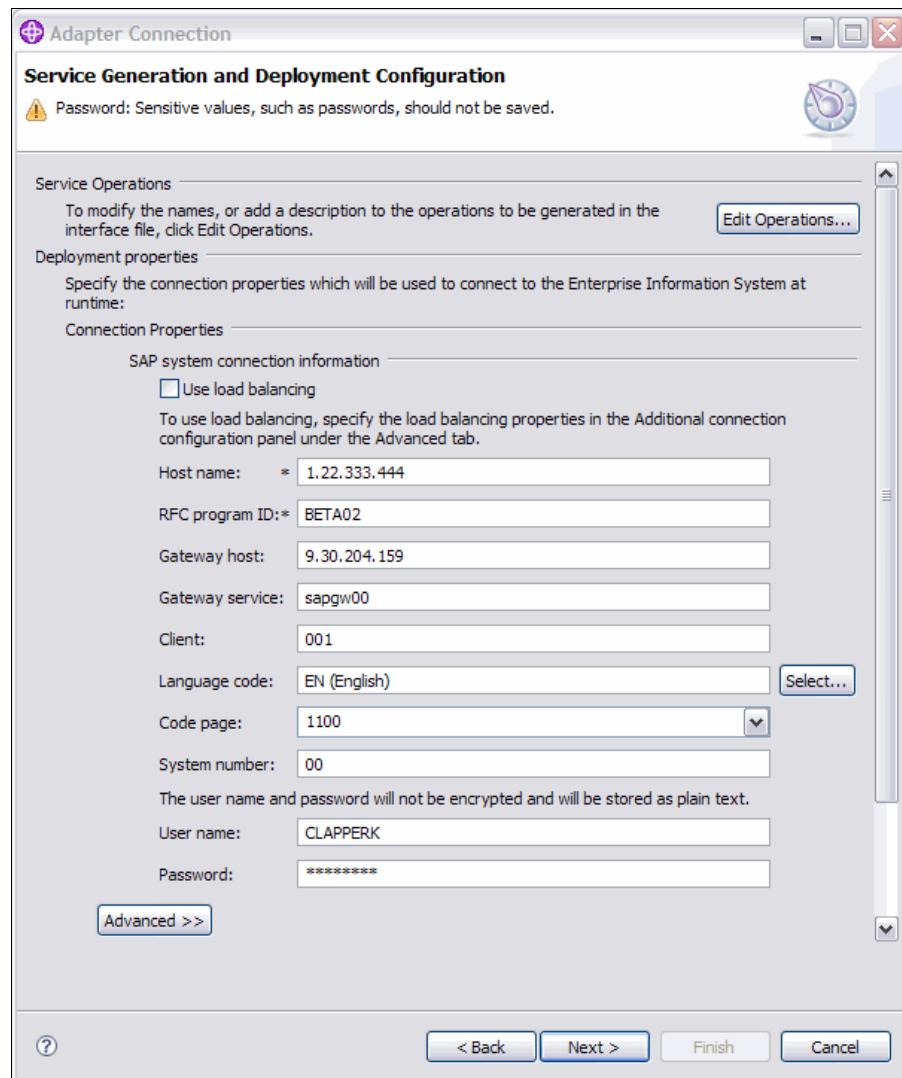


Figure 6-58 Adapter Connection wizard: Configure broker runtime connection

15. In the Publishing Properties panel, we provide the names for the resources that are generated. In the Message flow project name: * field, type SAPSCI. All of the fields, apart from Adapter component name: *, are completed for you. In the field for the Adapter component name, type SAPSCI, and click **Finish**, as shown in Figure 6-59 on page 210.

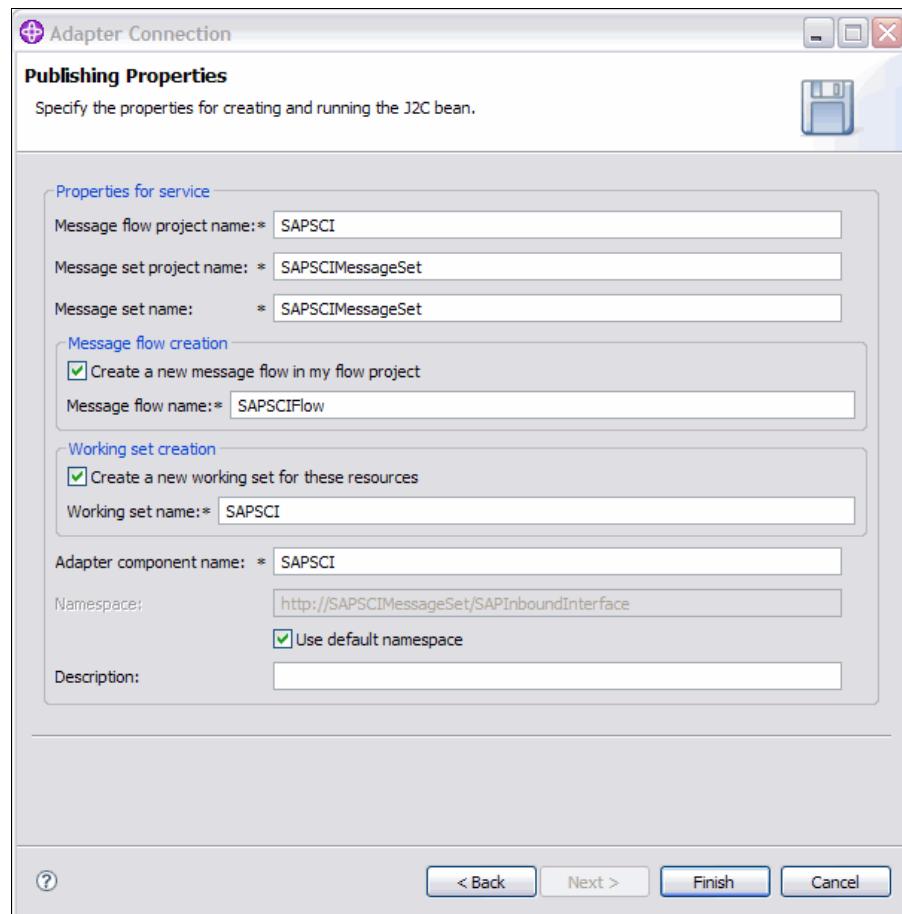


Figure 6-59 Adapter Connection wizard: Publishing properties SAPSCI

16. The generated resources are now listed in the Projects section of the Broker Development perspective in the WebSphere Message Broker Toolkit and include, as shown in Figure 6-60 on page 211:
- Message flow: SAPSCIFlow.msgflow
 - Message set: SAPSCIMessageSet
 - Adapter component: SAPSCI.inadapter

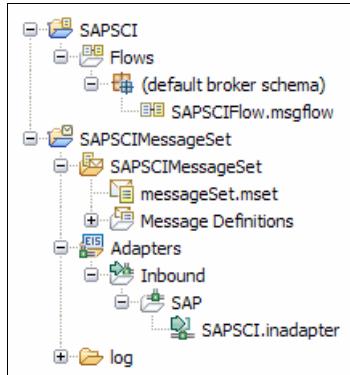


Figure 6-60 The SAPSCI resources generated by the Adapter Connection wizard

Step 2b: Discovering the Siebel objects and services

To discover the Siebel objects and services, we must create the first of two Siebel adapters. We only want to create a new adapter and message set. We do not need a new message flow:

1. In the Broker Application Development perspective of the WebSphere Message Broker Toolkit, click **File** → **New** → **Adapter Connection**.
2. Select **IBM WebSphere Adapter for Siebel Business Applications (IBM: 7.0.0.0)**. Click **Next**.
3. Leave the default value (CWYEB_SiebelAdapter), and click **Next**.
4. Specify the location of the required Siebel Siebel.jar and SiebelJI_language code.jar files, as shown Figure 6-61 on page 212. For each option, click **Browse**, and navigate to C:\AdapterFiles\SiebelFiles. Select the appropriate file, and click **Open**. Click **Next**.

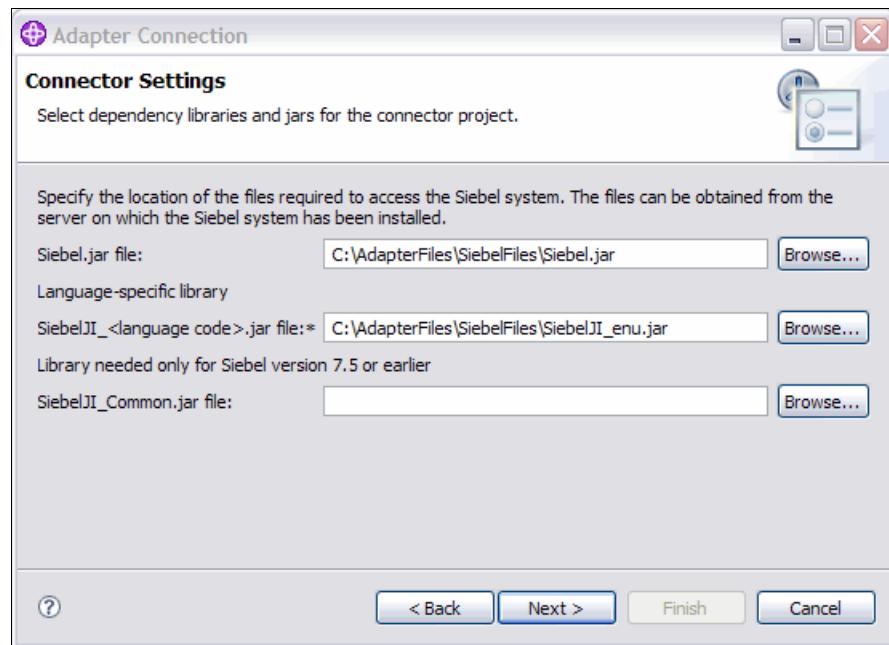


Figure 6-61 Adapter Connection wizard: Connector Settings

5. Select **Outbound**, and click **Next**, as shown in Figure 6-62 on page 213.

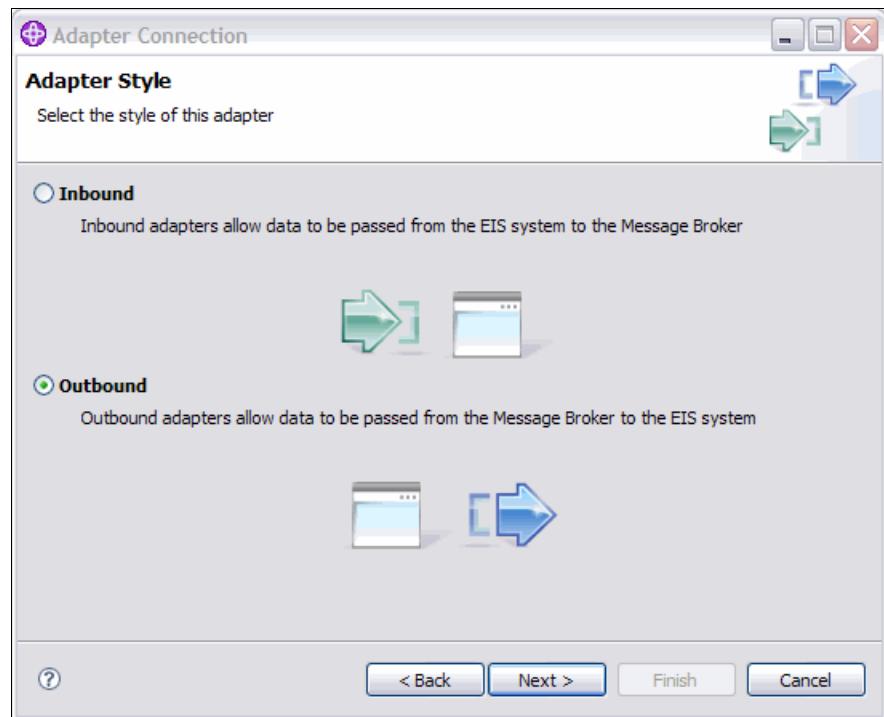


Figure 6-62 Adapter Connection wizard: Outbound

6. In the next panel, we configure the Discovery Agent. Enter the Connection URL, User name, and Password. Accept the remaining default values, as shown as Figure 6-63 on page 214. Click **Next**.

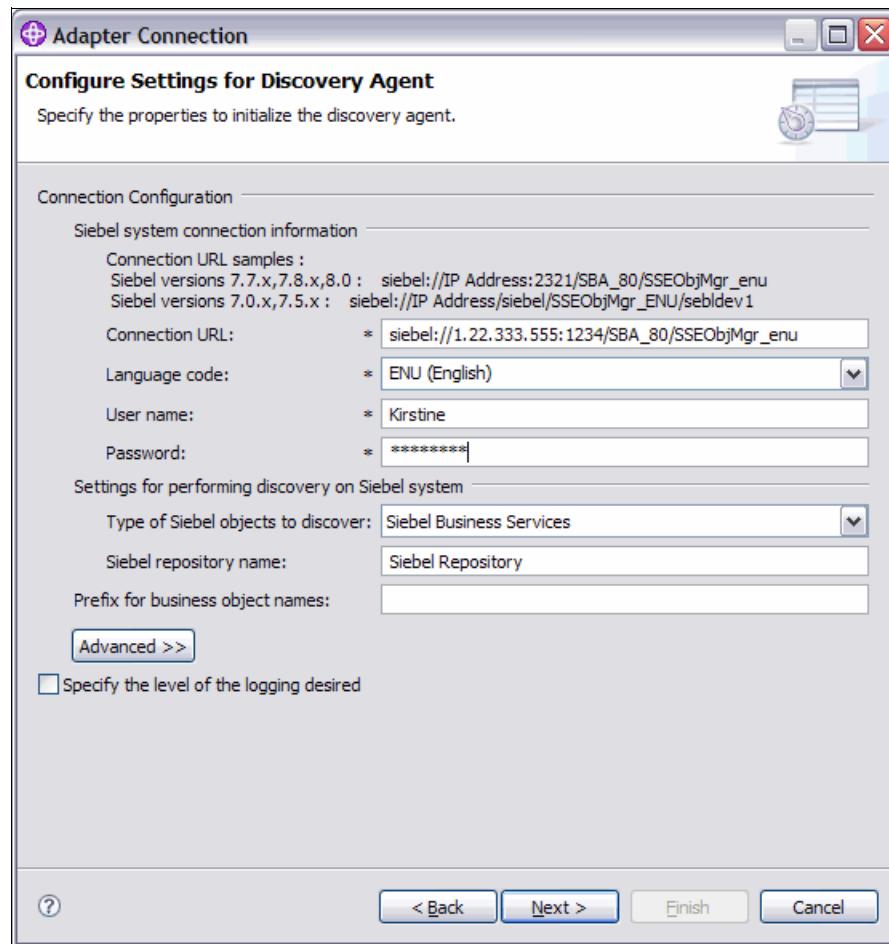


Figure 6-63 Adapter Connection wizard: Configure Siebel connection

7. Click **Edit Query**. In the Query Filter Parameters window, type EAI Siebel*, as shown Figure 6-64 on page 215, and click **OK**.

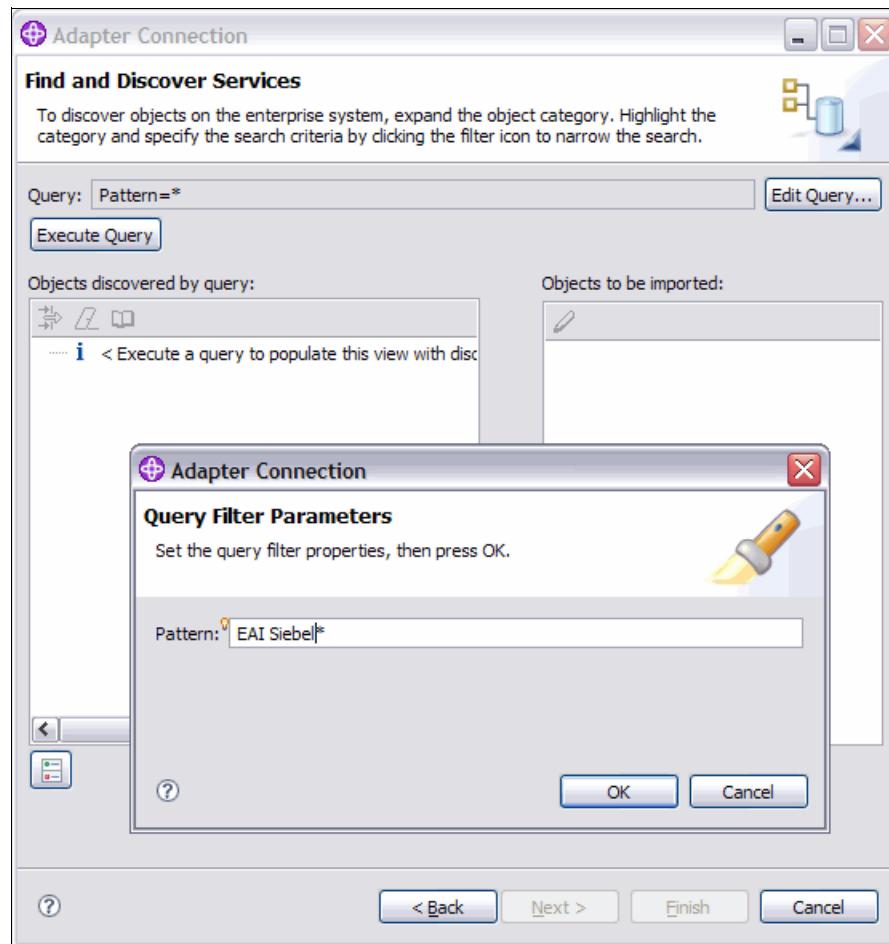


Figure 6-64 Adapter Connection wizard: Query Filter Parameters

8. In the top-left side of the Find and Discover Services panel, click **Execute Query**.
9. Expand the Siebel Business Services folder. Expand EAI Siebel Adapter I.
10. Select **Query**, and click the **>** sign to add the selected object to the right panel, as shown in Figure 6-65 on page 216.

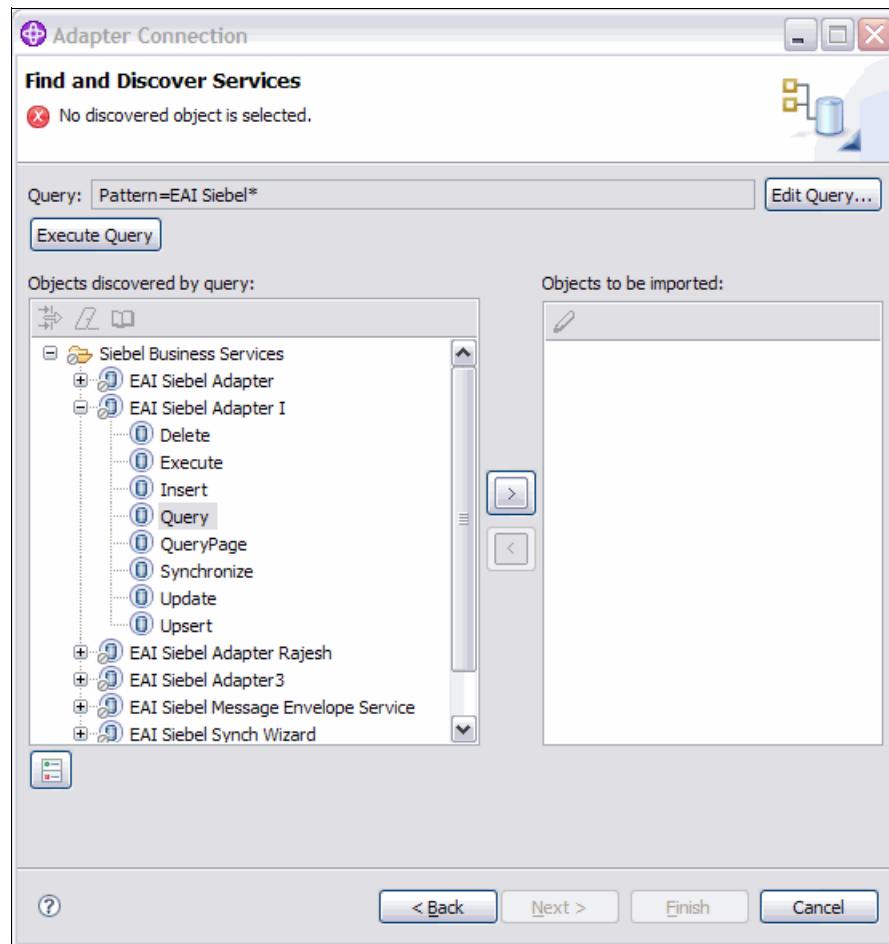


Figure 6-65 Adapter Connection wizard: Add Query object to be imported

11. In the Select Value window, click **Select**. Scroll down, and select **Account (PRM ANI)**, as shown in Figure 6-66 on page 217. Click **OK**.

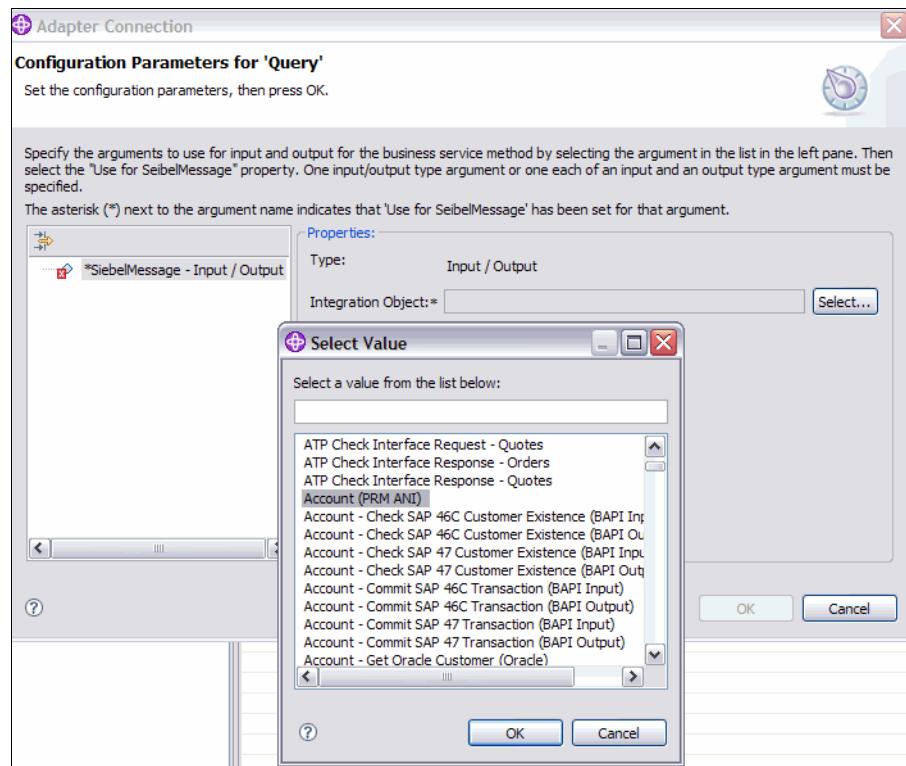


Figure 6-66 Adapter Connection wizard: Select the Query value

12. On the Configuration Parameters for Query panel, click **OK**.
13. On the Find and Discover Services panel, click **Next**.
14. You can choose to Generate business objects with shorter names. Accept the default values in the Configure Objects panel. Click **Next**.
15. We must now configure the adapter connection properties for the broker runtime. Most of the values are already provided. You only need to enter the password for the specified user, as shown in Figure 6-67 on page 218). Click **Next**.

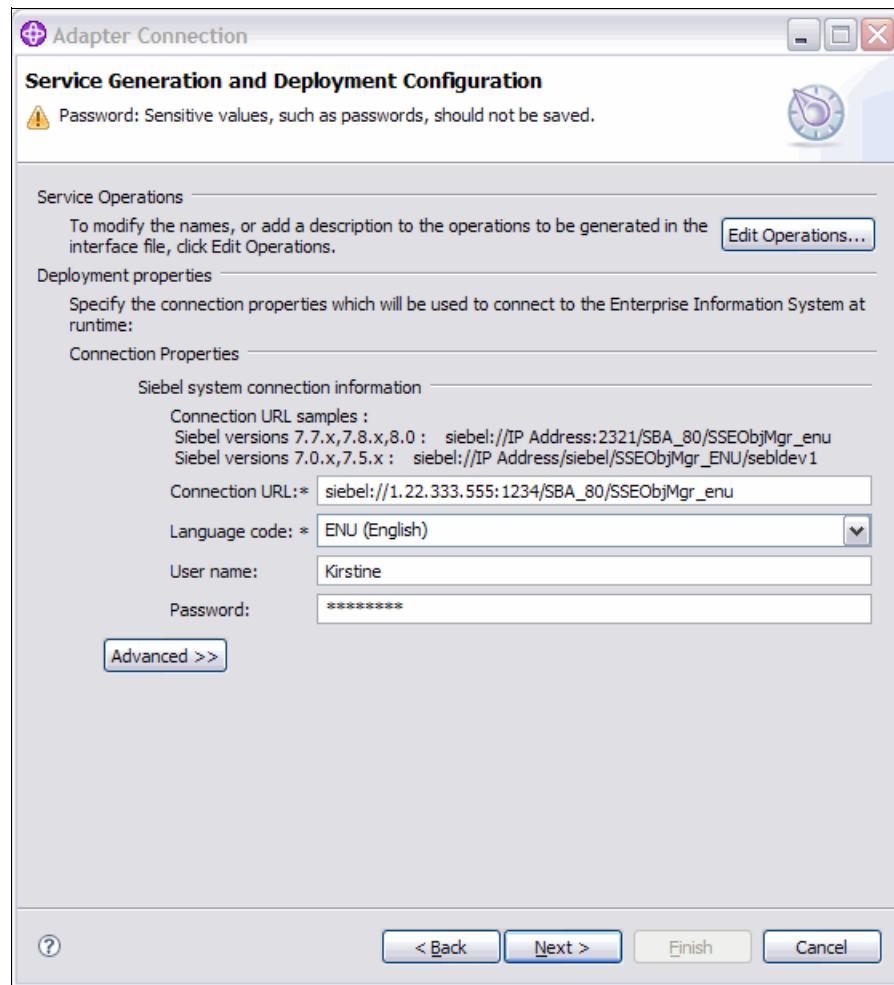


Figure 6-67 Adapter Connection wizard: Configure broker runtime connection

16. Click **New** to create a new message set for the Siebel artifacts. In the Create a new message set window, in both the Message set name and Message set project name fields, type SiebelMessageSet, as shown in Figure 6-68 on page 219. Click **Finish**.

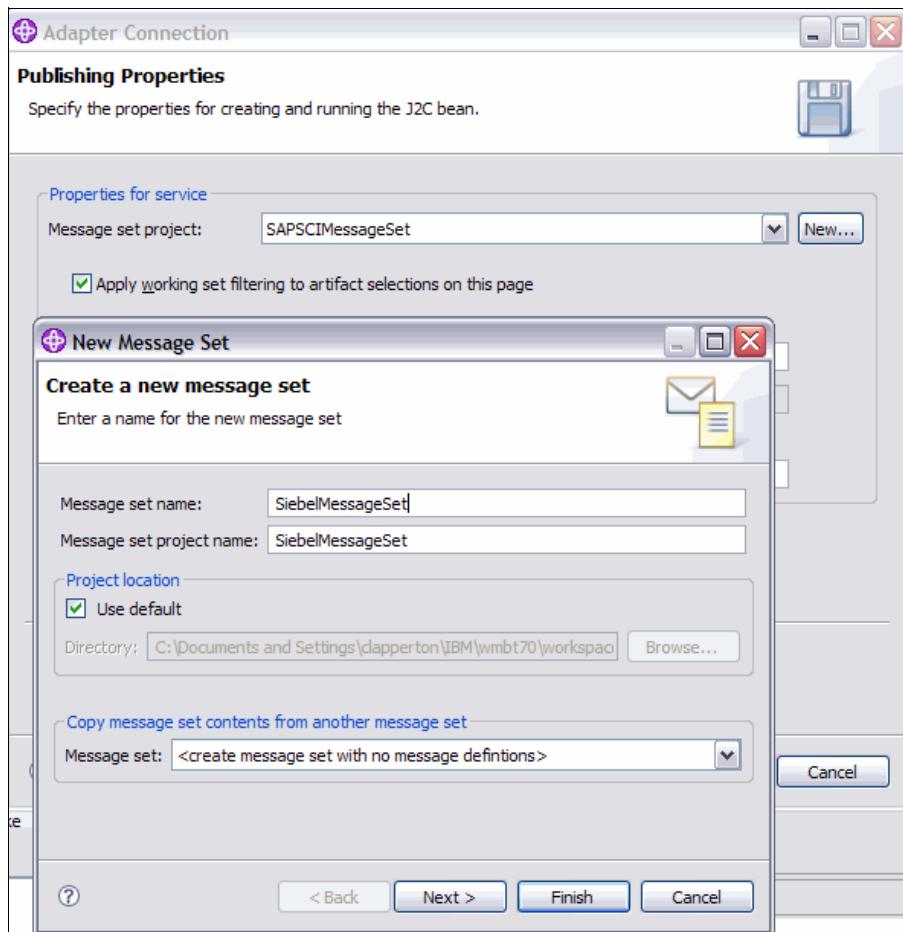


Figure 6-68 Create a new Siebel message set

17. Back in the Publishing Properties panel, in the Adapter component name field, type SiebelQuery, and click **Finish**.

Now we create the second of the required Siebel adapters. As with the first Siebel adapter, we only want to create a new adapter and message set. We do not need a new message flow:

1. In the Broker Application Development perspective of the WebSphere Message Broker Toolkit, select **File** → **New** → **Adapter Connection**.
2. Under the CWYEB_SiebelAdapter folder, select the existing IBM WebSphere Adapter for Siebel Business Applications, and click **Next**.
3. In the Adapter Style panel, select **Outbound**, and click **Next**.

4. Most of the values, except the password, might already exist in the Configure Settings for Discovery Agent panel, as shown in Figure 6-69, complete these fields:
 - a. Enter the password for the specified user.
 - b. In the Type of Siebel objects to discover field, select **Siebel Business**.
 - c. Select the option: **Prompt for additional configuration settings when adding a business object**.
 - d. Click **Next**.

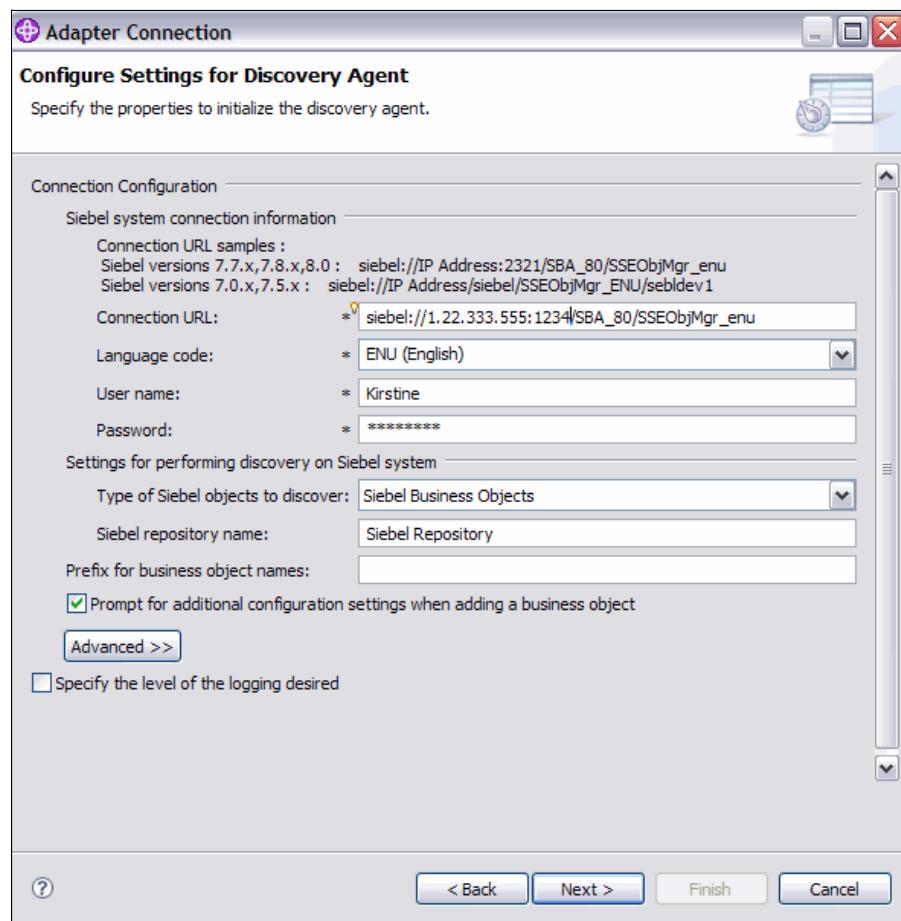


Figure 6-69 Adapter Connection wizard: Configure Siebel connection

5. Click **Edit Query**. In the pop-up Query Filter Parameters window, Figure 6-69, type ***Account***, and click **OK**.

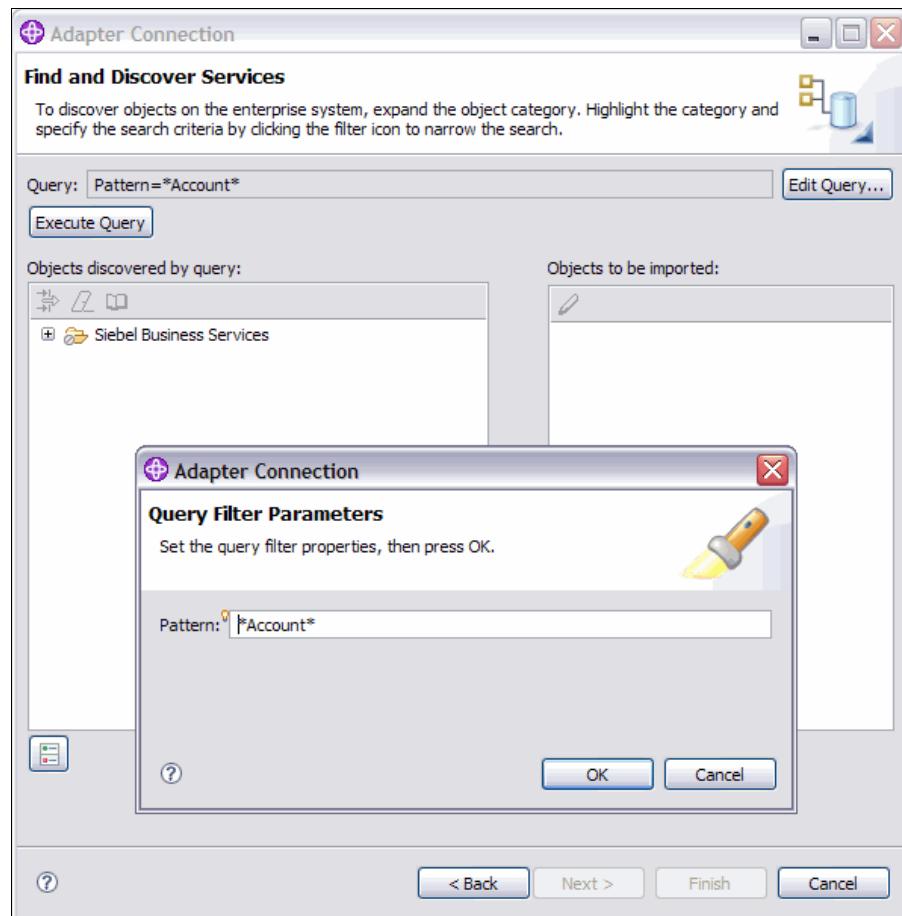


Figure 6-70 Adapter Connection wizard: Query Filter Parameters

6. In the top-left side of the Find and Discover Services panel, click **Execute Query**.
7. Expand the Siebel Business Objects folder. Expand **Account - ESP**.
8. Select **Account**, and click the > sign to add the selected object to the right panel, as shown in Figure 6-71 on page 222.

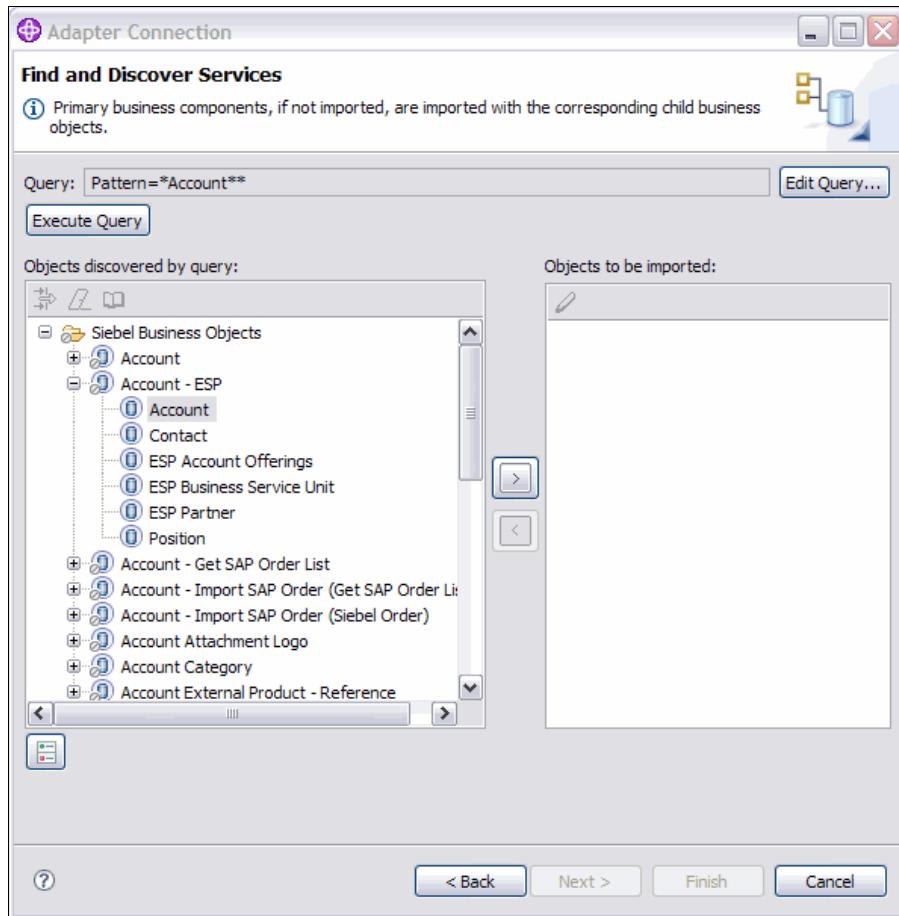


Figure 6-71 Adapter Connection wizard: Add Account object to be imported

9. Clear the All Attributes option. Select the following options, and click **OK**:
 - CurrencyCode
 - CustomerAccountGroup
 - Name
10. On the Find and Discover Services panel, click **Next**.
11. Only the Create operation is required. Select all of the other operations (hold down the Ctrl key and select them), and click **Remove**, as shown in Figure 6-72 on page 223.

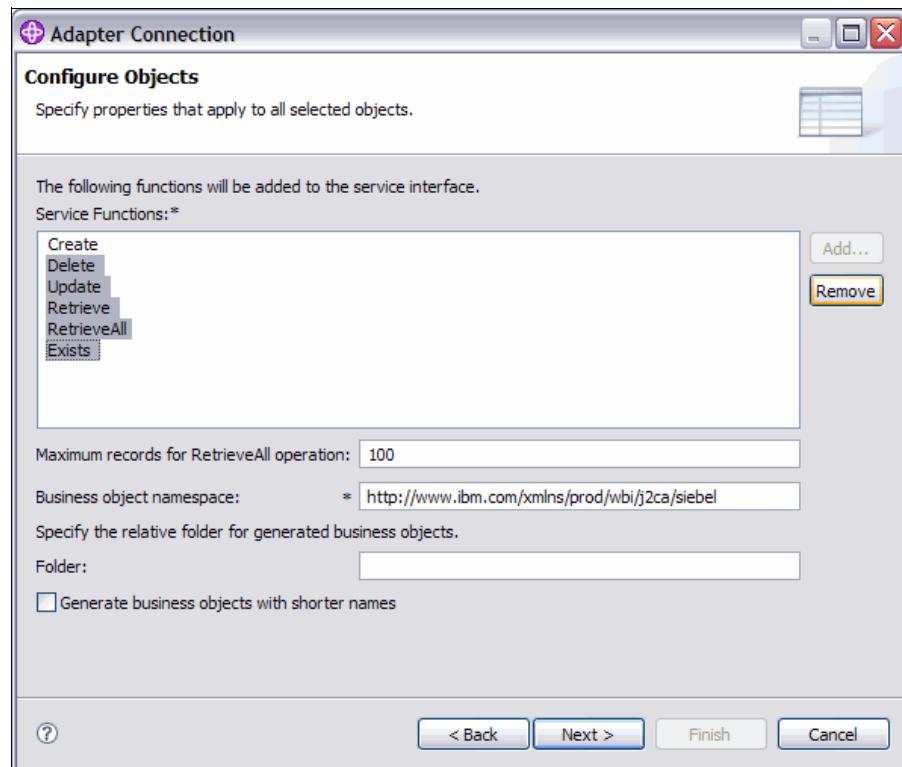


Figure 6-72 Adapter Connection wizard: Remove unnecessary operations

12. After the Create operation is all that remains, click **Next**.
13. In the Service Generation and Deployment Configuration panel, type the password for the specified user, as shown in Figure 6-73 on page 224.

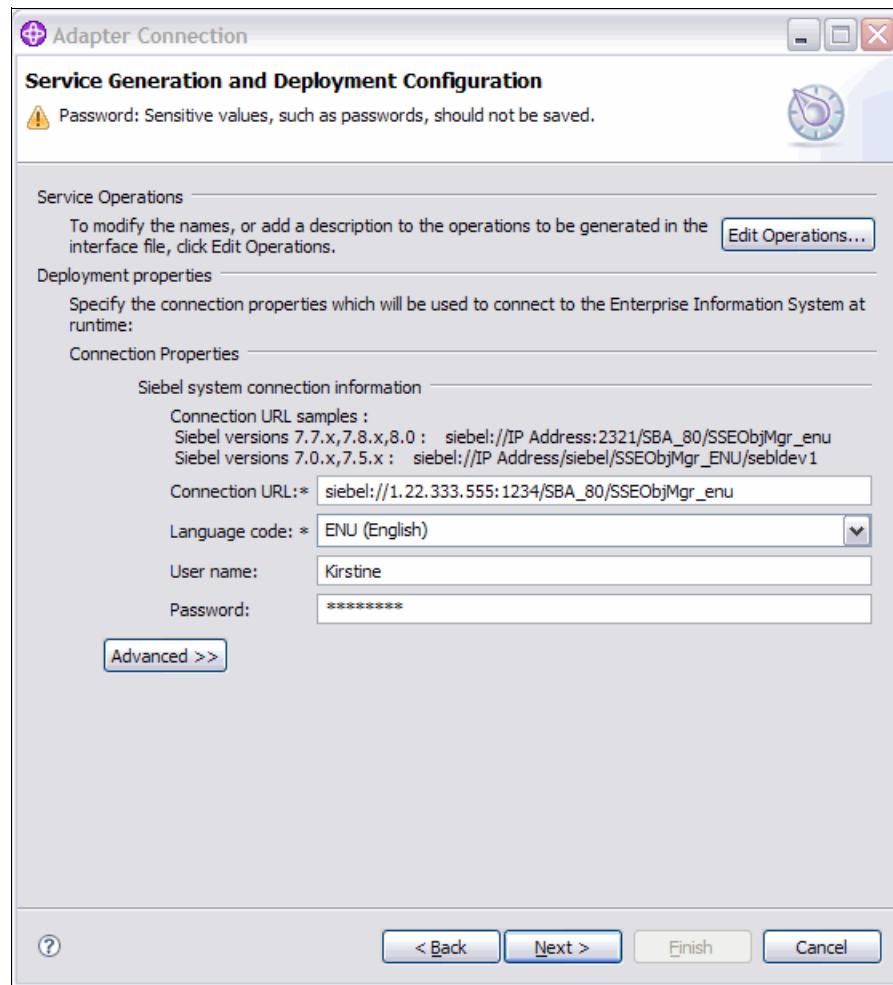


Figure 6-73 Adapter Connection wizard: Configure broker runtime connection

14. Select **SiebelMessageSet** as the Message set project. Type **SiebelCreate** as the name for the Adapter component, as shown in Figure 6-74 on page 225, and click **Finish**.

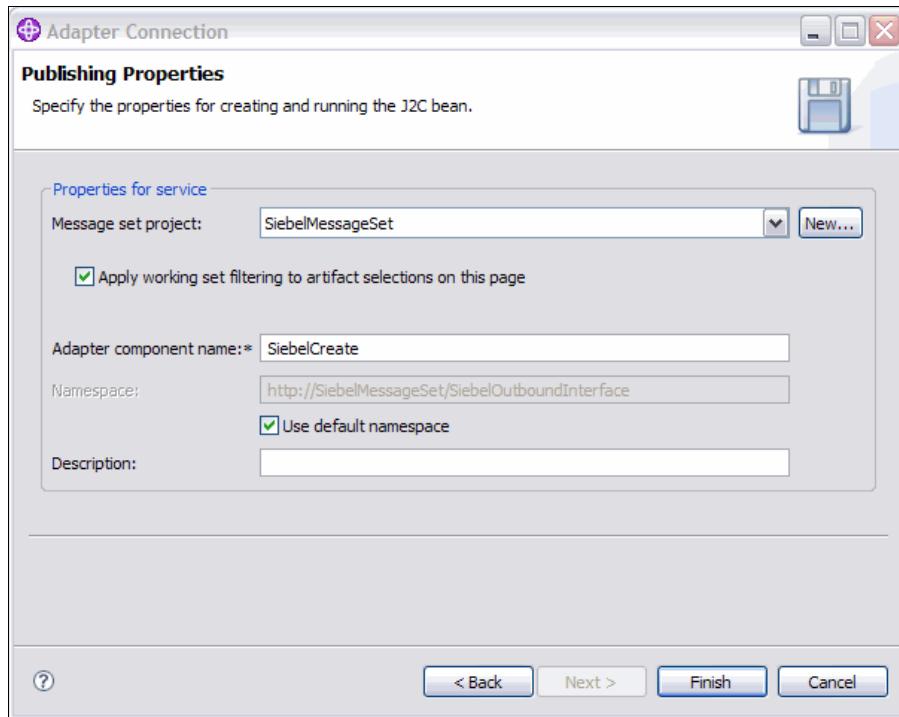


Figure 6-74 Create a new Siebel Adapter component

15. The generated resources should now be listed in the Projects section of the Broker Development perspective in the WebSphere Message Broker Toolkit, as shown in Figure 6-75 on page 226. These resources are:

- Message flow: SAPSCIFlow.msgflow
- Message set: SAPSCIMessageSet
- Adapter component: SAPSCI.inadapter
- Message set: SiebelMessageSet
- Adapter component: SiebelCreate.outadapter
- Adapter component: SiebelQuery.outadapter

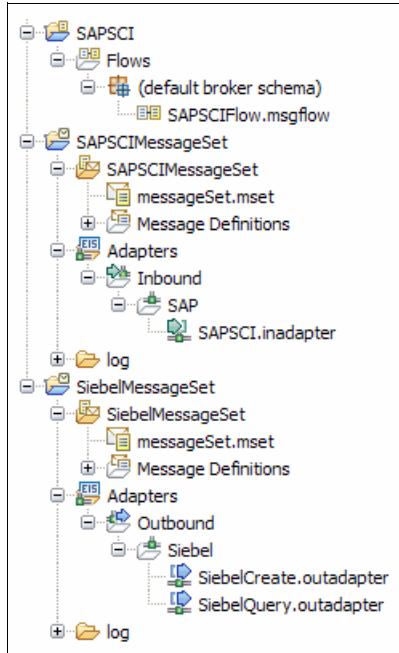


Figure 6-75 The SAP and Siebel resources

Step 3: Adding the WebSphere Adapters node into the message flow

Next, we must associate the SiebelMessageSet message set with the SAPSCI Message Flow Project:

1. Right-click the **SAPSCI Message Flow** Project, and click **Properties**.
2. Click **Project References**, select **SiebelMessageSet**, and click **OK**, as shown in Figure 6-76 on page 227.

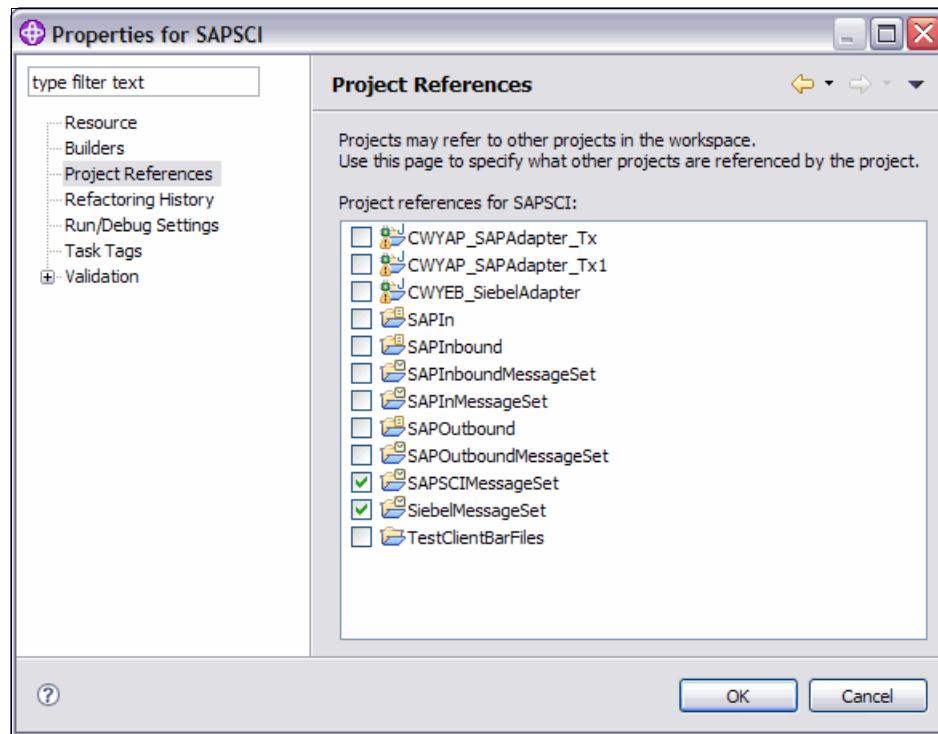


Figure 6-76 Set the SAPSCI project references

3. To open the flow in the flow editor, under the SAPSCI folder, double-click **SAPSCIFlow.msgflow**.
4. Click the **SAPSCI.inadapter** component and drag-and-drop it onto the flow editor.
5. We are not going to use the SAPSCI_Operations subflow, so delete it by selecting the **SAPSCI_OperationsSubFlow** node and pressing the Delete key.
6. From the Transformation folder of the node Palette, select a Compute node, and drag-and-drop it onto the message flow. We use the Compute node to store the contents of the original input message from SAP in the Environment:
 - a. Right-click the Compute node, and select **Open ESQL**.
 - b. Add the following ESQL:

```
DECLARE ns NAMESPACE 'http://namespace_value';

SET OutputRoot = InputRoot;
SET Environment.OriginalMessage =
```

- InputBody.ns:SapBapiBupaCreateFromDataWrapper.SapBapiBupaCreateFromData;
- Save the SAPSCIFlow.esql file.
 - Connect the Compute node in the message flow as follows:
 - SAPSCI_Operations node (out terminal) → Compute node (in terminal)
7. The next step is to create the mapping between the format of an SAP request and that of a Siebel query. We use a Mapping node to achieve this. Select a Mapping node from the Transformation folder of the node Palette, and drag-and-drop it onto the message flow. Double-click the Mapping node to start the Mapping wizard:
- Expand the Messages folder for both the source and target.
 - From the source panel, Figure 6-77, select **SapBapiBupaCreateFromDataWrapper**.
 - From the target panel, select Figure 6-77, select **EASiebelAdapterIQueryAccountU40PRMANIU41**.

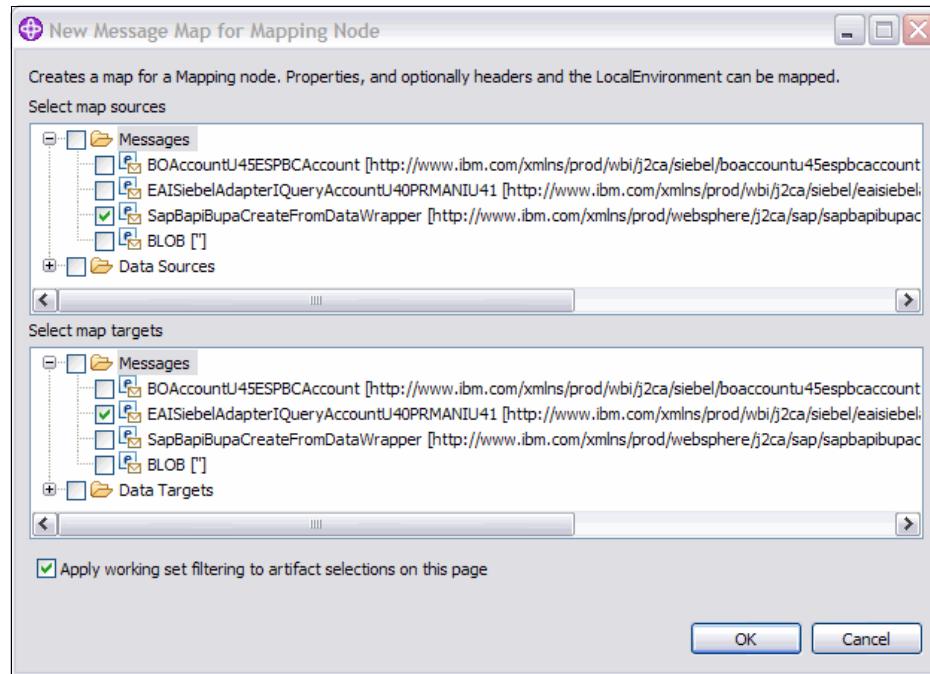


Figure 6-77 Mapping between the SAP input and the Siebel query format

- d. Click **OK** two times.
8. The mapping tool is automatically opened in the WebSphere Message Broker Toolkit. The Properties folders in the source and target panels are mapped by default:
- In the source panel, expand **source** → **sapbapibupacreatefromdatawrapper** → **sapbapibupacreatefromdata** → **sapcentraldataperson**.
 - In the target panel, expand **target** → **eaisiebeladapterqueryaccount** → **SiebelMessage**.
 - Drag-and-drop **FirstNameOfBusinessPartnerPerson** from the source onto the **Name** element in the target, as shown in Figure 6-78.

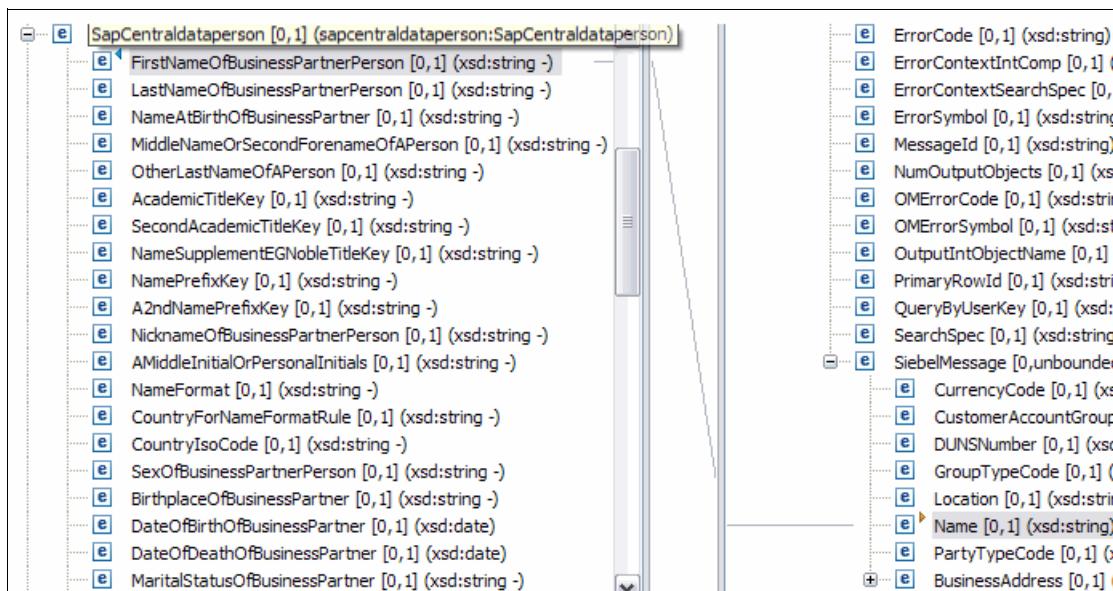


Figure 6-78 Map source *FirstName* to target *Name*

- d. The middle panel of the mapping tool contains the code generated by the mapping of the **FirstNameOfBusinessPartnerPerson** to **Name**. We are going to expand on this code. Add `fn:concat(` before the code and a comma after the end of the code. Drag-and-drop **LastnameOfBusinessPartnerPerson** after the comma, and add a closing parenthesis. The result should look like this:

```
fn:concat($source/sapbapibupacreatefromdatawrapper:SapBapiBupaCreateFromDataWrapper/SapBapiBupaCreateFromData/SapCentraldataperson/FirstNameOfBusinessPartnerPerson,$source/sapbapibupacreatefromda
```

```
tawrapper:SapBapiBupaCreateFromDataWrapper/SapBapiBupaCreateFromD  
ata/SapCentraldataperson/LastNameOfBusinessPartnerPerson)
```

- e. Save the mapping and close the mapping tool.
- f. Connect the Mapping node in the message flow as follows:
 - Compute node (out terminal) → Mapping (in terminal)

We now have the Siebel query format

9. Click the **SiebelQuery.outadapter** component, and drag-and-drop it onto the flow editor. Click **SiebelRequest** node, and in the Siebel Request Node Properties:
 - a. Select the **Request** tab:
 - In the Method location field, type \$LocalEnvironment/DoesNotExist. The SAP adapter information used at the start of the flow is stored at the default location, \$LocalEnvironment/Adapter/MethodName. If you do not change this default value, the adapter will use the information from the start of the flow instead of the current Default method that is listed in the Basic tab, for example, queryEAISiebelAdapterIQuerys.
 - b. Connect the SiebelRequest node in the message flow as follows:
 - Mapping (out terminal) → SiebelRequest (in terminal)
10. Depending on the result of the query to Siebel, the message flow decides whether or not a new account needs to be created in Siebel. We use a Route node to achieve this. Select a Route node from the Routing folder of the node Palette and drag-and-drop it onto the message flow:
 - a. Right-click the **Route** node, and select **Add Output Terminal**.
 - b. Name the Route node **NotExist**, and click **OK**.
 - c. In the Route Node Properties panel, click **Add**.
 - d. Select **NotExist for the Routing output terminal**, and click **Edit**.
 - e. Expand **\$Root**, and click **Add Data Type**.
 - f. Select **EAISiebelAdapterIQueryAccountU40PRMANIU41**, as shown in Figure 6-79 on page 231, and click **OK**.

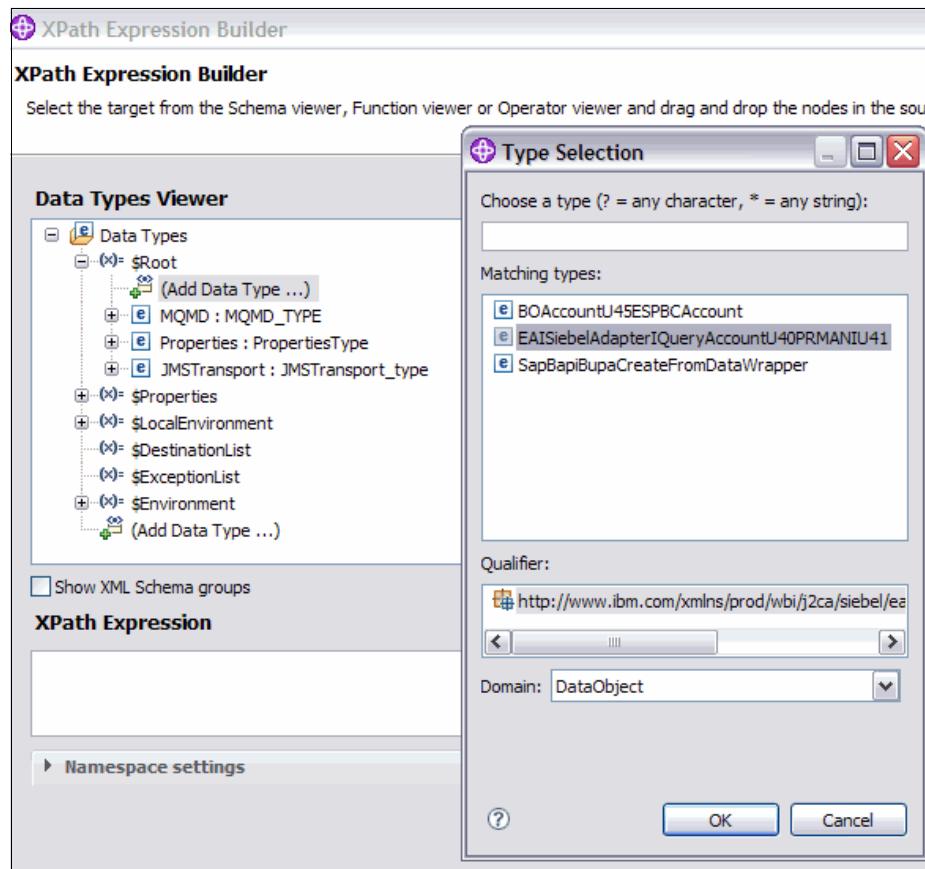


Figure 6-79 Set Data Type in Route node

- g. In the Data Types Viewer panel, expand **easisiebeladapterqueryaccountu40prmaniu41**.
- h. Select **NumOutputObject** and drag-and-drop it into the XPath Expression panel, as shown in Figure 6-80 on page 232.

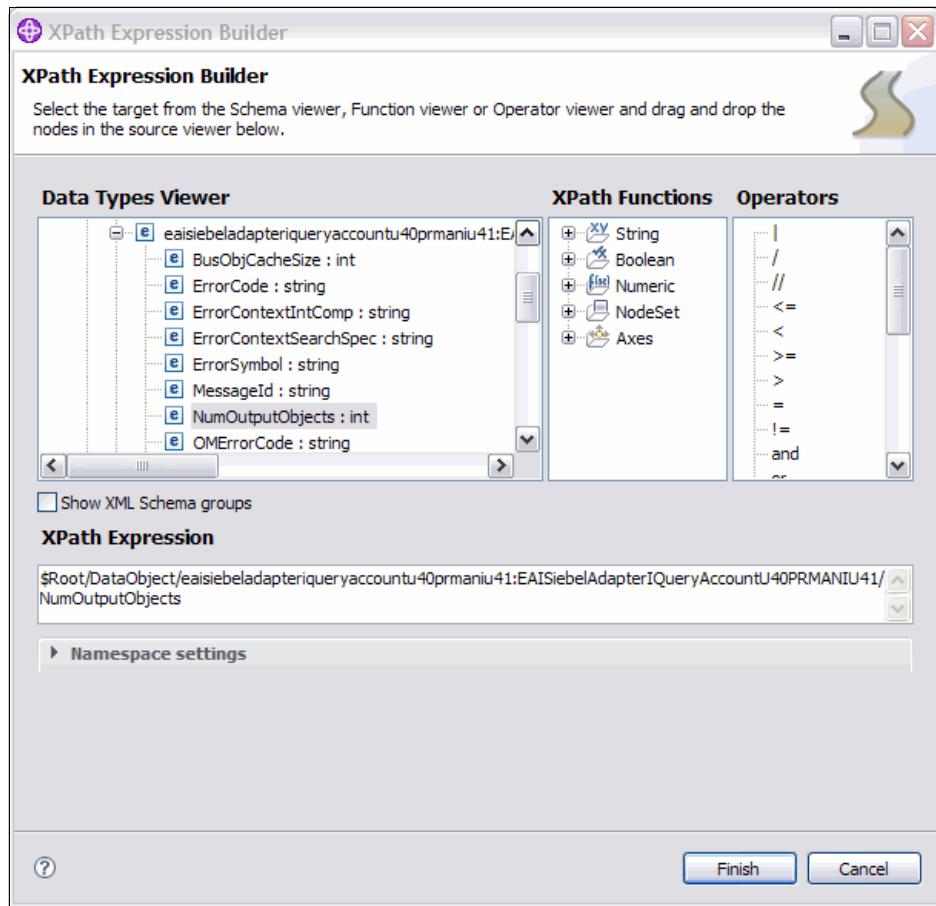


Figure 6-80 XPath Expression Builder in Route node

- i. Add `=0` to the XPath Expression so that the complete expression is:
`$Root/DataObject/eaisiebeladapterqueryaccountu40prmaniu41:EAISiebelAdapterIQueryAccountU40PRMANIU41/NumOutputObjects=0`
- j. Click **Finish**.
- k. On the Add Filter Table Entry panel, click **OK**.
- l. Connect the Route node in the message flow as follows:
 - SiebelRequest (out terminal) → Route node (in terminal).
11. If the result of the Siebel query is `NotExist`, you must create a new account. The original data from the SAP Business Partner was previously stored in the Environment tree using the Compute node. We use a new Compute node to map this data to the Siebel structure. From the Transformation folder of the

node Palette, select a Compute node and drag-and-drop it onto the message flow:

- a. Right-click the **Compute** node, and select **Open ESQL**.

- b. Add the following ESQL:

```
CALL CopyMessageHeaders();  
SET OutputRoot.DataObject.ns4:B0AccountU45ESPBCAccount.Name =  
  
Environment.OriginalMessage.SapCentral1dataperson.FirstNameOfBusinessPartnerPerson||  
  
Environment.OriginalMessage.SapCentral1dataperson.LastNameOfBusinessPartnerPerson;  
SET  
OutputRoot.DataObject.ns4:B0AccountU45ESPBCAccount.CurrencyCode =  
  
Environment.OriginalMessage.SapCentral1dataperson.CountryForNameFormatRule;
```

- c. Save the SAPSCIFlow.esql file.

- d. Connect the Compute1 node in the message flow as follows:

- Route node (NotExist terminal) → Compute1 node (in terminal).

12. Click the **SiebelCreate.outadapter** component and drag-and-drop it onto the flow editor. Click **SiebelRequest1 node**, and in the Siebel Request Node Properties:

- a. Select the **Request** tab, and in the Method location field, type \$LocalEnvironment/DoesNotExist.

- b. Connect the SiebelRequest1 node in the message flow as follows:

- i. Compute1 node (out terminal) → SiebelRequest1 (in terminal)

13. The result of the creation of the Siebel account must be returned to the SAP system. The message flow will return the name of the BP that was created, the currency code, organization, and a string message stating that this BP was created in the Siebel system. We use a Mapping node to convert between the different formats. From the Transformation folder of the node Palette, select a Mapping node and drag-and-drop it onto the message flow. Double-click the Mapping node to start the Mapping wizard:

- a. Expand the **Messages** folder for both the source and target.

- b. From the source panel, shown in Figure 6-81 on page 234, select **BOAccountU45ESPBAccount**.

- c. From the target panel, shown in Figure 6-81 on page 234, select **SapBapiBupaCreateFromDataWrapper**.

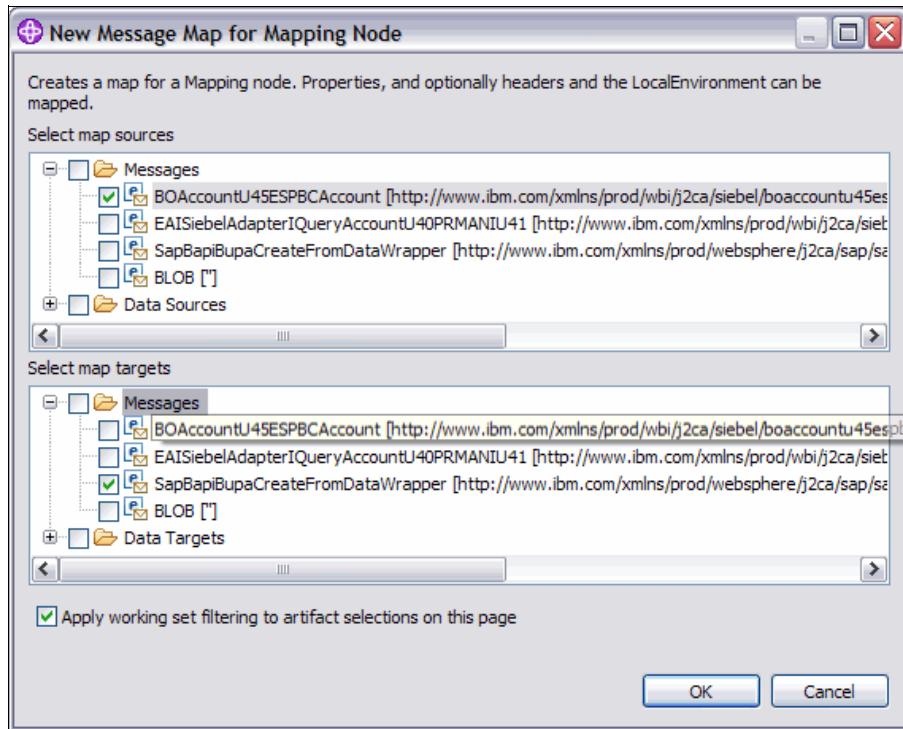


Figure 6-81 Mapping between the Siebel and SAP formats

- d. Click **OK** two times.
14. The mapping tool automatically opens in the WebSphere Message Broker Toolkit. The Properties folders in the source and target panels are mapped by default:
 - a. In the source panel, expand **source** → **boaccount**.
 - b. In the target panel, expand **target** → **sapbapibupacreatefromdatawrapper** → **SapBapiBupaCreateFromData**.
 - c. Drag-and-drop Name from the source onto the **BusinessPartnerNumber1601401696** element in the target panel.
 - d. Expand **target** → **SapReturn**. Map Name to **MessageText**, **CurrencyCode** to the first Message Variable, and **CustomerAccountGroup** to the second MessageVariable, as shown in Figure 6-82 on page 235.
 - e. Right-click the third MessageVariable, and click **Enter Expression**. In the middle panel, type **Created in Siebel**, as shown in Figure 6-82 on page 235.

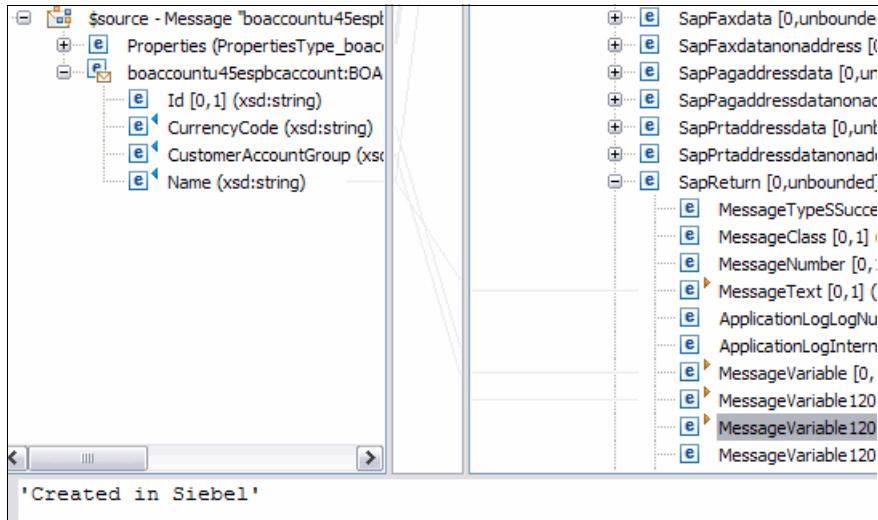


Figure 6-82 Mapping1 node: Mapped elements

- f. Save and close the mapping tool.
- g. Connect the Mapping1 node in the message flow as follows:
 - SiebelRequest1 (out terminal) → Mapping1 (in terminal).
15. From the WebSphere Adapters folder of the node Palette, select a SAPReply node and drag-and-drop it onto the message flow.
 - a. Connect the SAPReply node in the message flow as follows:
 - Mapping1 (out terminal) → SAPReply (in terminal).
16. If the result of the Siebel query is that the Siebel account already exists, the message flow returns the account data to the SAP system. We use a Mapping node to map the response from the SiebelRequest (query) node to the SAP BAPI structure. From the Transformation folder of the node Palette, select a Mapping node and drag-and-drop it onto the message flow. Double-click **Mapping node** to start the Mapping wizard:
 - a. Expand the **Messages** folder for both the source and target.
 - b. From the source panel, select **EASiebelAdapterIQueryAccount**, as shown in Figure 6-83 on page 236.
 - c. From the target panel, select **SapBapiBupaCreateFromDataWrapper**, as shown in Figure 6-83 on page 236.

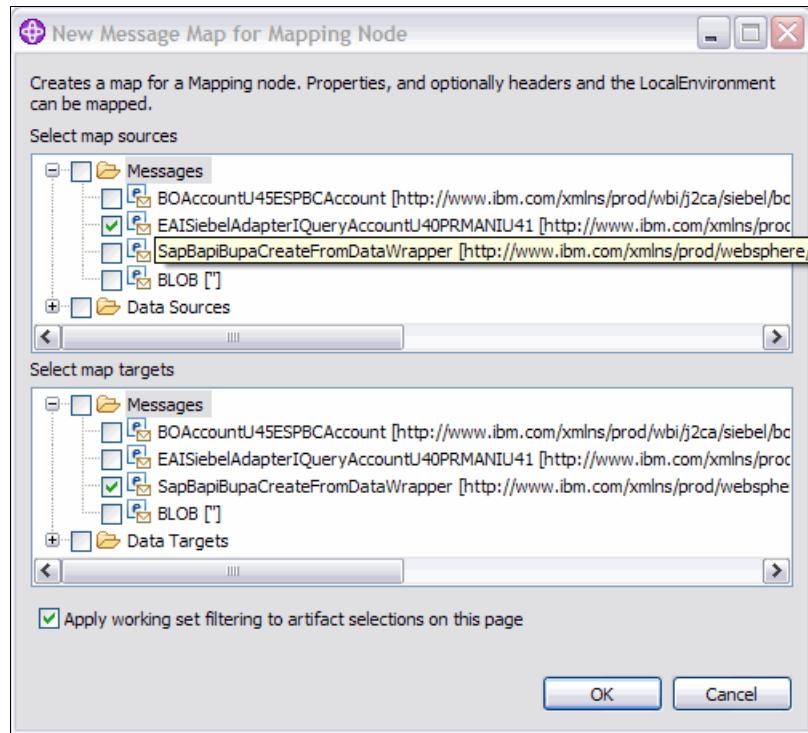


Figure 6-83 Mapping between the Siebel Query and SAP BAPI formats

- d. Click **OK** two times.
17. The mapping tool automatically opens in the WebSphere Message Broker Toolkit. The Properties folders in the source and target panels are mapped by default:
 - a. In the source panel, expand **source** → **easisiebeladapteriqueryaccountu40prmaniu41** → **SiebelMessage**.
 - b. In the target panel, expand **target** → **sapbapibupacreatefromdatawrapper** → **SapBapiBupaCreateFromData**.
 - c. Drag-and-drop Name from the source onto the BusinessPartnerNumber1601401696 element in the target.
 - d. In the target panel, expand **SapReturn**. Map:
 - Name to MessageText
 - CurrencyCode to the first MessageVariable
 - CustomerAccountGroup to the second MessageVariable

- e. Right-click **third MessageVariable**, and click **Enter Expression**. In the middle panel, type Account already exists in Siebel, as shown in Figure 6-84.

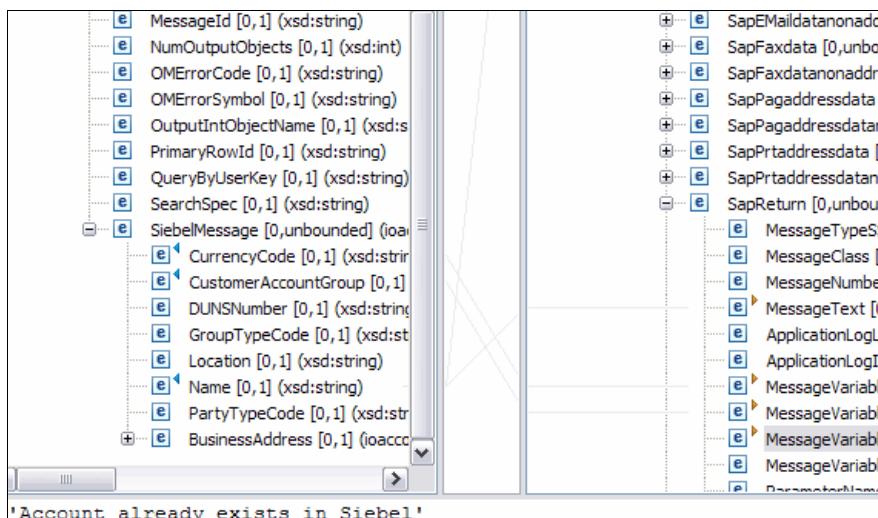


Figure 6-84 Mapping2 node: mapped elements

- f. Save and close the mapping tool.
g. Connect the Mapping2 node in the message flow as follows:
- Route node (Default terminal) → Mapping2 node (in terminal)
 - Mapping2 node (out terminal) → SAPReply (in terminal)

Step 4: Creating and deploying the broker archive file to the broker

To create and deploy the broker archive file to the broker:

1. Right-click **SAPSCI**, and then select **New → Message Broker Archive**.
2. For the name of the archive file, type SAPSCI, and click **Finish**, which opens the Prepare BAR file window.
3. Select all of the components that were created for the SAPSCI scenario:
 - SAPSCIFlow.msgflow
 - SAPSCIMessageSet
 - SiebelMessageSet
 - SAPSCI.inadapter
 - SiebelQuery.outadapter
 - SiebelCreate.outadapter
4. Click **Build broker archive**. When the operation successfully completes, click **OK**, and save the BAR file.

5. Click **SAPSCI.bar** file and drag it onto the WebSphere MQ Explorer taskbar. When the MQ Explorer opens up, drag the BAR file onto the brokers execution group.
6. After the components successfully deploy to the execution group, Figure 6-85, you can test the message flow.

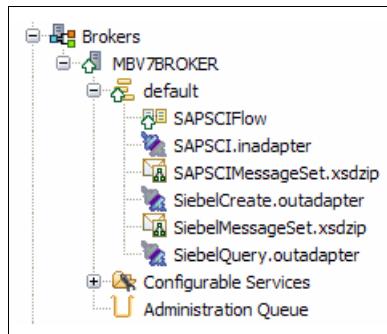


Figure 6-85 Successfully deployed SAPSCI components

Testing the SAPSCIFlow message flow

The test simulates the ABAP program using the BAPI Test tool (se37) and assumes that the create action took place in the SAP system, after which, the event was sent to WebSphere Message Broker. To test the SAPSCIFlow message flow:

1. In the SAP main menu, issue transaction code se37.
2. Type **BAPI_BUPA_CREATE_FROM_DATA** into the Function Module field, as shown in Figure 6-86 on page 239, and press F8.

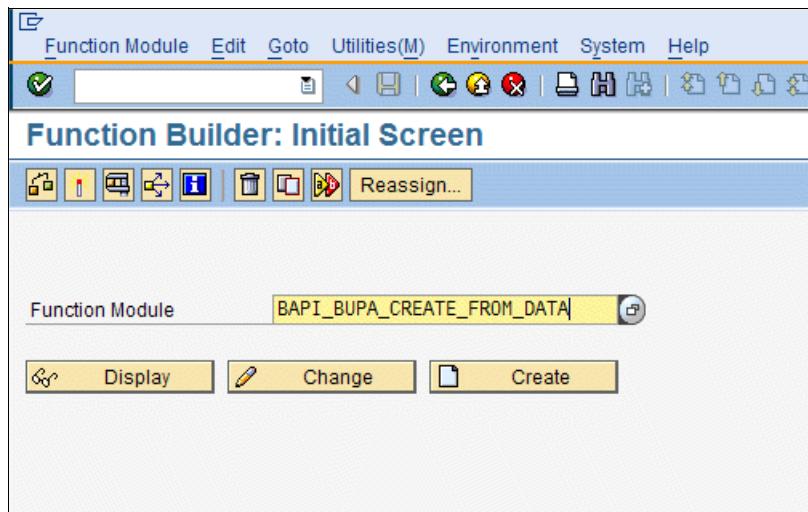


Figure 6-86 SAP Function Builder Initial panel

3. In the Structure Editor window, enter the following data into the specified fields:
 - RFC target sys: BETA02
 - PARTNERCATEGORY: 1
 - Click **CENTRALDATAPERSON**, and enter the following data into the specified fields. Press F3 when these are completed:
 - FIRST NAME: Kirstine
 - LASTNAME: Clapperton
 - NAM: USD (We use the NAM field in this structure to represent CurrencyCode)
4. After all of the data is entered, and you are back in the Test Function Module window, Figure 6-87 on page 240, press F8 to execute.

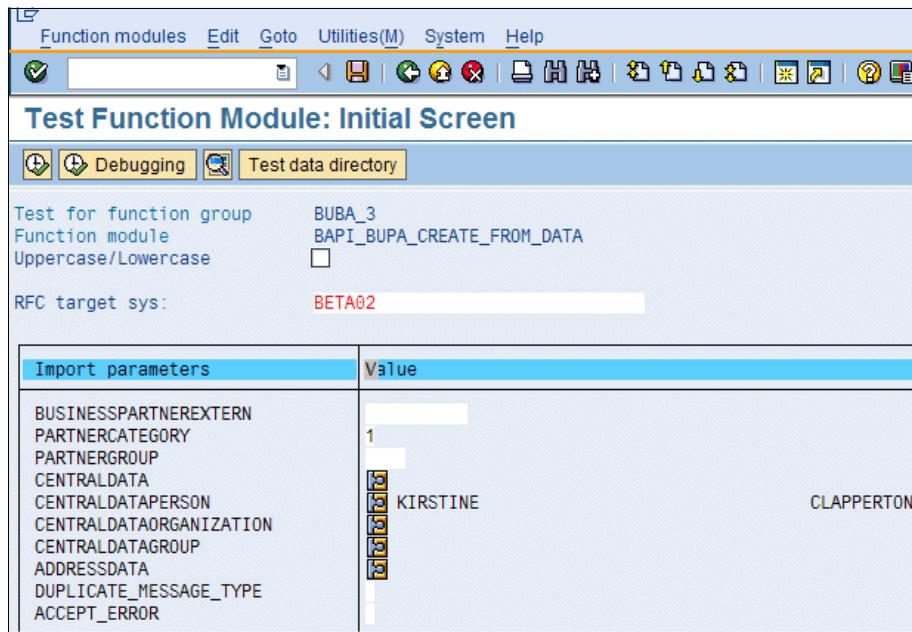


Figure 6-87 SAP Test Function Module panel

- At this point, Kirstine Clapperton does not exist as a Business Partner account in the Siebel system. After the test successfully executes, the SAP Test Function Module will display the return message that was mapped in the third MessageVariable of the SAPReturn element of the Mapping1 node map, for example, Created in Siebel, as shown in Figure 6-88.

MESSAGE_V3
Created in Siebel

Figure 6-88 SAP Return message: *Created in Siebel*

We can log into the Siebel system to view the Account that was just created.

1. In the Siebel Home page, click the **Accounts** tab.
2. Click **All Accounts**, and scroll down to find the Kirstine Clapperton account that was just created. Click the account to open it, as shown in Figure 6-89.

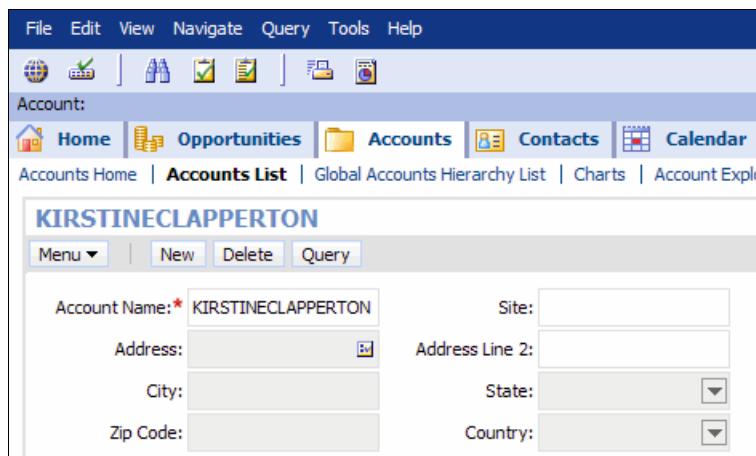


Figure 6-89 The new account created in the Siebel system

Now that the Business Partner account exists in the Siebel system, we can return to the SAP system to test the message flow route when the account already exists:

1. Press F3 twice to return to the Test Function Module, and press F8 to execute it.
2. After the second test successfully executes, the SAP Test Function Module will display the return message that was mapped in the third MessageVariable of the SAPReturn element of the Mapping2 node map, for example, Account already exists in Siebel, as shown in Figure 6-90 on page 242.

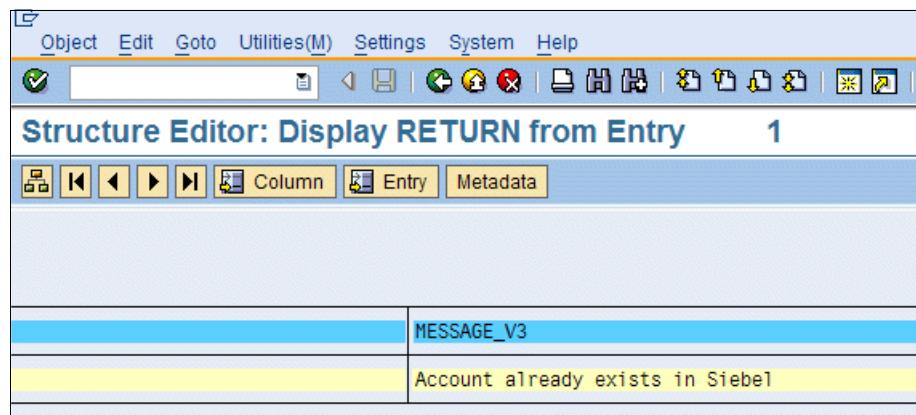


Figure 6-90 SAP Return message: Account already exists in Siebel



Service enable TCP/IP-based applications with WebSphere Message Broker

In this chapter, we review the various TCP/IP nodes that are provided in WebSphere Message Broker and provide scenarios that illustrate different patterns and the flexibility of using these nodes.

Specifically, we cover the following topics in this chapter:

- ▶ “Introduction to TCP/IP-based applications” on page 244
- ▶ “TCP/IP nodes” on page 244
- ▶ “Scenarios” on page 253

7.1 Introduction to TCP/IP-based applications

In this service-oriented architecture(SOA) world, there are a number of applications that transfer data through raw TCP/IP sockets. Like any other applications, these TCP/IP-based applications need to be:

- ▶ Service enabled
- ▶ Governed
- ▶ Decoupled from their clients because many of these applications might go out of service

The TCP/IP nodes that are available in WebSphere Message Broker V6.1.0.2, or later, are designed to address these requirements.

Existing applications that use raw TCP/IP sockets for transferring data can now use the TCP/IP nodes in WebSphere Message Broker to connect directly to the applications. There is no longer a requirement to use WebSphere MQ to enable the applications.

7.1.1 TCP/IP nodes

WebSphere Message Broker implements access to the TCP/IP input and output streams through a series of nodes. There are two sets of TCP/IP nodes: server nodes and client nodes. Both sets have identical function in terms of accessing the data streams. The only difference between them is that one set uses client connections and the other set uses server connections, which means that the difference between them is in establishing the connections, but there is no difference in the way that they use the streams when the connections are established. There are six TCP/IP nodes, which we show in Figure 7-1 on page 245.

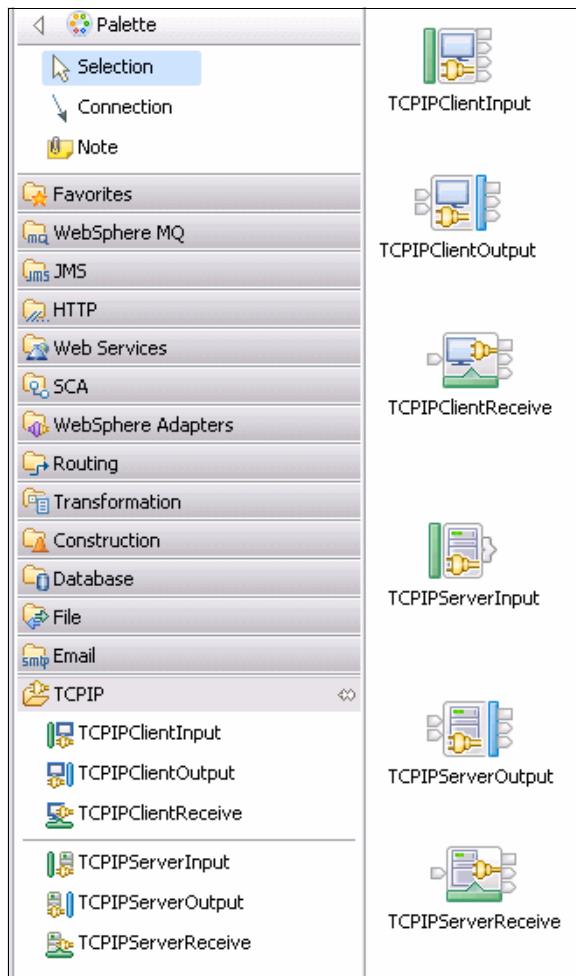


Figure 7-1 TCP/IP nodes

The nodes can be categorized into two sets. One set is made up of four input and output nodes. Because TCP/IP is bi-directional, these nodes correspond to the end points that you have in a bi-directional flow. The other set is made up of two receive nodes. The receive nodes are placed in the middle of a message flow to carry out a GET procedure that is equivalent to an MQGET.

Descriptions of TCP/IP nodes

In this section, we provide a description of the six TCP/IP nodes:

- ▶ **TCPIPClientInput node**

The TCPIPClientInput node creates a client connection to a raw TCP/IP socket and receives data over that connection.

The purpose of the TCPIPClientInput node is to obtain connections to a remote server application that is listening on a TCP/IP port. The connections are not made directly by the node; instead, they are obtained from a connection pool that the WebSphere Message Broker execution group manages. The node requests a client connection that contains data that is ready for reading. Until such a connection is available, the node is paused, waiting for data (in a similar way to the MQInput node).

- ▶ **TCPIPClientOutput node**

The TCPIPClientOutput node creates a client connection to a raw TCP/IP socket and sends data over that connection to an external application.

The purpose of the TCPIPClientOutput node is to obtain connections to a remote server application that is listening on a TCP/IP port. The connections are not made directly by the node but are obtained from a connection pool that the WebSphere Message Broker execution group manages.

The node requests a client connection. If no connections are available for sending data, the output node requests that the pool creates a new connection. When the connection is established, the data is sent. If the data was not sent successfully within the time limit specified by the node's *Timeout sending a data record* property, an exception is thrown.

- ▶ **TCPIPClientReceive node**

The TCPIPClientReceive node receives data over a client TCP/IP connection.

The purpose of the TCPIPClientReceive node is to wait for data to be received on a TCP/IP connection and to retrieve the data. If the connection is closed, an exception is thrown.

- ▶ **TCPIPServerInput node**

The TCPIPServerInput node creates a server connection to a raw TCP/IP socket and receives data over that connection.

The purpose of the TCPIPServerInput node is to listen on a port and when a client socket connects to the port the server socket creates a new connection for the client. Unlike the TCPIPClientInput node, the TCPIPServerInput node does not attempt to make a minimum number of connections. Because the server end of the socket cannot initiate new connections, it can only accept connections.

► **TCPIPServerOutput node**

The TCPIPServerOutput node creates a server connection to a raw TCP/IP socket and sends data over that connection to an external application.

The purpose of the TCPIPServerOutput is to listen on a TCP/IP port and to wait for a client node to make a connection with the port. When the client node connects to the port, the server node creates a new connection for the client. The connections are not made directly by the node but are obtained from a connection pool that the WebSphere Message Broker execution group manages.

When the connection is established, the data is sent. If the data is not sent successfully within the time limit specified by the node's Timeout sending a data record property, an exception is thrown.

► **TCPIPServerReceive node**

The TCPIPServerReceive node receives data over a server TCP/IP connection.

The purpose of the TCPIPServerReceive node is to wait for the data to be received on a TCP/IP connection and to retrieve the data. If the connection is closed, an exception is thrown.

Basic node properties

The Basic properties tab, in the TCP/IP node properties, allows the TCP/IP connection properties to be set, as shown in Figure 7-2.

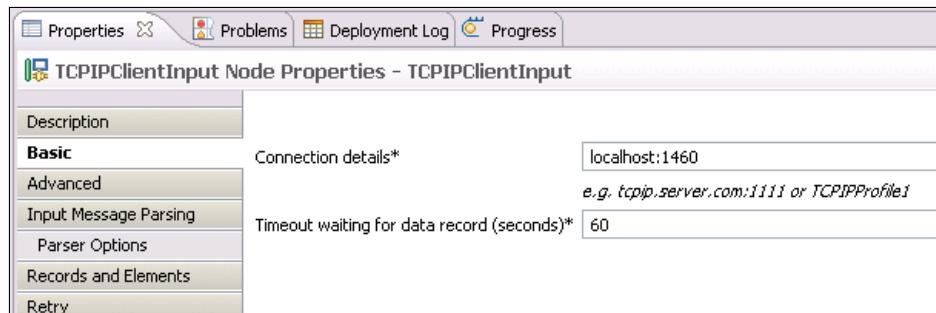


Figure 7-2 Basic Properties of TCP/IP nodes

Use the *Connection details* property to specify either the host name and port number to be used or the name of a configurable service. This property is mandatory. The following formats are supported:

- Configurable service name: This value is used to look up the port and host name in configurable services, for example, TCPIPPProfile1.

- ▶ <Hostname>:<Port>: This value is the host name followed by the port number (separated by a colon), for example, tcpip.server.com:1111.
- ▶ <Port>: This value is the port number. In this case, the host name is assumed to be localhost.

Overriding mechanisms on TCP/IP nodes

The Request tab specifies the location of the data to be written. All properties under the Request tab are mandatory. You can specify the properties on this tab as XPath or ESQL expressions. Figure 7-3 shows the Request properties tab.

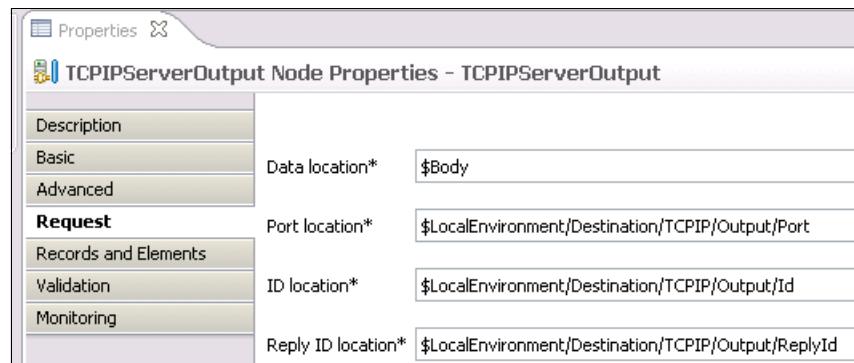


Figure 7-3 Overriding mechanisms on TCP/IP nodes

The following overriding mechanisms can be achieved:

- ▶ Override Port and host name using data in the message or local environment using the *Port location* property
- ▶ Use connection ID from the local environment to make sure that the correct connection is used and is specified using the *ID location* property
- ▶ Set a *Reply ID location* on a connection that is available to any node that uses the same connection later

Record detection

Use the *Records and Elements* tab to specify how the data is interpreted as records. Figure 7-4 on page 249 shows the Record and Elements tab.

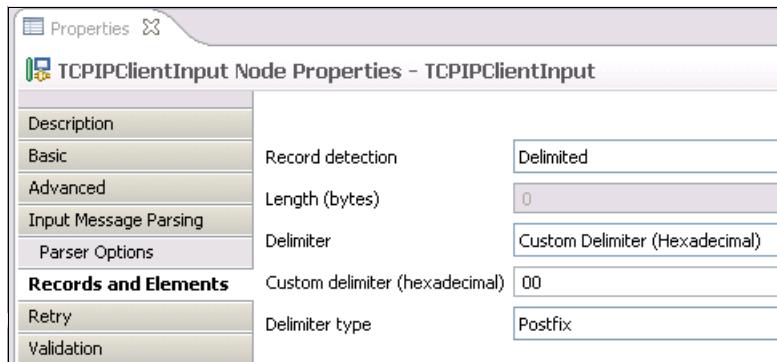


Figure 7-4 Record detection

Use the Record detection property to determine how the data is split into records, each of which generates a single message. Choose from these options:

- ▶ *End of stream* specifies that all of the data that is sent in the data stream is a single record.
- ▶ *Fixed Length* specifies that each record is a fixed number of bytes in length. Each record must contain the number of bytes that are specified in the Length property, except possibly a shorter final record in the file.
- ▶ *Delimited* specifies if the records that you are processing are separated or terminated by a DOS or UNIX® line end or by a sequence of user-defined delimiter bytes. Specify the delimiter and delimiter type in the *Delimiter* and *Delimiter type* properties.
- ▶ *Parsed Record Sequence* specifies if the data contains a sequence of one or more records that are serially recognized by the parser that is specified in the Message domain. The node propagates each recognized record as a separate message. If you select this Record detection option, the parser that is specified in the Message domain must be either XMLNSC or MRM (either CWF or TDS physical format).

For additional information about configuring and using each of the TCP/IP nodes, refer to the following topic in the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.e.tools.mft.doc/ac67360_.htm

7.1.2 Combining TCP/IP nodes

The nodes can be used in the following ways:

- ▶ To connect existing TCP/IP client applications with MQ, use the TCPIPServerInput, TCPIPServerReceive, and TCPIPServerOutput nodes.
- ▶ To connect existing applications to TCP/IP server programs, use the TCPIPClientInput, TCPIPClientReceive, and TCPIPClientOutput nodes.

A TCP/IP connection between two applications has a client end and a server end, which means that one application acts as a server and the other as a client. The terms client and server refer only to the mechanism that is used to establish a connection. They do not refer to the pattern of data exchange. When the connection is established, both client and server can perform the same operations and can both send and receive data.

The TCP/IP nodes were developed to provide full protocol support, including handshakes, request-reply, and many more. These nodes are based on stream technology, exploiting stream-based parsing to interpret stream data as messages because TCP/IP is a stream-based protocol.

Request/Reply pattern

Figure 7-5 shows a simple request/reply interaction pattern.

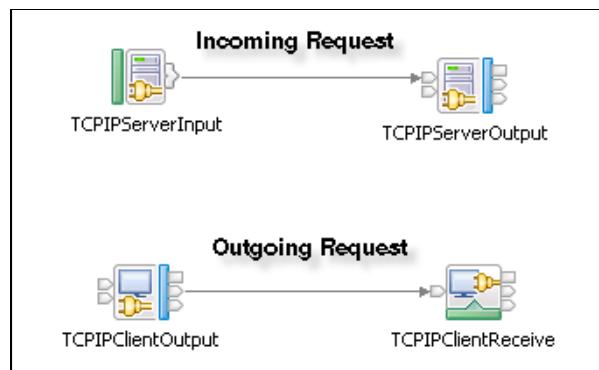


Figure 7-5 Request/Reply pattern

An incoming request shows a server input node receiving data and sending the same data back to the same connection. Other nodes can be inserted between the TCP/IP server input and output node to transform and interact with other transports and systems.

An outgoing request sends data to a client connection and then waits for a response to return from the remote system.

Synchronous and asynchronous TCP/IP request pattern

The routing of TCP/IP messages in a synchronous or asynchronous manner can be achieved using a combination of nodes. Figure 7-6 shows the message flow patterns for routing synchronous and asynchronous TCP/IP requests.

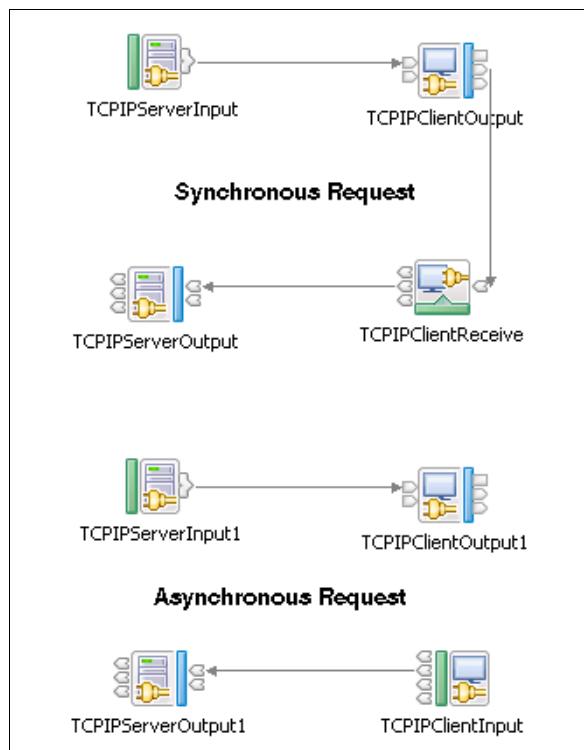


Figure 7-6 Redirect TCP/IP

Synchronous routing is the combination of the incoming server request and outgoing client request from the Request/Reply pattern. It allows the redirecting of TCP/IP traffic where the interaction is always started from the client.

Asynchronous routing is the combination of the incoming server request and outgoing client request but with the client reply received asynchronously through a client input node. It allows the redirecting of TCP/IP traffic where the interaction can be initiated from either side (client-to-server or server-to-client).

Three-way handshake

More complex interactions using TCP/IP nodes in combination with additional nodes can occur where a series of nodes are working on the same connection,

such as a three-way handshake. Figure 7-7 shows a three-way handshake pattern.

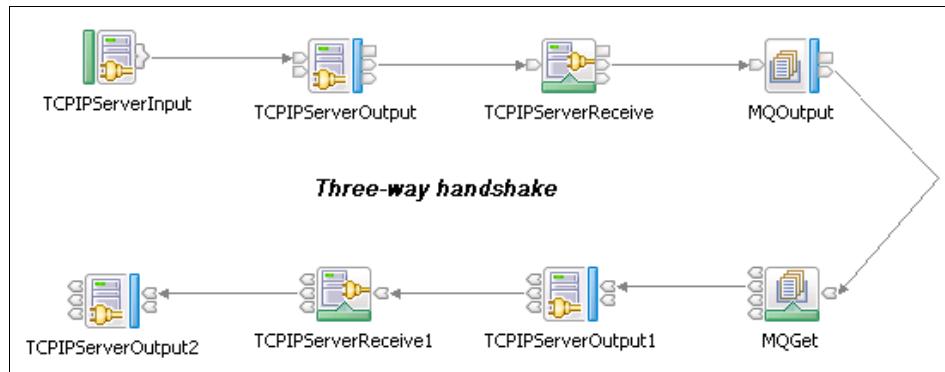


Figure 7-7 Three-way handshake

In Figure 7-7, data is received in the flow on a server connection. The flow sends an acknowledgement to the client and waits for the client to send back a reply to signal that it can continue. The flow then does some processing, in this case it sends an MQ Request/Reply message. When the reply is received, the flow sends the data back to the client and then waits for an acknowledgement from the client. When the acknowledgement is received, the flow sends a reply to tell the client to continue processing.

Countless other complex patterns can be achieved that require arbitrary sending or receiving of data on a connection.

Examples: The following examples illustrate connectivity using TCP/IP nodes:

- ▶ The WebSphere Message Broker samples gallery includes a sample that shows both synchronous and asynchronous communication from the WebSphere Message Broker to a TCP/IP server. In the WebSphere Message Broker Toolkit, select **Help → Samples and Tutorials → WebSphere Message Broker Toolkit → Transports and connectivity → TCPIP Client Nodes sample.**
- ▶ The WebSphere Message Broker samples gallery includes a working example that shows how to implement an application-level handshake protocol for a synchronous request reply model of communication between a client and a server. In the toolkit, select **Help → Samples and Tutorials → WebSphere Message Broker Toolkit → Transports and connectivity → TCPIP Handshake sample.**

7.2 Scenario environment

The scenario was developed in an environment with the following components:

- ▶ WebSphere Message Broker 7.0.0
- ▶ WebSphere MQ 7.0
- ▶ Java version 1.5.0 or later

In the following scenario(s), we assume that the broker was created using the default configuration, which creates the broker and the queue manager with the following names:

- ▶ Broker Name: MB7BROKER
- ▶ Queue Manager Name: MB7QMGR running on port 2414

7.3 Scenarios

In this section, we illustrate various scenarios and usage patterns of using TCP/IP nodes.

7.3.1 Scenario 1: Healthcare System integration using TCP/IP

As with all industries and markets, the integration of data across disparate applications is a key goal within healthcare institutions and providers. The exchange of healthcare-related data among these systems is required both for administrative and for medical reasons.

Many applications within the healthcare industry are based on TCP/IP. Because of the nature of the TCP/IP protocol and the type of data being transmitted, for example, patient movement data, there are a number of specific challenges that must be overcome.

Scenario overview

In a healthcare system, every patient's bed is associated with a patient monitoring device. Each patient monitoring device is connected to its own data collector device. The patient monitoring device sends out the data at regular intervals of time (say every five seconds) to the data collector device. Every minute, each data collector device is expected to combine data from each bed and send out consolidated data to the final console for further use, for example, this data can be used by nurses to take appropriate action.

The data collector device produces data in different formats and uses TCP/IP-based techniques to publish the data. Based on this, some form of data

integration is required to consolidate the various data feeds into a single format that can be sent to the final console.

The key component of the solution is WebSphere Message Broker as a data integrator. It is here that all data is received, collected into a single format, and sent to the final console or presentation layer (example: jviews). The following scenario shows how the WebSphere Message Broker TCP/IP nodes can be added to those systems to generate a more flexible architecture for communication between components.

Figure 7-8 shows the basic flow of the scenario.

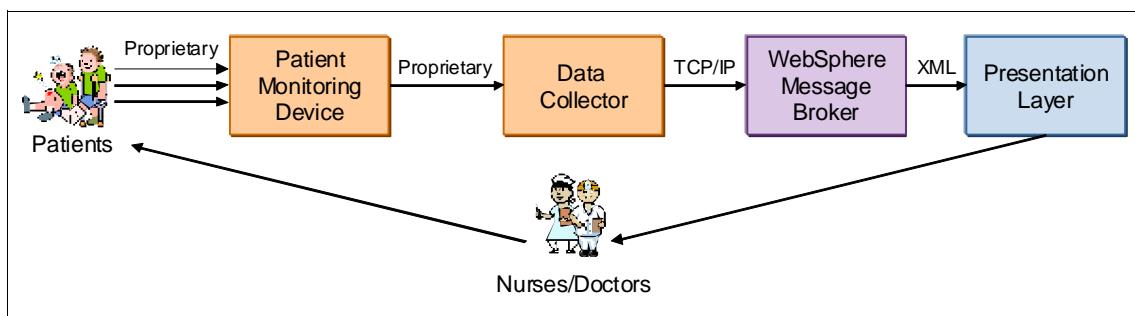


Figure 7-8 Integration with healthcare systems using WebSphere Message Broker

The healthcare scenario requires a TCP/IP connection between the data collector device and the TCP/IP nodes. In this scenario, the data collector device acts as a server and pumps in the data to TCP/IP client input nodes.

Simple server side application

Here is a simple java application that spawns off server applications that send data when connected to by a client. The application acts as a data collector device that pumps in data to WebSphere Message Broker when connected to it as a client. The application takes three input parameters:

- ▶ Port number attached to bed
- ▶ Number of bed applications
- ▶ Pause between sends for each application

Example 7-1 provides the java code for the sample.

Example 7-1 Simple Server side application

```
// ****
// 
// (C) Copyright IBM Corp. 2010 All rights reserved
//
```

```

// ****
import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {

    ServerSocket sSocket;
    int port;
    int pause;
    private static Server receiver;

    public static void main(String[] args) throws Exception {
        int startPort = 2111;
        int serverNumber = 200;
        int pause = 1000;

        if (args.length > 0) {
            startPort = Integer.parseInt(args[0]);
        }
        if (args.length > 1) {
            serverNumber = Integer.parseInt(args[1]);
        }
        if (args.length > 2) {
            pause = Integer.parseInt(args[2]);
        }
        for (int port = startPort; port < (startPort + serverNumber);
port++) {
            receiver = new Server();
            receiver.runSender(port, pause);
        }
        Thread.sleep(10000000);
    }

    public void runSender(int port, int pause) {
        this.port = port;
        this.pause = pause;
        SendData sendData = new SendData();
        Thread thread = new Thread(sendData);
        thread.start();
    }
}

```

```

}

public class SendData implements Runnable {

    BufferedInputStream bis;
    OutputStream os;
    InputStream is;
    InputStreamReader isr;
    BufferedReader br;
    Socket socket;

    public SendData() {

    }

    public void run() {
        while (sSocket == null) {
            try {
                sSocket = new ServerSocket(port);
            } catch (IOException e1) {
                e1.printStackTrace();
            }
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println(port);
        }
        while (true) {
            try {
                try {
                    socket = sSocket.accept();
                    BufferedInputStream bis = new BufferedInputStream(
                        socket.getInputStream());
                    isr = new InputStreamReader(bis);
                    br = new BufferedReader(isr);
                    os = socket.getOutputStream();

                } catch (Exception e) {
                    e.printStackTrace();
                }
                while (true) {
                    String line = "<Message><Text>my bed needs
changing</Text><Number>"
```

```
        + port + "</Number></Message>\n";
    os.write(line.getBytes());
    Thread.sleep(pause);
}
} catch (Exception e) {
    e.printStackTrace();
}
}
}
}
```

Message flow

Figure 7-9 shows the bed monitoring message flow that is used in this scenario.

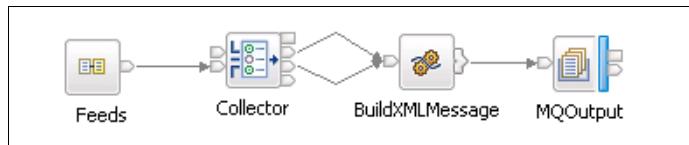


Figure 7-9 BedMonitoring message flow (*BedMonitoring.msgflow*)

Feeds subflow

The feeds subflow contains 10 TCPIPClientInput nodes. Each TCPIPClientInput node connection detail is promoted to the main flow and hence gives the flexibility to change the port numbers based on the availability. The sample included in this scenario uses ports starting from 2900 to 2909. Figure 7-10 on page 258 is a snapshot of the feeds subflow.

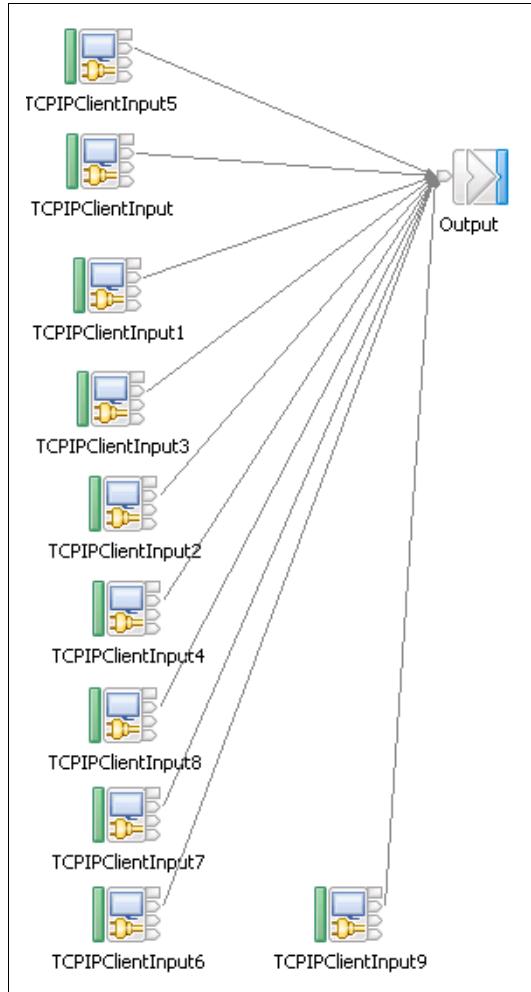


Figure 7-10 Feeds subflow (Feeds.msgflow)

Figure 7-11 on page 259 shows the properties of the feeds subflow that are set in the bed monitoring message flow.



Figure 7-11 Feeds subflow properties

TCPIPCientInput node properties

All 10 of the TCPIPCientInput nodes in the feeds subflow are configured to read the connection details from the main bed monitoring message flow and to read XML messages. Figure 7-12 shows the basic connection details of the TCPIPCientInput nodes of the feeds subflow.

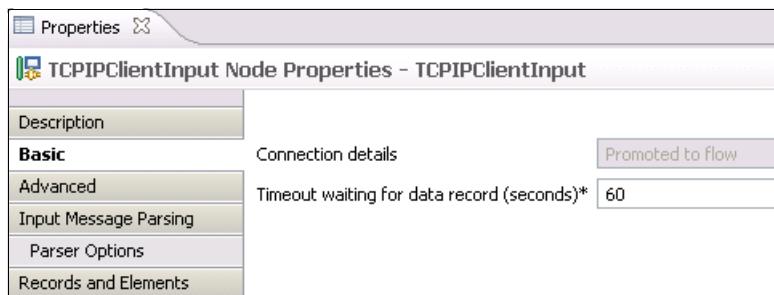


Figure 7-12 TCPIPCientInput node connection details

Figure 7-13 on page 260 shows the Input Message Parsing properties that are set on the TCPIPCientInput nodes of the feeds subflow.

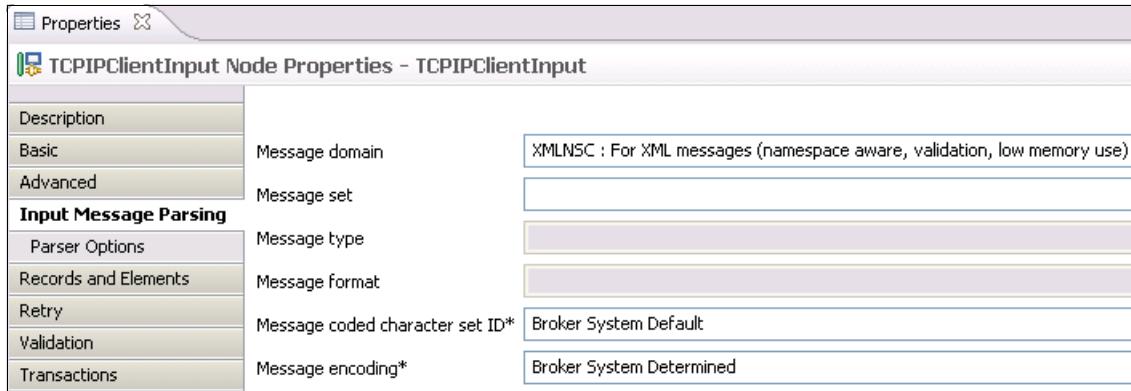


Figure 7-13 TCPIPCientInput node Input Message Parsing properties

Figure 7-14 shows the record detection properties that are set on the TCPIPCientInput nodes of the feeds subflow. The input messages are delimited and ended by a DOS or UNIX line end.

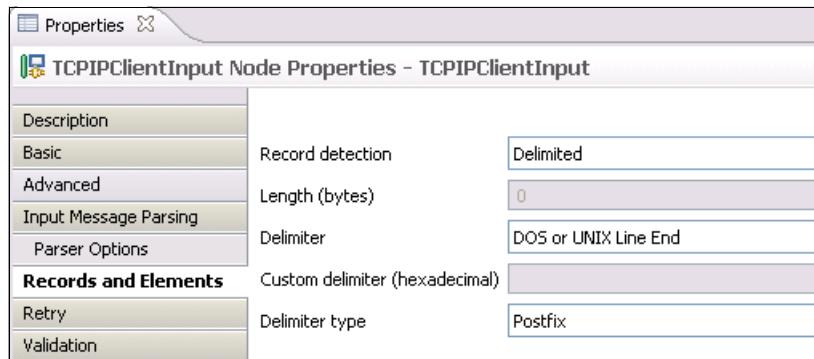


Figure 7-14 TCPIPCientInput node records and elements properties

All other properties of the TCPIPCientInput nodes of the feeds subflow are set to their defaults.

Collector node

When the message flow runs, the Collector node gathers messages from its IN terminal into message collections. As each collection completes at the end of five seconds, it is propagated to the BuildXMLMessage compute node.

Figure 7-15 on page 261 shows the Collector node basic properties. All other properties of the Collector node of the Bed Monitoring message flow are set to their defaults.

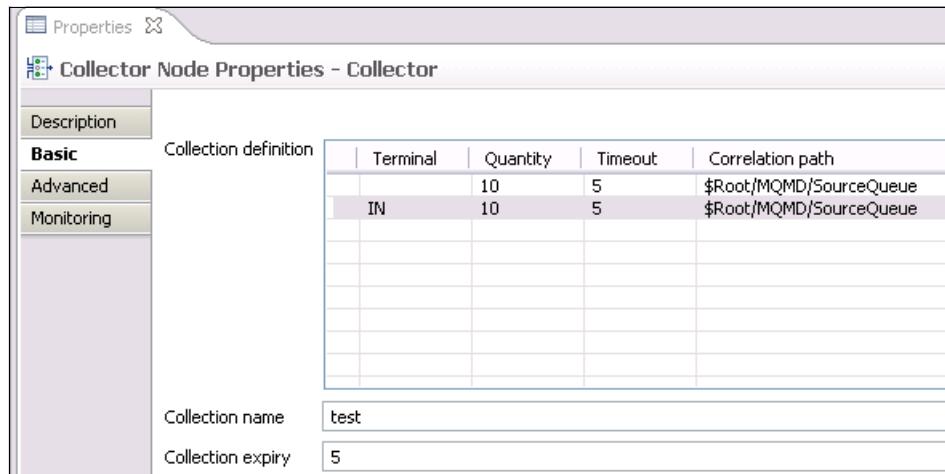


Figure 7-15 Collector node basic properties

BuildXMLMessage compute node

The compute node examines all responses that are sent back, builds a single XML output message, and makes sure that there is only one record for each port in the final message. It then propagates the message to the queue, COMBINED.MESSAGING.OUT.

Example 7-2 gives the necessary ESQL.

Example 7-2 BuildXMLMessage ESQL

```
CREATE COMPUTE MODULE BuildXMLMessage
  CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
      DECLARE card INTEGER cardinality(InputBody.IN[]);
      DECLARE i INTEGER 1;
      DECLARE j INTEGER 2900;
      WHILE j <= 2909 DO
        SET OutputRoot.XMLNSC.Top.Beds.Bed[j-2899].Number = j;
        SET OutputRoot.XMLNSC.Top.Beds.Bed[j-2899].Text    = '';
        SET j = j +1;
      END WHILE;
      WHILE i <= card DO
        SET
          OutputRoot.XMLNSC.Top.Beds.Bed[cast(InputBody.IN[i].XMLNSC.Message.Number AS INTEGER)-2899].Text = InputBody.IN[i].XMLNSC.Message.Text;
        SET i = i +1;
      END WHILE;
    END;
```

```
    RETURN TRUE;  
END;  
END MODULE;
```

MQOutput Node

The MQOutput node puts the message that is received from the compute node to the COMBINED.MESSAGE.OUT queue. Figure 7-16 shows the Basic properties tab of MQOutput node.

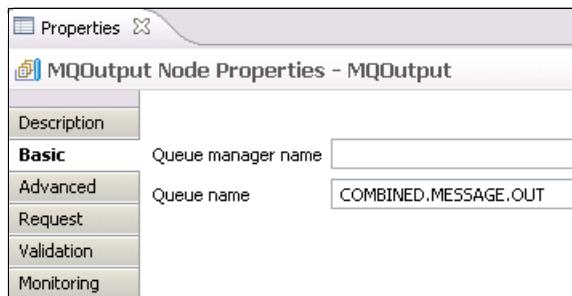


Figure 7-16 MQOutput node properties

All other properties of the MQOutput node are set to their defaults.

Testing the scenario

Refer to Appendix A, “Additional material” on page 533 to download the project interchange file BedMonitoringApplication.zip and the jar file bedapp.jar, which are part of SG247826.zip file. To test the scenario:

1. Copy the bedapp.jar to your local directory.
2. Create the local queue COMBINED.MESSAGE.OUT on MB7QMGR.
3. Import the project interchange file BedMonitoringApplication.zip file into the workspace.
4. Expand **BedMonitoringApplication** → **Broker Archives** → **default broker schema**.
5. Deploy the broker archive file bedmonitoringapp.bar to the MB7BROKER broker.
6. Open a command prompt, and run the Server application with the port number, number of bed applications, and pause between sends for each application (in milliseconds) as input parameters.

Example command: java -classpath <Location of the jar file>\bedapp.jar Server 2900 10 2000

7. Check the queue COMBINED.MESSAGE.OUT for the output messages.
Example 7-3 shows sample Input messages used by the scenario coming from ten different ports with the starting port 2900 and ending port 2909.

Example 7-3 Sample Input messages

```
<Message><Text>my bed needs  
changing</Text><Number>2900</Number></Message>  
<Message><Text>my bed needs  
changing</Text><Number>2901</Number></Message>  
<Message><Text>my bed needs  
changing</Text><Number>2902</Number></Message>  
<Message><Text>my bed needs  
changing</Text><Number>2903</Number></Message>  
<Message><Text>my bed needs  
changing</Text><Number>2904</Number></Message>  
<Message><Text>my bed needs  
changing</Text><Number>2905</Number></Message>  
<Message><Text>my bed needs  
changing</Text><Number>2906</Number></Message>  
<Message><Text>my bed needs  
changing</Text><Number>2907</Number></Message>  
<Message><Text>my bed needs  
changing</Text><Number>2908</Number></Message>  
<Message><Text>my bed needs  
changing</Text><Number>2909</Number></Message>
```

Sample Output message for the above input messages:

```
<Top>  
<Beds>  
<Bed><Number>2900</Number><Text>my bed needs changing</Text></Bed>  
<Bed><Number>2901</Number><Text>my bed needs changing</Text></Bed>  
<Bed><Number>2902</Number><Text>my bed needs changing</Text></Bed>  
<Bed><Number>2903</Number><Text>my bed needs changing</Text></Bed>  
<Bed><Number>2904</Number><Text>my bed needs changing</Text></Bed>  
<Bed><Number>2905</Number><Text>my bed needs changing</Text></Bed>  
<Bed><Number>2906</Number><Text>my bed needs changing</Text></Bed>  
<Bed><Number>2907</Number><Text>my bed needs changing</Text></Bed>  
<Bed><Number>2908</Number><Text>my bed needs changing</Text></Bed>  
<Bed><Number>2909</Number><Text>my bed needs changing</Text></Bed>  
</Beds>  
</Top>
```

Extending the sample

The sample can be extended to read data from more than 10 bed monitoring applications or servers. To read the data from multiple servers, include multiple feeds subflow in the main flow and configure the port numbers accordingly. The ESQL code in the sample assumes that the TCP/IP ports are in sequential order and hence might need modifying according to the business requirements.

Additional information

Also refer to the healthcare sample in the WebSphere Message Broker Samples Gallery. The sample that is shipped with the WebSphere Message Broker V7 toolkit is a general healthcare sample for HL7 with support for a wide-range of functionalities, whereas the scenario that we explain in this section is based on the story bound scenario.

Example: The following example addresses common problems that are associated with processing healthcare application messages over TCP/IP:

The WebSphere Message Broker samples gallery includes the Healthcare sample as a development accelerator, decreasing the time an integration developer in the healthcare industry requires to deliver integration solutions. This sample provides a number of healthcare processing assets that solve key healthcare-specific integration problems. In the WebSphere Message Broker Toolkit, select **Help → Samples and Tutorials → WebSphere Message Broker Toolkit → Industry → Healthcare sample**.

Many applications in the healthcare industry are based on the Internet Protocol network, specifically Minimal Lower Layer Protocol (MLLP), which enables Health Level 7 (HL7) messages to be transmitted over the Internet Protocol network. Because of the nature of the Internet Protocol network and the type of data that is being transmitted, that is, patient movement data, a number of specific challenges must be overcome, for example, the sequence of the messages, acknowledging the received messages, and the ability to detect duplicated messages that are passing through the system. This sample demonstrates how to solve the aforementioned problems that are associated with processing messages over the MLLP Internet Protocol network.

7.3.2 Scenario 2: Telnet to WebSphere MQ

Scenario 2 explains simple telnet to a WebSphere MQ queue using TCP/IP nodes. Figure 7-17 on page 265 shows the basic architecture of the scenario.

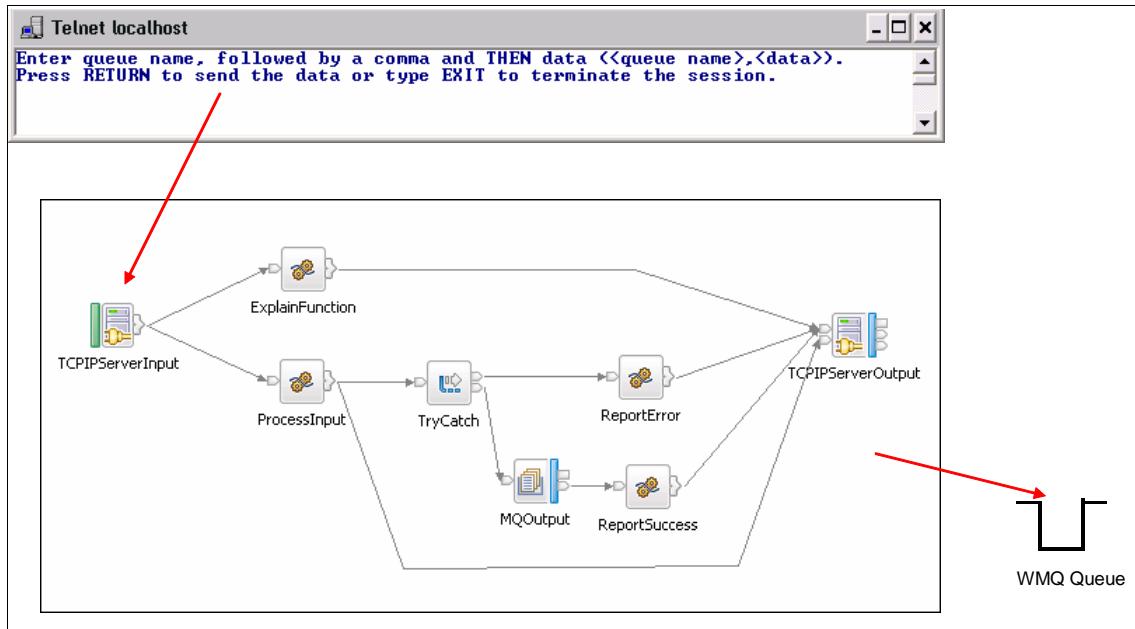


Figure 7-17 Telnet to WebSphere MQ using TCP/IP nodes

Figure 7-17 shows the following exchange of messages between a telnet client and a server: WebSphere Message Broker flow:

1. A telnet client starts the session by connecting to WebSphere Message Broker flow.
2. The server accepts the request from the client.
3. The client sends the queue name and data.
4. The server puts the data to a WebSphere MQ queue if the queue exists.
5. The server reports the success/failure of the MQPUT operation to the client.

Message flow

Figure 7-18 on page 266 shows the message flow, which acts as a server and accepts and rejects the request from the telnet client.

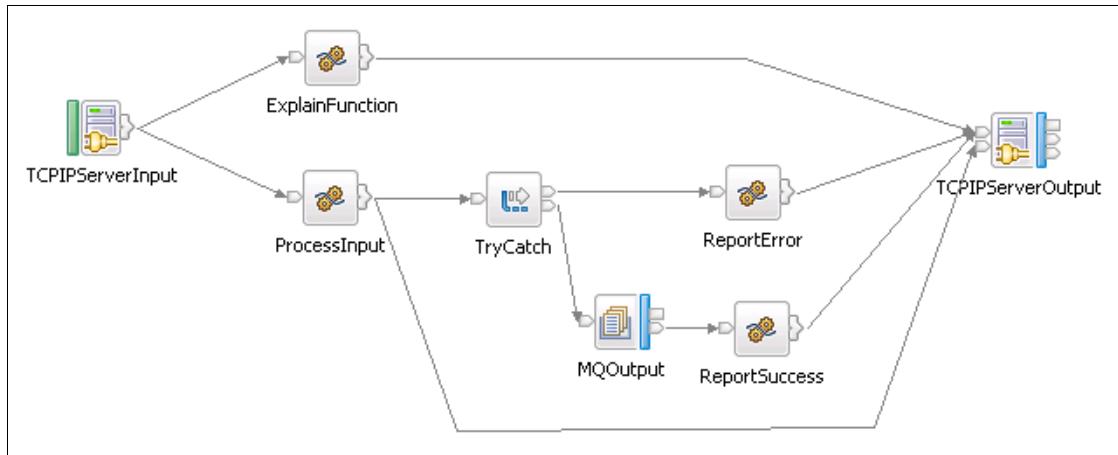


Figure 7-18 Telnet to WebSphere MQ message flow (*Telnet_to_Queue.msgflow*)

Message flow nodes

In this section, we explain the different nodes that are used in the Telnet to WebSphere MQ message flow.

TCPIPServerInput node

The TCPIPServerInput node listens on port 1460, and when a client socket connects to the port, the server socket creates a new connection for the client and reads the data. The output terminal 'Open' of the TCPIPServerInput node is connected to the ExplainFunction compute node to which a message is routed when it is first opened. The 'Open' terminal receives an event when the telnet client connects, and then the ExplainFunction compute node writes the message to the telnet client.

Further messages from the telnet client are routed to the 'Out' terminal of the TCPIPServerOutput node. Figure 7-19 shows the basic properties that are set on TCPIPServerInput node.

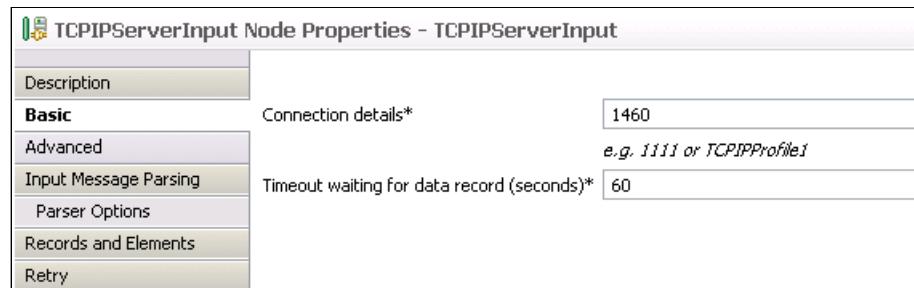


Figure 7-19 TCPIPServerInput node basic properties

The TCPIPServerInput node reserves the input stream until the end of the flow. That is until the telnet client is connected to the server and releases the input stream when the telnet client disconnects. Figure 7-20 shows the Input Stream properties of the TCPIPServerInput node.

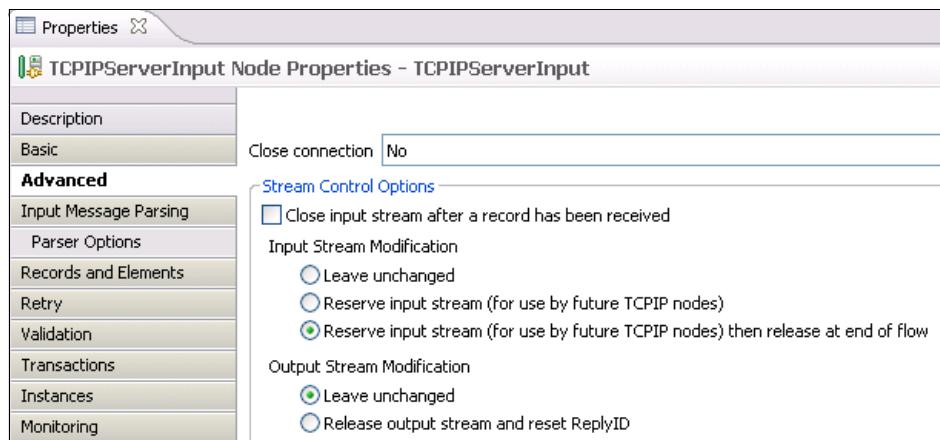


Figure 7-20 TCPIPServerInput node Input Stream properties

All other properties of the TCPIPServerInput node are set to their defaults.

TCPIPServerOutput node

The TCPIPServerOutput node creates a server connection to a raw TCP/IP socket and sends data over that connection to the telnet client. Figure 7-21 shows the basic properties that are set on the TCPIPServerOutput node.

Description		
Basic	Connection details*	1460
Advanced		e.g. JJJJJ or TCPJPPProfile
Request		Timeout sending data record (seconds)*
Records and Elements		60

Figure 7-21 TCPIPServerOutput node Basic properties

All other properties of the TCPIPServerOutput node are set to their defaults.

ExplainFunction compute node

The ExplainFunction Compute node sends back the reply to the client to enter the queue name and data to be put to the queue. It also explains how to continue sending data through the telnet client and how to exit the telnet client session. Example 7-4 on page 268 provides the necessary ESQL code.

Example 7-4 ExplainFunction ESQL

```
CREATE COMPUTE MODULE ExplainFunction
  CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
      DECLARE tempBlob BLOB;
      SET tempBlob = CAST('Enter queue name, followed by a comma and
THEN data (<queue name>,<data>).' AS BLOB CCSID 1208) ||
                    x'0d0a' ||
                    CAST('Press RETURN to send the data or type EXIT to terminate the
session.' AS BLOB CCSID 1208);
      SET OutputRoot.BLOB.BLOB = tempBlob;
      RETURN TRUE;
    END;
END MODULE;
```

ProcessInput compute node

The ProcessInput compute node parses the data from the telnet client to get the queue name and the data to be put to the queue.

As shown in Example 7-5, the following ESQL code sets the Destination queue name in the LocalEnvironment, which will be read by the MQOutput node:

```
SET
InputLocalEnvironment.Destination.MQ.DestinationData[1].queueName[1]
= TRIM(SUBSTRING(temp FROM 0 FOR I));
```

The ESQL code in Example 7-5 sets the data into OutputRoot as BLOB data:

```
SET OutputRoot.BLOB.BLOB = CAST(SUBSTRING(temp FROM I+1) AS BLOB
CCSID 1208);
```

Example 7-5 ProcessInput ESQL

```
CREATE COMPUTE MODULE ProcessInput
  CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
      DECLARE temp CHAR;
      SET temp = CAST(InputRoot.BLOB.BLOB AS char CCSID 1208);
      DECLARE I INTEGER;
      SET I = POSITION('EXIT' IN TRIM(UPPER(temp)));
      IF I = 1 THEN
        propagate TO TERMINAL 1;
      ELSE
        SET I = POSITION(',') IN temp);
```

```

        SET
InputLocalEnvironment.Destination.MQ.DestinationData[1].queueName[1] =
TRIM(SUBSTRING(temp FROM 0 FOR I));
        SET OutputRoot.BLOB.BLOB = CAST(SUBSTRING(temp FROM I+1) AS
BLOB CCSID 1208);
        RETURN TRUE;
    END IF;
    RETURN false;
END;
END MODULE;

```

The data from the ProcessInput compute node is propagated to its 'OUT' terminal when the data has to be put to the queue. It is then propagated to its 'OUT1' terminal when the telnet client exits. Hence the 'OUT1' terminal of the ProcessInput compute node is wired to close the terminal of the TCPIPServerOutput node.

ReportSuccess compute node

The ReportSuccess compute node reports to the telnet client that the data was put to the queue successfully. Example 7-6 is the necessary ESQL.

Example 7-6 ReportSuccess ESQL

```

CREATE COMPUTE MODULE ReportSuccess
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        DECLARE temp BLOB;
        SET temp = CAST('Sent data: ' AS BLOB CCSID 1208) ||
InputRoot.BLOB.BLOB || CAST(' to queue '''' ||
InputLocalEnvironment.Destination.MQ.DestinationData[1].queueName[1] ||
'''' AS BLOB CCSID 1208)||
x'0a0d' ||

CAST('=====
===== AS BLOB CCSID 1208);
        SET OutputRoot.BLOB.BLOB = temp;
        RETURN TRUE;
    END;
END MODULE;

```

ReportError compute node

The ReportError compute node reports to the telnet client that the broker failed to put the data to the requested queue name. Example 7-7 shows the necessary ESQL.

Example 7-7 ReportError ESQL

```
CREATE COMPUTE MODULE ReportError
    CREATE FUNCTION Main() RETURNS BOOLEAN
        BEGIN
            DECLARE temp BLOB;
            SET temp = CAST('Failed to send data to: ' AS BLOB CCSID 1208) ||
            InputRoot.BLOB.BLOB ||
            CAST(' to queue ' ||
InputLocalEnvironment.Destination.MQ.DestinationData[1].queueName[1] AS
BLOB CCSID 1208) ||
            x'0a0d' ||

            CAST('=====
=====' AS BLOB CCSID 1208);
            SET OutputRoot.BLOB.BLOB = temp;
            RETURN TRUE;
        END;
    END MODULE;
```

TryCatch node

The TryCatch node provides a special handler for exception processing. Initially, the message that is received from the ProcessInput compute node is routed on the Try terminal, which is connected to the MQOutput node. If a MQOutput node throws an exception failing to put data to the queue, the TryCatch node catches it and routes the original message to its Catch terminal. The Catch terminal is connected to the ReportError compute node to report the appropriate message to the telnet client.

MQOutput node

The MQOutput node puts a message to a specific WebSphere MQ queue destination that is identified in the LocalEnvironment, which is also known as the DestinationList that is associated with the message.

As shown in Figure 7-22 on page 271, MQOutput node reads the queue name from the Destination List.

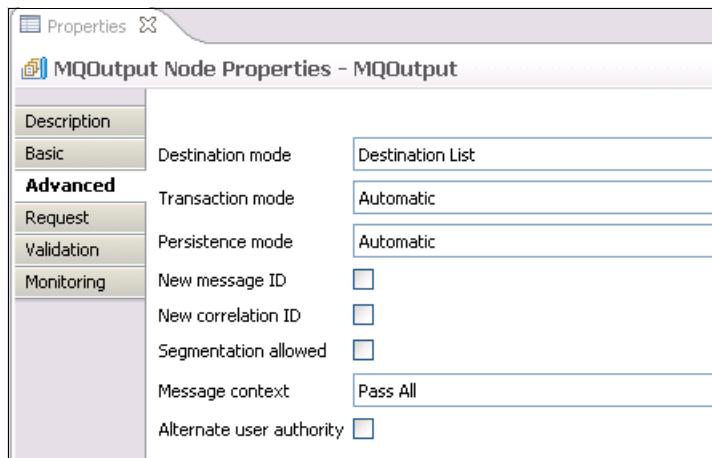


Figure 7-22 MQOutput Node properties

All of the other properties of the MQOutput node are set to their defaults.

Testing the scenario

Refer to Appendix A, “Additional material” on page 533 to download the project interchange file TelnetToWMQ.zip, which is part of the sg247826.zip file:

1. Import the project interchange file TelnetToWMQ.zip into the WebSphere Message Broker V7 toolkit using **File → Import → Other → Project Interchange**, and browse to the project TelnetToWMQ.
2. Expand **TelnetToWMQ → Broker Archives → default broker schema**.
3. Deploy the broker archive file telnettoqueue.bar to the MB7BROKER broker.
4. In a command prompt, type `telnet localhost 1460` to telnet to the broker TCP/IP port, which opens up a telnet session, as shown in Figure 7-23.

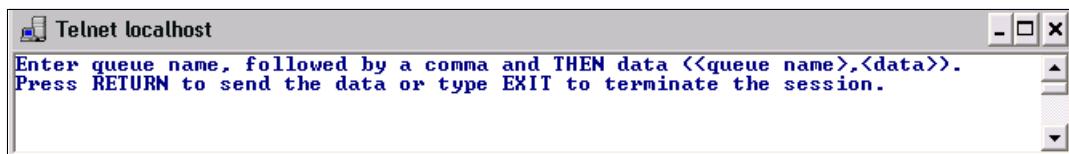


Figure 7-23 Telnet to localhost

5. Create the local queue TEST.OUT on MB7QMGR.
6. Enter TEST.OUT, Testing telnet to WMQ to put the text Testing telnet to WMQ to queue TEST.OUT. Figure 7-24 on page 272 shows the output on the telnet client when the data is put to the queue successfully.

```
Telnet localhost
Enter queue name, followed by a comma and THEN data <<queue name>,<data>>.
Press RETURN to send the data or type EXIT to terminate the session.
TEST.OUT,Testing telnet to WMQ
Sent data: Testing telnet to WMQ to queue 'TEST.OUT'
=====
```

Figure 7-24 Telnet to WebSphere MQ queue success case

7. Delete the local queue TEST.FAIL on the MB7QMGR, if it exists.
8. Enter TEST.FAIL,Testing report error. The telnet client reports the failure, as shown in the Figure 7-25.

```
Telnet localhost
Enter queue name, followed by a comma and THEN data <<queue name>,<data>>.
Press RETURN to send the data or type EXIT to terminate the session.
TEST.OUT,Testing telnet to WMQ
Sent data: Testing telnet to WMQ to queue 'TEST.OUT'
=====
TEST.FAIL,Testing report error
Failed to send data to: Testing report error to queue TEST.FAIL
=====
```

Figure 7-25 Telnet to WebSphere MQ queue Failure case

7.3.3 Scenario 3: Configurable Services for TCP/IP nodes

Configurable services are typically runtime properties. You can use them to define properties that are related to external services on which the broker relies, for example, a TCP/IP server port or a JMS provider.

Instead of defining properties on the node or message flow, you can create configurable services so that nodes and message flows can refer to them for properties at run time. If you use this method, you can change the value of attributes for a configurable service on the broker, which then affects the behavior of a node or message flow without the need for redeployment.

Scenario 3 explains how to create configurable services for the TCP/IP nodes using commands and using WebSphere Message Broker explorer.

There are two configurable services that are associated with TCP/IP nodes:

- ▶ TCPIPServer: Details of TCP/IP connections for a server connection.
- ▶ TCPIPClient: Details of TCP/IP connections for a client connection.

Note: Ensure that the broker MB7BROKER is running. If it is not, use the `mqsi start` command to start it.

In the following sections, we provide the list of commands to create, report, and update TCP/IP server and client configurable services.

Creating a TCPIPServer configurable service

The command in Example 7-8 creates a TCPIPServer configurable service named TCPIPServerPort, as highlighted in Figure 7-26.

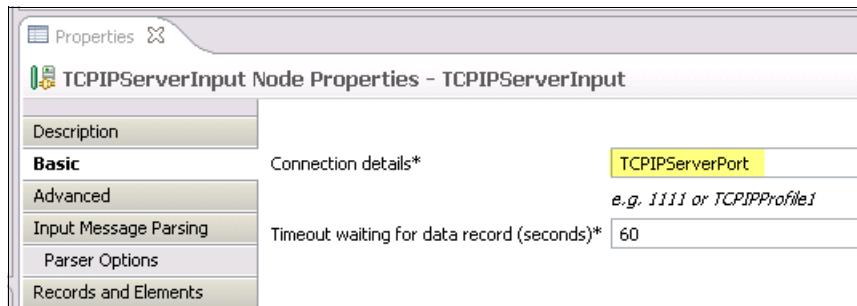


Figure 7-26 TCPIPServer configurable service

The command in Example 7-8 creates a TCPIPServer configurable service, TCPIPServerPort for port 1460, which allows 10000 connections and expires any connection if the data is not received within 20 seconds.

Example 7-8 TCPIPServer configurable service

```
mqsicreateconfigurableservice MB7BROKER -c TCPIPServer -o  
TCPIPServerPort -n ExpireConnectionSec,MaximumConnections,Port -v  
20,10000,1460
```

Figure 7-27 shows the result of the successful command completion.

```
Select IBM WebSphere Message Broker 7.0  
C:\Program Files\IBM\MQSI\7.0>mqsicreateconfigurableservice MB7BROKER -c TCPIPServer -o TCPIPServerPort -n ExpireConnectionSec,MaximumConnections,Port -v 20,10000,1460  
BIP8071I: Successful command completion.
```

Figure 7-27 Create TCPIPServer configurable service

Report all TCPIPServer configurable services

The command shown in Example 7-9 displays all of the properties that are associated with the TCPIPServer configurable service.

Example 7-9 TCPIPServer configurable service

```
mqsi reportproperties MB7BROKER -c TCPIPServer -o
AllReportableEntityNames -r
```

Figure 7-28 shows the result of the successful command completion.

The screenshot shows a terminal window titled "IBM WebSphere Message Broker 7.0". The command entered is "C:\Program Files\IBM\MQSI\7.0>mqsi reportproperties MB7BROKER -c TCPIPServer -o AllReportableEntityNames -r". The output displays configuration properties for the TCPIPServer and TCPIPServerPort. For the TCPIPServer, properties include: Default, ExpireConnectionSec='-1', MaxReceiveRecordBytes='1000000000', MaximumConnections='100', Port='0', SO_KEEPALIVE='false', SO_LINGER='false', SO_LINGER_TIMEOUT_SEC='-1', SO_RCVBUF='0', SO_SNDBUF='0', TCP_NODELAY='false', TrafficClass='-1'. For the TCPIPServerPort, properties include: ExpireConnectionSec='20', MaxReceiveRecordBytes='1000000000', MaximumConnections='10000', Port='1460', SO_KEEPALIVE='false', SO_LINGER='false', SO_LINGER_TIMEOUT_SEC='-1', SO_RCVBUF='0', SO_SNDBUF='0', TCP_NODELAY='false', TrafficClass='-1'. A message at the bottom states "BIP8071I: Successful command completion."

Figure 7-28 Report TCPIPServer configurable service properties

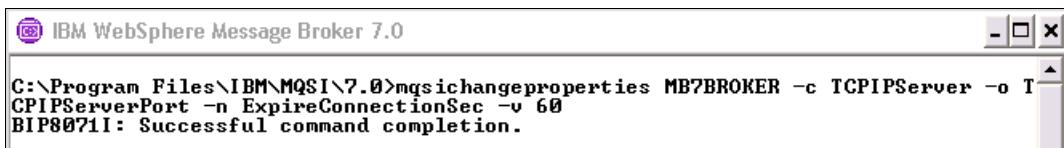
Change TCPIPServer configurable services

The command shown in Example 7-11 on page 275, changes the connection expire time on the connections to 60 seconds.

Example 7-10 Time on connections

```
mqsicahgeproperties MB7BROKER -c TCPIPServer -o TCPIPServerPort -n
ExpireConnectionSec -v 60
```

Figure 7-29 shows the results of the successful command completion.



```
C:\Program Files\IBM\MQSI\7.0>mqsiclipsechangeproperties MB7BROKER -c TCPIPServer -o TCP  
CPIPServerPort -n ExpireConnectionSec -v 60  
BIP8071I: Successful command completion.
```

Figure 7-29 Change TCPIPServer configurable service property

Execute the **mqsicreportproperties** command to see the changes.

Creating a TCPIPClient configurable service

The command in Example 7-11 creates a TCPIPClient configurable service named TCPIPClientPort, as highlighted in the Figure 7-30.

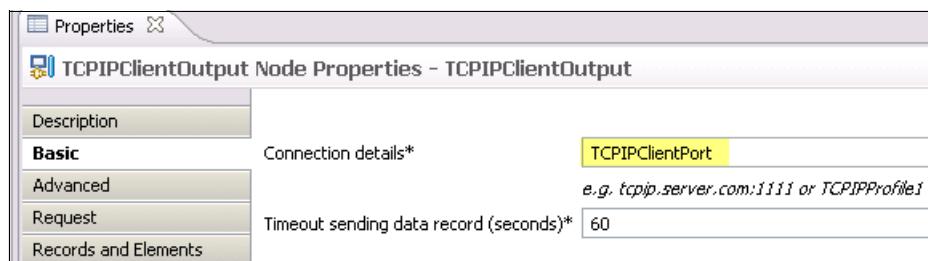


Figure 7-30 TCPIPClient configurable service

Figure 7-30 creates a TCPIPClient configurable service for port 1425 on localhost, which will always have 10 connections open and allow a maximum of 30 connections. Example 7-11 shows the command for the configurable service port.

Example 7-11 Command for configurable service port

```
mqsicreateconfigurableservice MB7BROKER -c TCPIPClient -o  
TCPIPClientPort -n  
Hostname,MinimumConnections,MaximumConnections,Port -v  
localhost,10,30,1425
```

Figure 7-31 on page 276 shows the result of the successful command completion.

```
C:\Program Files\IBM\MQSI\7.0>mqscicreateconfigurableservice MB7BROKER -c TCPIPC1ient -o TCPIPC1ientPort -n Hostname,MinimumConnections,MaximumConnections,Port -v localhost,10,30,1425
BIP8071I: Successful command completion.
```

Figure 7-31 Create TCPIPC1ient configurable service

Reporting all TCPIPC1ient configurable services

The command in Example 7-12 displays all of the properties that are associated with the TCPIPC1ient configurable service.

Example 7-12 Configurable Services

```
mqscreportproperties MB7BROKER -c TCPIPC1ient -o
AllReportableEntityNames -r
```

Figure 7-32 shows the result of the successful command completion.

```
C:\Program Files\IBM\MQSI\7.0>mqscireportproperties MB7BROKER -c TCPIPC1ient -o AllReportableEntityNames -r
TCPIPC1ient
Default
  ExpireConnectionSec='-1'
  Hostname='localhost'
  MaxReceiveRecordBytes='1000000000'
  MaximumConnections='100'
  MinimumConnections='0'
  Port='0'
  SO_KEEPALIVE='false'
  SO_LINGER='false'
  SO_LINGER_TIMEOUT_SEC='-1'
  SO_RCUBUF='0'
  SO_SNDBUF='0'
  TCP_NODELAY='false'
  TrafficClass='-1'
TCPIPC1ientPort
  ExpireConnectionSec='-1'
  Hostname='localhost'
  MaxReceiveRecordBytes='1000000000'
  MaximumConnections='30'
  MinimumConnections='10'
  Port='1425'
  SO_KEEPALIVE='false'
  SO_LINGER='false'
  SO_LINGER_TIMEOUT_SEC='-1'
  SO_RCUBUF='0'
  SO_SNDBUF='0'
  TCP_NODELAY='false'
  TrafficClass='-1'
BIP8071I: Successful command completion.
```

Figure 7-32 Report TCPIPC1ient configurable service properties

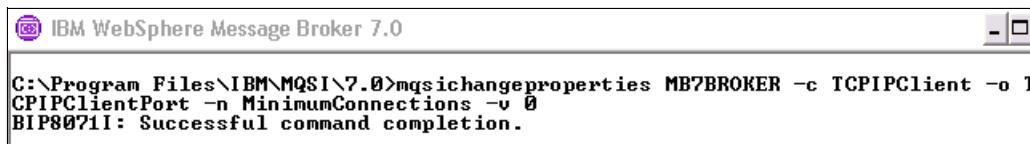
Changing TCPIPCClient configurable services

The command in Example 7-13 changes the configurable service to not make any client connections until the output or receive node receives them.

Example 7-13 Mqsichangeproperties command

```
mqsichangeproperties MB7BROKER -c TCPIPCClient -o TCPIPCClientPort -n MinimumConnections -v 0
```

Figure 7-33 shows the result of the successful command completion.



The screenshot shows a terminal window with the title "IBM WebSphere Message Broker 7.0". The command entered is "C:\Program Files\IBM\MQSI\7.0>mqsichangeproperties MB7BROKER -c TCPIPCClient -o TCPIPCClientPort -n MinimumConnections -v 0". The output message "BIP8071I: Successful command completion." is displayed below the command line.

Figure 7-33 Change TCPIPCClient configurable service property

Execute the **mqsireportproperties** command to see the changes.

For more information about TCP/IP configurable service and options, refer to the following topic in the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/com.ibm.etools.mft.doc/an37200_.htm

Using the WebSphere Message Broker Explorer to work with configurable services

The configurable services can be created, modified, and deleted using the new WebSphere Message Broker explorer.

To add a new TCPIPCClient or TCPIPServer configurable service using the WebSphere Message Broker Explorer:

1. In the Navigator view, expand the broker on which you want to add a new configurable service.
2. Right-click the **Configurable Services** folder, and click **New → Configurable service**. The Configurable services window is displayed.
3. Enter a name for your configurable service.
4. Select the type of configurable service to create, for example: TCPIPCClient or TCPIPServer.

5. For some configurable services, you can select an IBM-defined template to provide default values that you can use or update. If appropriate, select the IBM-defined template to use for your configurable service.
6. Enter values for the configurable service properties. The properties are populated with default values.
7. Click **OK** to create the new configurable service.

Figure 7-34 shows how to create a TCPIPServer configurable service using WebSphere Message Broker Explorer.

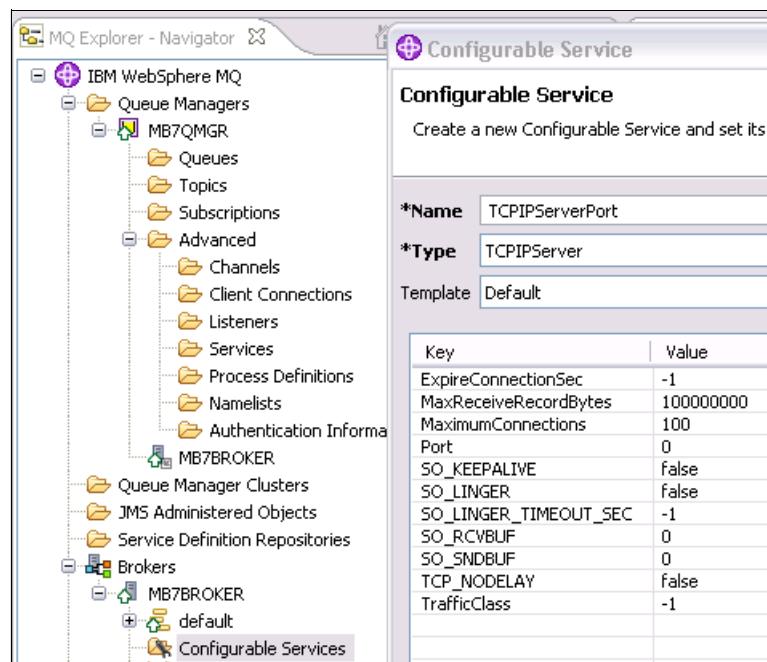


Figure 7-34 WebSphere Message Broker explorer

For more details about using WebSphere Message Broker Explorer to create a configurable service, refer to the following topic in the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/com.ibm.etools.mft.doc/be10320_.htm



Scenario: File processing

In this chapter, we demonstrate the file processing capability of WebSphere Message Broker with a new general purpose programmable node called PHPCompute node for message transformation and routing.

The step-by-step instructions in this chapter implement a file processing scenario that is based on a file processing pattern that WebSphere Message Broker V7 provides.

We address the following topics:

- ▶ “Introduction to file processing” on page 280
- ▶ “FtpServer configurable service for remote file processing” on page 291
- ▶ “Message transformation with PHPCompute node” on page 292
- ▶ “Message flows” on page 311

8.1 Introduction to file processing

In many business organizations, such as card companies and banks, file-based processing is still playing a significant role in their every day business. Data that is stored in files is read and processed by business applications or transferred to other destinations to be processed. In most cases, the file processing involves very different types of platforms and business applications, which might require message transformation and routing.

WebSphere Message Broker integrates applications that use different protocols and message formats and enables those applications to process the information in the files without changing the applications.

WebSphere Message Broker can support file processing in conjunction with the following methods:

- ▶ WebSphere MQ File Transfer Edition (MQ FTE)

MQ File Transfer Edition provides a managed file transfer service on top of WebSphere MQ. WebSphere Message Broker can provide MQ File Transfer Edition with additional capabilities, for example, the transformation or translation of data that is held in the files, content-based routing, code conversion, and so forth.

- ▶ WebSphere Transformation Extender (WTX)

WebSphere Transformation Extender is a powerful universal data transformation engine that extends the file processing capability of WebSphere Message Broker with the WebSphere TX map node.

- ▶ VSAM nodes for z/OS

The Virtual Storage Access Method (VSAM) nodes for z/OS (SupportPac IA13) extends the file processing capability of WebSphere Message Broker to VSAM data sets on z/OS. This SupportPac provides five nodes: VSAMInput, VSAMRead, VSAMWrite, VSAMUpdate, and VSAMDelete.

- ▶ QSAM nodes for z/OS

The Queue Sequential Access Method (QSAM) nodes for z/OS (SupportPac IA11) offers the ability to access QSAM files on z/OS for WebSphere Message Broker and can be used to integrate batch applications using the message flows.

- ▶ WebSphere Message Broker built-in file nodes

WebSphere Message Broker has two built-in nodes for file processing, which are the FileInput node and FileOutput node, as shown in Figure 8-1 on page 281. These nodes support local and remote file processing using either

File Transfer Protocol (FTP) or Secure File Transfer Protocol (SFTP), which enables file transfer by using the Secure Shell (SSH) protocol.

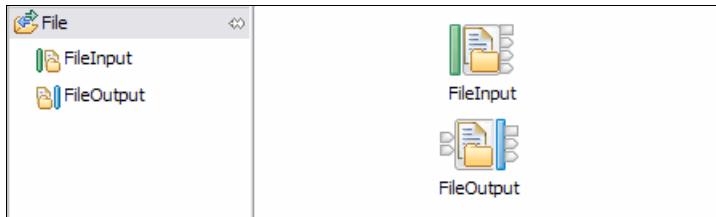


Figure 8-1 File nodes

The WebSphere Message Broker built-in File nodes offer the core capability for file support, differentiating them from any WebSphere Message Broker extensions, which require additional integration. In this chapter, we focus on the file handling capabilities of these built-in nodes, message transformation, and routing using the PHPCompute node.

Note: For more information about processing files in WebSphere Message Broker, refer to the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac55170_.htm

SupportPacs: IBM WebSphere MQ family SupportPacs are available at no charge in most cases, although others can be purchased as fee-based services. The SupportPacs provide a wide range of downloadable code and documentation that complements the WebSphere MQ family of products.

For more information about SupportPacs:

<http://www-01.ibm.com/support/docview.wss?rs=849&uid=swg27007205>

For the list of SupportPacs that are available for WebSphere Message Broker:

<http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg27007197#2>

8.1.1 File handling using the FileInput node

The FileInput node reads local or remote files that contain messages and then processes each message as a separate flow transaction.

Reading a file

The FileInput node scans a specified directory in the local file system of the broker machine, or a remote directory in a FTP server, searching for files to process. The FileInput node searches for files based on information, such as the file name and location, which is specified in the FileInput Node Properties view.

A file name or a file name pattern and directory information is provided on the Basic tab of the FileInput Node Properties view. The properties on this tab must have valid values, regardless of local or remote file processing. For the local file processing, the directory that is defined in the Input directory field is where the broker searches for files. For the remote file processing, the directory that is defined in this field stores files that are transferred from a FTP server for processing.

Figure 8-2 shows the Basic FileInput Node properties.

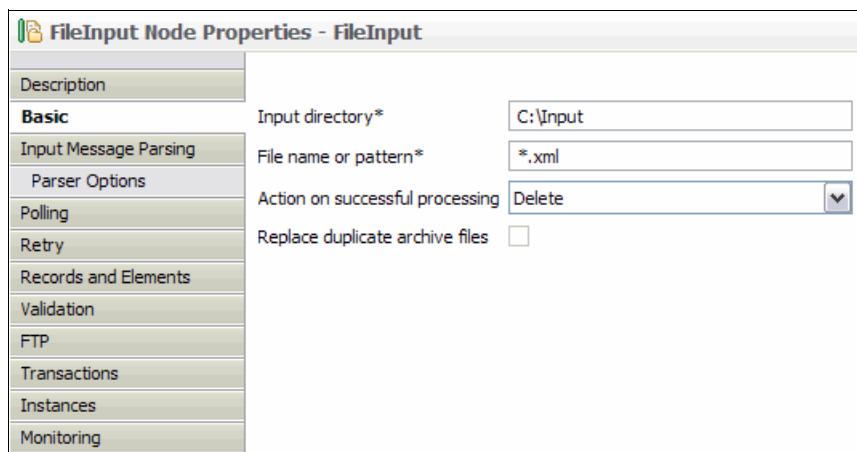


Figure 8-2 FileInput node Basic tab properties

In the Input directory field, you must enter the absolute or a relative directory path of the location of the files to be read. To use a relative directory path, you also must set the MQSI_FILENOES_ROOT_DIRECTORY environment variable to the base directory to be used to resolve the path. After it is set, a value of '.' can be specified in the Input directory field to write files using the path specified by the variable. If the environment variable is not set correctly, you will see an error during file processing, for example, on Windows systems, you might find the error in the Event Viewer, as seen in Example 8-1 on page 283. The Event Viewer can be opened from the WebSphere MQ Alert Monitor (Windows only) or using **Start → Control Panel → Administrative Tools → Event Viewer** from the Windows start menu. The message can be browsed in the Application category.

Example 8-1 Error message for incorrect relative file paths

(MB7BROKER.default) The File node "'FileInput'" in message flow
'MyApplicationFlow' cannot resolve the relative file path "'file'".

Relative file paths are resolved by using the absolute directory path in the 'MQSI_FILENOES_ROOT_DIRECTORY' environment variable.

This environment variable is not set correctly and cannot be used to resolve the path.

Either ensure that the 'MQSI_FILENOES_ROOT_DIRECTORY' environment variable contains an absolute directory path and that the directory exists and can be accessed by the broker, or change the node configuration to use absolute directory paths.

See the WebSphere Message Broker online documentation topic "FileInput node" or "FileOutput node" for more information.

In the File name or pattern field, either an exact file name or a file name pattern can be used. Acceptable wildcard characters on the file name pattern are '*' for multiple characters and '?' for a single character, for example *.xml matches all of the file names with an xml extension and f?????.csv matches all file names that start with the letter f followed by six characters and then the extension .csv.

The FileInput node initially scans the Input directory for the specified file, and if there is no file to process, it waits for a certain period of time, which is specified by the Polling interval value on the Polling tab. If any file that matches the input pattern or the input file name is found during the scan, the node reads the file and propagates any messages that are available in the file. Even if the input file is empty, an empty message is propagated because it is still a valid file.

While the file is processed, the FileInput node keeps the file in the Input directory and after the successful completion of the file processing, the file is deleted or archived in a subdirectory of the Input directory called mqiarchive, as defined in the Action on successful processing properties.

FileInput node: For more information about reading a file with the FileInput node, refer to the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac55420_.htm

FileInput node properties: For more information about the FileInput node properties, refer to the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac55150_.htm

Accessing a file on a remote FTP or SFTP server

To allow the FileInput node to access remote files, the Remote Transfer property on the FTP tab must be selected, as shown in Figure 8-3, and valid values must be provided for accessing a FTP server in addition to the properties in the Basic tab. If you select the Remote Transfer property, the Scan delay property on the FTP tab overrides the Polling interval property on the Polling tab.

The Transfer Protocol property specifies the protocol that the FileInput node uses for the remote file access, either FTP or SFTP. The FileInput node must, naturally, have the information about the IP address and port number of the FTP or SFTP server and the user and password details to log on with.

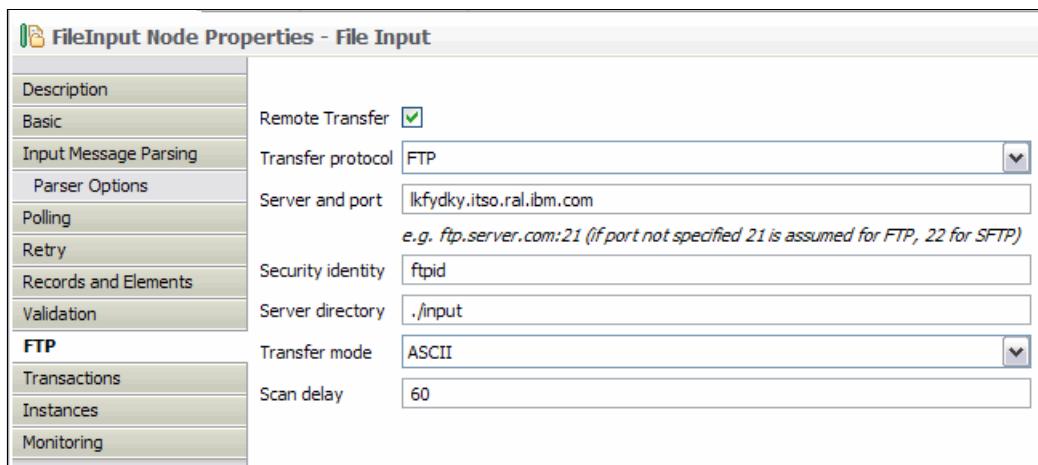


Figure 8-3 FileInput node FTP properties

For the IP address, the FileInput node supports IPv4, IPv6, and URL formats. To provide the log-on information that is necessary for the broker to access the FTP or SFTP server, you must define a security identity using the `mqsisetdbparms` command.

The `mqisetdbparms` command associates a specific user ID and password (or SSH identity file) with one or more resources that are accessed by the broker. The user ID and password pair are created in the DSN folder under the broker registry folder:

- ▶ For FTP:

```
mqisetdbparms MyBroker -n ftp::myidentity -u myuserid -p mypassword
```

- ▶ For SFTP:

```
mqisetdbparms MyBroker -n sftp::myidentity -u myuserid -p  
mypassword
```

The `mqisetdbparms` command: For more information about the `mqisetdbparms` command, refer to the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/an09155_.htm

FtpServer configurable service can be used to provide information about the Server and port, Security identity, Transfer mode, and Scan delay. It overrides the values that are provided for the properties on the FTP tab if it is defined. Refer to 8.1.3, “FtpServer configurable service for remote file processing” on page 291 for additional information.

The FileInput node first tries to log on to the FTP or SFTP server with the provided security credentials and scans a server directory for a file. If the file is found, the file is transferred to the local directory that is defined in the Input directory property on the Basic tab, and the file is deleted from the server directory and processed.

Reading files from remote FTP or SFTP directories: For more information about reading a file from a remote FTP or SFTP directory, refer to the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac55422_.htm

Increasing throughput with additional instances

WebSphere Message Broker can create up to 256 additional instance threads of a message flow to increase the throughput of the message flow. Assuming that the sequence of messages is not significantly important. You can define the number of additional instances in the Manage and Configure view of the Broker Archive Editor.

Both the WebSphere Message Broker Toolkit and the WebSphere Message Broker Explorer provide the Broker Archive Editor to create and manage broker archive (BAR) files, as shown in Figure 8-4.

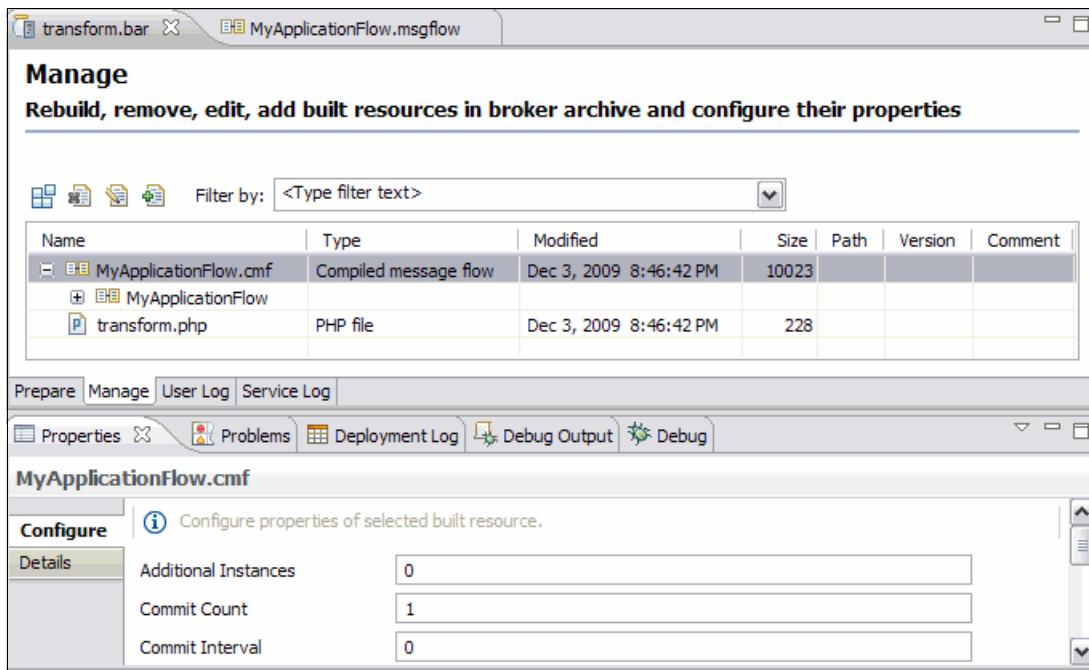


Figure 8-4 Additional Instances property of a message flow

The default value for additional instances of the message flow is 0, which means no additional threads and therefore only a single instance of the message flow exists in the execution group by default.

The broker creates additional threads when more than one message or file is available for input nodes. However, if multiple input nodes exist in the message flow, there is no guarantee that the available threads are evenly distributed among the input nodes. The additional threads therefore, might not actually improve the throughput of the message flow at all if the available threads are not properly allocated to the troubled input nodes.

The FileInputStream node has two properties to address this potential issue. These properties can be used to ensure that the required number of additional instances are allocated to the specific input nodes. See Figure 8-5 on page 287.

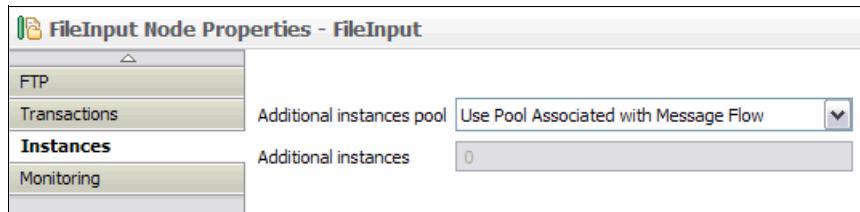


Figure 8-5 Additional instances properties of FileInput nodes

In the Additional Instances Pool property, the choice that is made between Use Pool Associated with Message Flow and Use Pool Associated with Node decides whether the additional threads are allocated from a thread pool for the whole message flow or from a thread pool for the specific node that it is associated with. In Additional Instances, the value is the number of additional instances in the selected pool in the Additional instances pool property. These properties are also available for other Input nodes, such as the SOAP node and the MQInput node.

Generally speaking, multiple threads in application programming can resolve the throughput problem, but the use of multiple threads might also imply a certain complication in handling a file if the file is shared by the multiple threads, for instance, there must be some consideration in avoiding the same record being processed multiple times or dealing with random order processing in case sequential processing is required.

The broker uses a lock mechanism to avoid these potential problems with the multiple threads. Only one instance is allowed to read a file and this instance serially processes each record in the file. The multiple instances might, therefore, not actually be so effective unless the message flow processes more than one file simultaneously.

In addition, message flows in the same execution group cooperate to avoid files being processed or accessed multiple times. There is no such cooperation between message flows in different execution groups or brokers. In that case, the integrity of the file processing solely relies on operating system file locks, which might not work well when the input directory for the FileInput nodes are shared across a network, for example, a shared input directory on the Network File System (NFS).

8.1.2 File handling using the FileOutputStream node

The FileOutputStream node writes messages to files in a specified output directory in the broker's local file system or a remote file system using FTP or SFTP.

Writing a file

The information about an output file name and the output directory of the **FileOutput** node is defined in the Basic tab of the **FileOutput Node Properties** view. This information is required for both local and remote file writing. While the **FileOutput** node is writing a file with the file name defined on the Basic tab, the file is kept in the transit subdirectory of the output directory called `mqsitransit`. After the completion of the file writing, the file is placed in the output directory or transferred to a FTP server.

If a file of the same name that the **FileOutput** node is trying to place already exists in the output directory of the local file system, the node replaces the existing file or archives the file in a subdirectory of the output directory called `mqsiarchive`, depending on the Output file action property.

In the case of writing a remote file, the **FileOutput** node overwrites the existing file in a remote FTP server directory, if the name of the existing file is the same as the name of the new file that the **FileOutput** node is transferring.

Figure 8-6 shows the Basic **FileOutput** node properties.

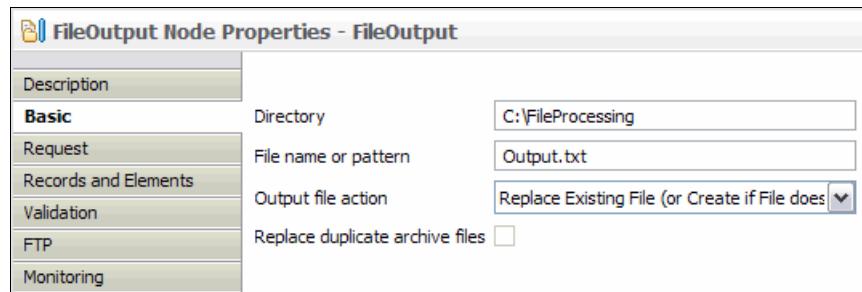


Figure 8-6 FileOutput Node Properties

In the Directory field, an absolute or relative path can be used. The **FileOutput** node also resolves the relative path by using the absolute directory path in the `MQSI_FILENOES_ROOT_DIRECTORY` environment variable, which is the same as the **FileInput** node, for instance, a value of '.' can be specified in this property to write files in the directory that is defined in the `MQSI_FILENOES_ROOT_DIRECTORY` environment variable itself.

In the File name or pattern field, a specific file name or a file name pattern must be valid on the broker's file system where the message flows are deployed. If the output file is to transferred to a remote FTP server, the filename must respect the conventions of the remote file system, for instance case-sensitivity.

Writing a file with the FileInput node: For more information about writing a file with the FileInput node, refer to the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac55460_.htm

FileOutput node properties: For more information about the FileOutput node properties, refer to the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac55160_.htm

Accessing a remote FTP or SFTP server for writing

Figure 8-7 on page 290 shows the FTP properties of the FileOutput node. In order for the FileOutput node to write a file in a remote directory, you must define the properties on the FTP tab in addition to the properties on the Basic tab.

If the Remote Transfer property is selected, the FileOutput node creates destination files on a remote FTP or SFTP server and transfers the finally processed files to the server.

The destination files are readable even before the file transfer is complete. If there are remote applications waiting to read these files, there is a possibility that incomplete or corrupted data can be processed. You must ensure that the files are not processed before the file transfer is complete.

Valid information about the transfer protocol and the server address is required for the FileOutput node to access a remote FTP or SFTP server. The security identity that is defined in the FileInput node can be used for the FileOutput node, to log on to the FTP server. The security identity can also be defined using the `mqisetdbparms` command, as explained in “Accessing a file on a remote FTP or SFTP server”.

An FtpServer configurable service can also be used for the FileOutput node to provide information about the FTP Server and port, Security identity, and Transfer mode. It overrides the properties on the FTP tab if it is defined. Refer to 8.1.3, “FtpServer configurable service for remote file processing” on page 291 for additional information.

If the Retain local file after transfer property is selected, the FileOutput node deletes local files that it created from the local file system after the files are successfully transferred to the remote FTP server.

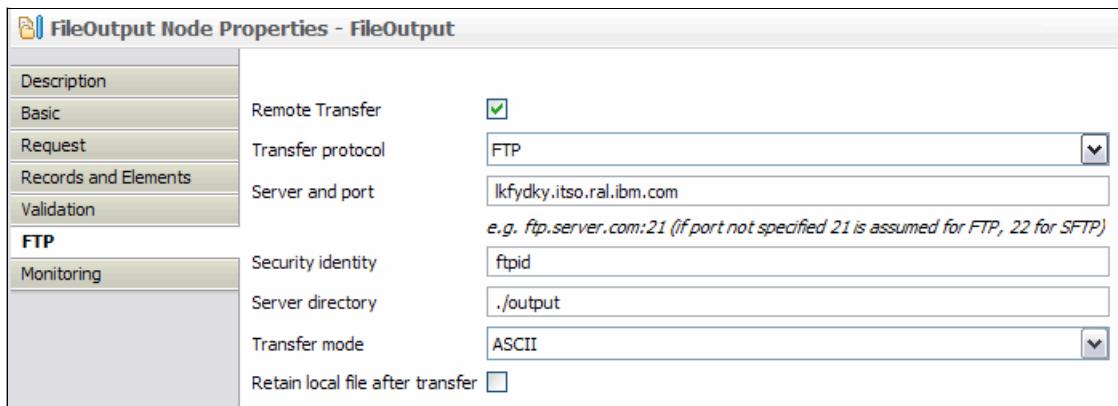


Figure 8-7 FileOutput node FTP properties

Note: For more information about writing a file to a remote FTP or SFTP server, refer to the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac55462_.htm

Writing a file with multiple instances

If a message flow is configured to use additional instances, all of the instances of the message flow try to write to the same file. Message flows that are in the same execution group or in other execution groups can be configured to write to the same file.

When records are being added to a file, each record is written by a single message flow instance. Although the FileOutput node ensures that a file is available to only a single instance at a time, multiple message flow instances can be configured to append records to the same file.

Because the multiple instances can run in any order, the order of the records in the file can be disrupted. Therefore, only one FileInput node instance should write the file in case the sequence of the records in the output file must be kept.

While the file is being written by the FileOutput node, the file is locked by the execution group to prevent it from being corrupted. Other programs, such as File nodes in other execution groups cannot read, write, or delete the file. However, the lock can be released if there are longer intervals between record writings, and it is possible that another execution group can obtain a lock on the file.

8.1.3 FtpServer configurable service for remote file processing

The FtpServer configurable service specifies the FTP and SFTP settings for a message flow. If the FtpServer configurable service is defined, it overrides the FTP or SFTP settings on the FileInput and FileOutput nodes.

As one of configurable services in WebSphere Message Broker, FtpServer configurable service is a set of runtime properties for an external service, in this case FTP or SFTP, that is used within message flows but is independent to the nodes that use the services. The nodes and the message flows refer to the configurable services to find properties at run time.

You can change the properties of the configurable services without redeploying the message flows, but you might need to restart the execution groups and the relevant message flows to apply the changes, if not explicitly stated otherwise.

FtpServer configurable service properties: For more information about FtpServer configurable service properties, refer to the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/an60170_.htm#an60170__ftp

The **mqsicreateconfigurableservice** command creates a configurable service, and the **mqsideleteconfigurableservice** command deletes the configurable service, for example, the following command creates an FtpServer configurable service called *FtpService* for the FTP server at *lkfydky.itso.ral.ibm.com*, with 20 seconds on the scan delay.

```
mqsicreateconfigurableservice MB7BROKER -c FtpServer -o FtpService -n  
serverName,scanDelay -v lkfydky.itso.ral.ibm.com,20
```

To delete the FtpServer configurable service called *FtpService*:

```
mqsideleteconfigurableservice MB7BROKER -c FtpServer -o FtpService
```

To change the properties of the configurable service, use the **mqsicchangeproperties** command, and to display the properties of the configurable service, use the **mqsicreportproperties** command, for example, to change the scan delay property of the FtpServer configurable service called *FtpService* from 20 seconds to 120 seconds:

```
mqsicchangeproperties MB7BROKER -c FtpServer -o FtpService -n scanDelay  
-v 120
```

To display all properties of the FtpServer configurable service called *FtpService*:

```
mqsi reportproperties MB7BROKER -c FtpServer -o FtpService -r
```

WebSphere Message Broker runtime commands: For more information about WebSphere Message Broker runtime commands including **mqsicreateconfigurableservice** and **mqsi reportproperties**, refer to the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/an07060_.htm

The WebSphere Message Broker Explorer can also be used to view, create, and modify the configurable services. You can review the properties of the configurable service that was created from the WebSphere Message Broker Explorer by right-clicking the name of the configurable service and clicking **Properties**, as shown in Figure 8-8.

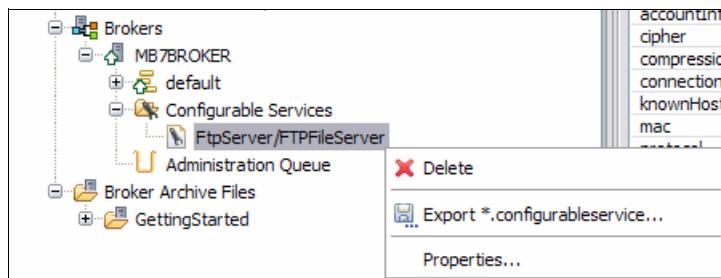


Figure 8-8 Configurable Services in the WebSphere Message Broker Explorer

WebSphere Message Broker Explorer: For more information about using the WebSphere Message Broker Explorer to work with configurable services, refer to the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/be10320_.htm

8.1.4 Message transformation with PHPCompute node

WebSphere Message Broker provides a number of built-in nodes for message transformation, as shown in Figure 8-9 on page 293.

- ▶ Mapping
- ▶ XSL Transform
- ▶ Compute

- ▶ JavaCompute
- ▶ PHPCompute

WebSphere Message Broker V6.1.0.3 introduces a new general purpose programmable node for message transformation and routing using PHP scripting language. This new node is called PHP compute node. In this section, we address the general description of this new node, and cover development and debugging.



Figure 8-9 PHPCompute node

PHP: Hypertext Preprocessor (PHP) is a general purpose scripting language that is widely used for Web development. The PHPCompute node uses the PHP scripting language to route and transform incoming messages and embeds the IBM sMash runtime for PHP, fully compliant with PHP version 5.2, to run PHP scripts and to provide a PHP API to work with messages and to interact with the broker. You can navigate WebSphere Message Broker message trees using syntax that is integrated into PHP language, and also you can dynamically add output terminals for message routing.

Logging

The PHP engine writes standard output and standard error messages to the console log for the broker, for example on the Windows system both standard logs are written to a single file:

- ▶ %workpath%/components/broker_name/execution_group_uuid/console.txt
- ▶ %workpath% is the working directory that is defined for the broker. If the workpath environment variable is not defined, the default location for console.txt on Windows XP is:

```
%ALLUSERSPROFILE%\Application  
Data\IBM\MQSI\component\broker_name\execution_group_uuid\
```

PHP scripts

The PHP script must be written within the <?php and ?> tags, as shown in Example 8-2 on page 294, and anything outside of the tags is ignored.

Example 8-2 PHP script

```
<?php  
// Body of the script  
?>
```

You can create a PHP script with or without a class and evaluate method. If you create a PHP script that includes a class and evaluate method, the PHP script must contain a class with the same name as the PHP file, as shown in Example 8-3, and this class must contain a function called evaluate.

Example 8-3 PHP script, named Hello with a class and evaluate method

```
<?php  
class Hello {  
    /**  
     * @MessageBrokerSimpleTransform  
     */  
    function evaluate ($output, $input) {  
        $output->XMLNSC->doc->greeting = 'Hello';  
    }  
}  
?>
```

Example 8-4 shows the output message from the PHP script in Example 8-3.

Example 8-4 The output message from Example 8-3

```
<doc>  
    <greeting>Hello</greeting>  
</doc>
```

Example 8-4 can be written without a class and evaluate method, as shown in Example 8-5.

Example 8-5 PHP script without a class and evaluate method

```
<?php  
    $output_message = new MbsMessage($assembly[MB_MESSAGE]);  
    $output_message->XMLNSC->doc->greeting = 'Hello';  
    $output_assembly = new  
MbsMessageAssembly($assembly,$output_message);  
    $output_assembly->propagate('out');  
?>
```

By default, the Invoke evaluate method property of the PHPCompute node is selected, as shown in Figure 8-10. If the property is selected, you must include a class and evaluate method in the PHP script. If you use a PHP script without a class and evaluate method, the Invoke evaluate method property of the PHPCompute node must be cleared.

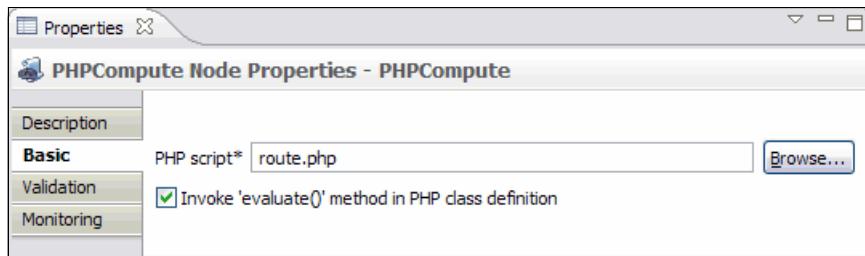


Figure 8-10 PHPCompute node basic properties

Writing PHP code: For more information about writing PHP code, refer to the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac69012_.htm

WebSphere Message Broker PHP API

The WebSphere Message Broker PHP API provides four classes to represent the message, the message assembly, and the element tree.

The four classes that are defined are:

- ▶ MbsElement

The MbsElement class represents a single parsed element in a message or other logical tree.

- ▶ MbsMessage

The MbsMessage class represents one of the logical trees that make up the message assembly.

- ▶ MbsMessageAssembly

The MbsMessageAssembly class represents the message assembly that is propagated between nodes in a message flow.

The message assembly comprises four individual trees:

- Message
- LocalEnvironment
- GlobalEnvironment

- ExceptionList
- ▶ MbsBlob
 - The MbsBlob class supports the ESQL BLOB type.

The PHP API: For more information about the PHP API, refer to the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac69021_.htm

Annotations

Annotations specify message transformation options and reduce the amount of code by removing repeatable code you, otherwise, have to write.

Annotations alter the behavior of the evaluate method when using the PHP class structure. Multiple annotations for an evaluate method can be combined, and also no annotations are possible. If annotations are not specified at all, the first argument to the evaluate method is a read-only assembly. If an annotation is specified, the output assembly is passed to the evaluate method as the first parameter and the input assembly is passed as the second parameter. Annotation names are case-sensitive, and unrecognizable annotations are ignored.

Example 8-6 shows the example of combined annotations for the evaluate method. You must create a JavaDoc style documentation block and specify a required annotation or annotations.

Example 8-6 Combined annotations

```
<?php
class Hello {
    /**
     * @MessageBrokerSimpleTransform
     * @MessageBrokerRouter
     */
    function evaluate ($output, $input) {
        // do something
    }
}
?>
```

The annotations that are supported by the broker are:

- ▶ `@MessageBrokerSimpleTransform`

The `@MessageBrokerSimpleTransform` annotation causes two parameters to be passed to the evaluate method. The first parameter is a reference to the output assembly, and the second parameter is a reference to the input assembly. The second parameter is optional.

- ▶ `@MessageBrokerCopyTransform`

The `@MessageBrokerCopyTransform` annotation causes one parameter to be passed to the evaluate method. This parameter is a reference to the output assembly with the contents of the input assembly already copied into it. The input assembly is available with the `@MessageBrokerCopyTransform`. If you declare a second parameter (which is optional), the input assembly is passed to it.

If the `@MessageBrokerCopyTransform` and `@MessageBrokerSimpleTransform` annotations are specified together, the `@MessageBrokerCopyTransform` annotation takes precedence.

- ▶ `@MessageBrokerRouter`

The `@MessageBrokerRouter` annotation causes the return value of the evaluate method to be used to specify the terminal through which the message is to be propagated. The terminal can be either the Out terminal (defined on the node) or a dynamic terminal that you created. You can add output terminals dynamically to your node instance in the Message Flow editor. The string that is returned from the evaluate method must match either the name of the dynamic terminal that you defined or the Out terminal. If no return value is specified, the output assembly is not propagated to the next node after the evaluate method returns.

- ▶ `@MessageBrokerLocalEnvironmentTransform`

The `@MessageBrokerLocalEnvironmentTransform` is similar to the `@MessageBrokerSimpleTransform` annotation but creates a copy of the local environment tree in the output assembly.

If the `@MessageBrokerLocalEnvironmentTransform` is used, nodes downstream of the PHPCompute node see changes to the local environment. If the `@MessageBrokerLocalEnvironmentTransform` is not used, the node can still modify the local environment, and all nodes in the flow (including upstream nodes) can see the changes.

Annotations: For more information about annotations, refer to the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac69007_.htm

PHP: For more information about PHP:

<http://php.net/>

For more information about using the PHPCompute, refer to the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac69250_.htm

PHP Eclipse plug-in

PHP Development Tools (PDT) is an open source development tool that is based on the Eclipse project and Web Tools Platform (WTP). For the development and debugging of PHP scripts for the PHPCompute node, you can use a standalone version of Eclipse PDT, *All in Ones/Eclipse PHP Package*, or a PDT plug-in that is installed on top of the WebSphere Message Broker Toolkit. Both of them are available from the eclipse.org Web site:

<http://www.eclipse.org/pdt/downloads/>

It is conceivable that a future release of the WebSphere Message Broker Toolkit will integrate the PDT as base functionality. Unfortunately, the current release of Message Broker does not have the capability to support PHP development, such as the content/code assist or the code templates, nor to debug PHP scripts that are executed by the PHPCompute node. For the time being, the PDT plug-in from eclipse.org can be installed on top of the WebSphere Message Broker Toolkit to support the PHP development and debugging.

In this scenario, we use the PHP plug-in V2.0.1 on top of the WebSphere Message Broker Toolkit V7.0.0, which is based on the Eclipse 3.4 platform.

Prerequisites

There are version level dependencies that you must consider when installing the PDT. Versions later than V2.0.1 of the PDT plug-in do not work with the WebSphere Message Broker Toolkit 7.0.0 because they are built on the Eclipse 3.5 platform.

The PDT 2.0.1 plug-in requires a specific version of the Dynamic Language Toolkit (DLTK) as a prerequisite, and the plug-in does not work with any other versions of DLTK. The versions of PDT and DLTK that we use in this scenario are:

- ▶ Dynamic Language Toolkit (DLTK) Core R1.0 1.0M4 200812290249
- ▶ PHP Development Tools (PDT) runtime 2.0.1 (2009/04/28)

You can also download the required versions of DLTK for the PDT from the same Web site for the PDT download:

<http://www.eclipse.org/pdt/downloads/>

Installing the PHP plug-in

To install the PHP plug-in:

1. From the WebSphere Message Broker Toolkit workbench menus, click **Help → Software Updates**, and click the **Available Software** tab from the Software Updates and Add-ons window to display the items that are available for installation.
2. Add your local directory location of PDT and DLTK as update sites. Click **Add Site**, and click **Local**.
3. Specify the directory of the downloaded files, and then click **OK** to add the update sites in the list.
4. As shown in Figure 8-11 on page 300, select the software items from the added update sites, and click **Install** to initiate the installation. After requirements and dependencies are calculated and the license agreements for the items are accepted, the installation begins.

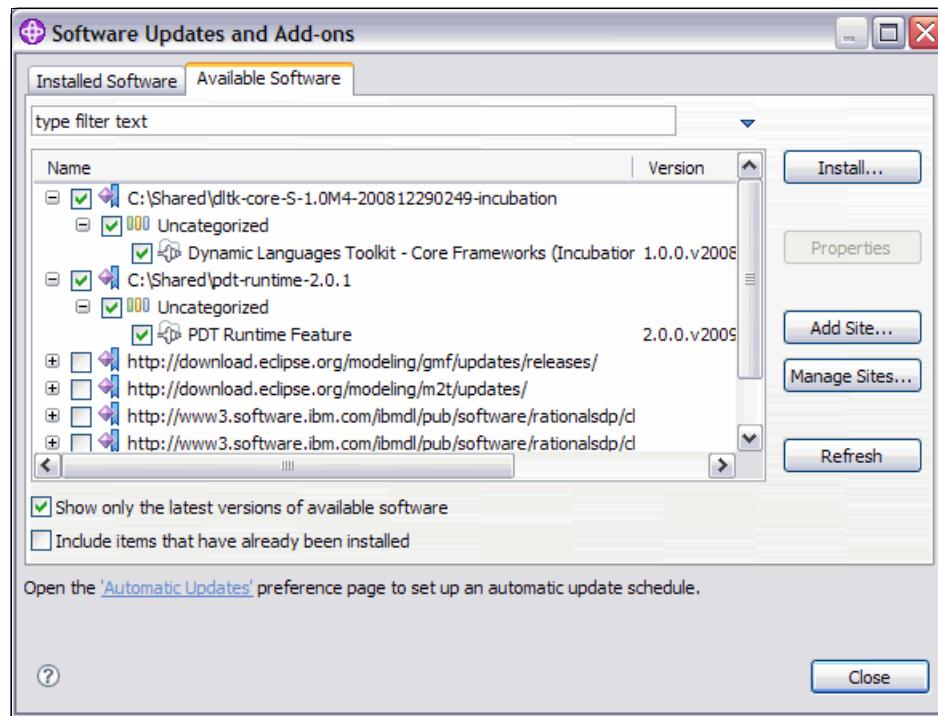


Figure 8-11 Software Updates and Add-ons

The completion of the installation adds additional perspectives in the toolkit, which are PHP and PHP debug, as shown in Figure 8-12.

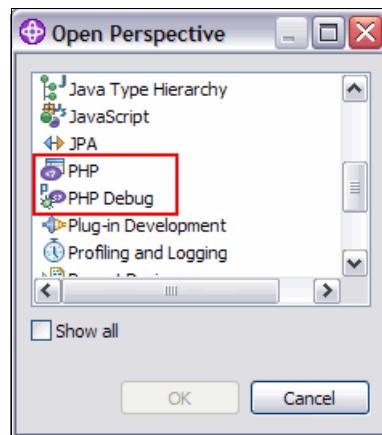


Figure 8-12 Open Perspective pop-up window

Preparing the workspace for PHP debugging activities

The PDT debug launch requires at least one PHP file in a PHP project in the current workspace. However, the PHP file and the project in the workspace do not have to be related to the actual PHP script that is being debugged because we use remote debugging through the XDebug protocol.

Creating a PHP project

To create a PHP project:

1. From the workbench menus, select **File** → **New** → **Project**.
 2. In the New Project wizard, select **PHP** → **PHP Project** → **Next**.
 3. Provide a name, for instance Dummy, for the Project name, and click **Finish**.
- Note:** You might be asked to open the PHP perspective if you have not already done so, depending on your preference setting.
4. Right-click the project created in the PHP Explorer view, and select **New** → **File** to create a new file called dummy.php, as shown in Figure 8-13.

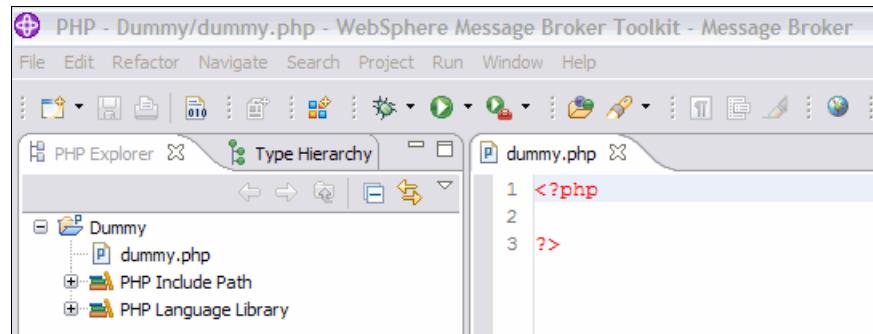


Figure 8-13 PHP Explorer

Debug configurations

To configure a launch setting for the PHPCompute node debug engine, a certain configuration of the PHP Web Page must be created to ensure that the PDT can communicate with the WebSphere Message Broker.

To create a new configuration:

1. To open the dialog for a launch setting configuration, go to the workbench menus, and select **Run** → **Debug Configurations**.
2. In the pane on the left of the dialog, right-click **PHP Web Page**, and select **New**.

3. Provide a name, for instance BrokerDebug, for the new launch configuration.
4. In the **Server tab**, shown in Figure 8-14:
 - a. Select **XDebug** for the Server Debugger.
 - b. Leave **Default PHP Web Server** for the PHP Server.
 - c. In the File field, click **Browse**, and select the dummy.php file that you previously created in the Dummy project.
 - d. Leave Break at First Line in the Breakpoint section and Auto Generate in the URL section selected.
 - e. Click **Apply**, and then click **Close**.

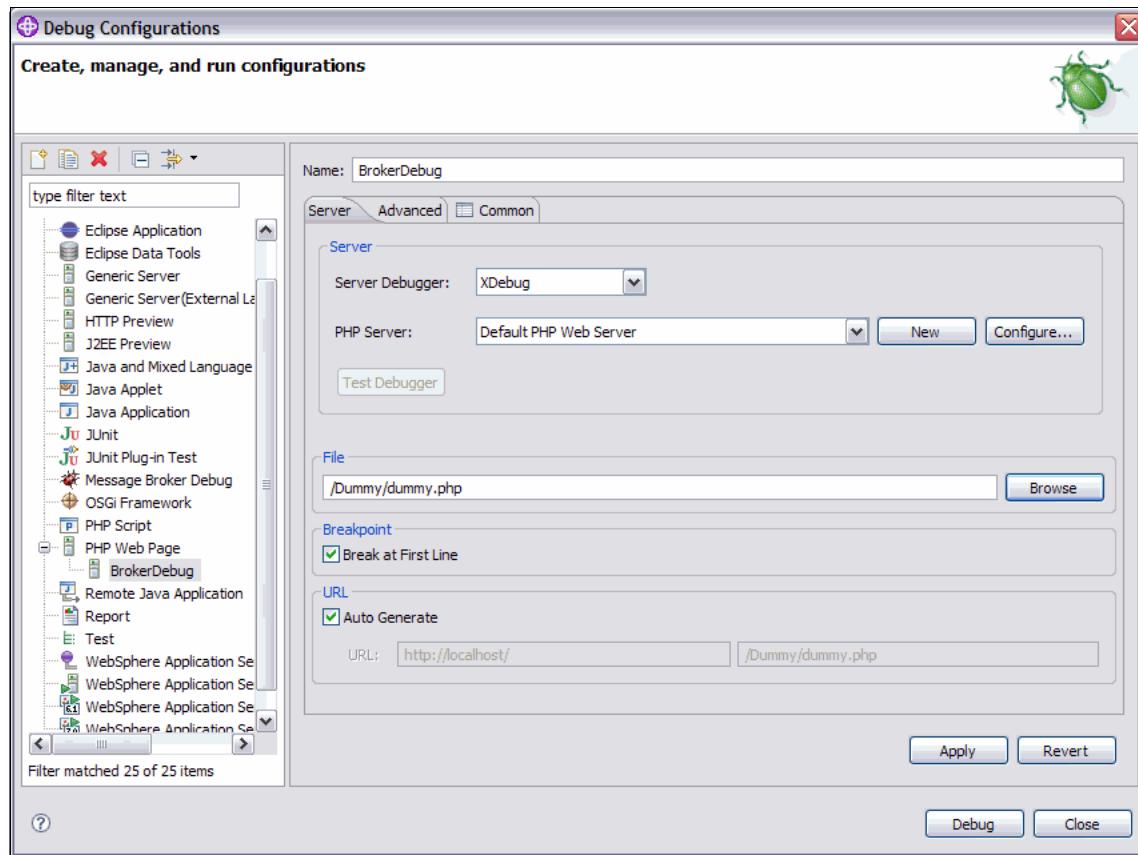


Figure 8-14 Debug configurations

PHP Web page: The PDT provides two types of the debug launch settings for the PHPCompute node, PHP Script and PHP Web Page. A PHP Script configuration debugs PHP scripts on your workspace locally using an internal PHP engine. A PHP Web page configuration debugs applications that are deployed on a server, in our case, the WebSphere Message Broker.

In our case, the PHP application process must belong to the WebSphere Message Broker; therefore, a PHP Web page launch setting is the only acceptable choice for debugging PHP scripts in the PHPCompute node. This type of debug launch first makes a request to a Web server that does not exist, Default PHP Web Server in this scenario, and then waits for a debug session from the PHPCompute node debug engine.

XDebug: XDebug is an open source tool for PHP development and a PHP extension for debugging. The embedded WebSphere sMash Runtime for PHP in the PHPCompute node has a debug connector that uses the DBGp protocol. This protocol is also used by XDebug.

For more information:

<http://www.projectzero.org>

<http://xdebug.org>

Starting the debugging session

Before starting the debugging session, you must set a system environment variable, MQSI_PHP_DEBUG_PORT, to enable the debug mode in the PHPCompute node. This variable specifies the port that the PDT is listening on. It is the same port that the debug engine in the PHPCompute node communicates on.

In the PDT, the XDebug listens on 9000 by default. If you want to change the port number for the XDebug:

1. Select **Windows → Preferences** from the WebSphere Message Broker Toolkit workbench menus to open the Preferences window.
2. Select **PHP → Debug → Installed Debuggers**, as shown in Figure 8-15 on page 304.
3. Select **XDebug**, click **Configure**, and change the value of the **Debug Port** in the **General Settings**.
4. Click **OK** to save the change.

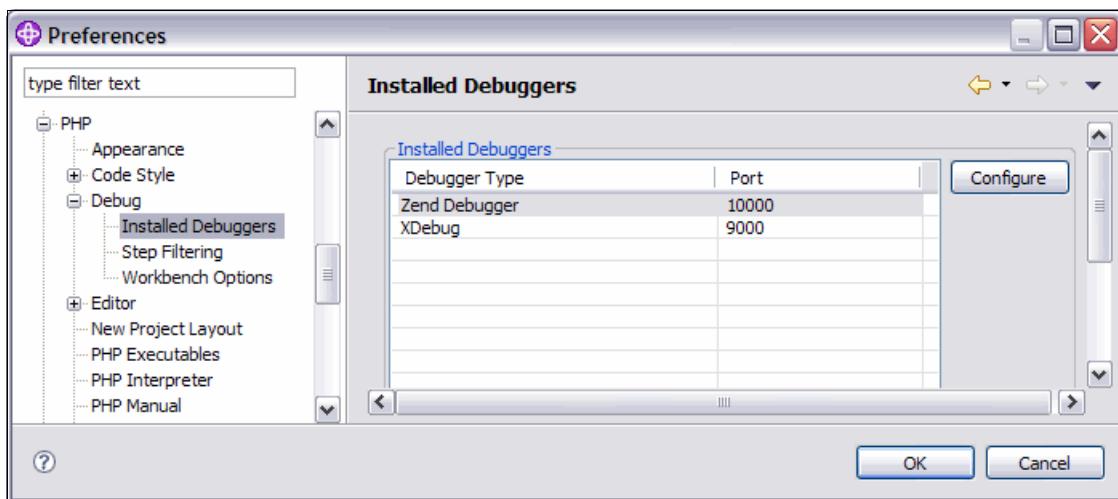


Figure 8-15 Installed Debuggers

To set the environment variable in Windows:

1. From the Windows start menu, select **Start** → **Control Panel** → **System**.
2. In the **Advanced** tab, click **Environment Variables**.
3. In the system variable section, click **New** to add a new system variable called **MQSI_PHP_DEBUG_PORT** with the default value **9000**, as shown in Figure 8-16 on page 305.
4. Click **OK** to close the dialog.
5. Restart the broker. You might also need to restart Windows for the changes you made in the system environment to take effect before starting the broker.
6. Restart the execution groups and the message flows for debugging, if they are not already started.

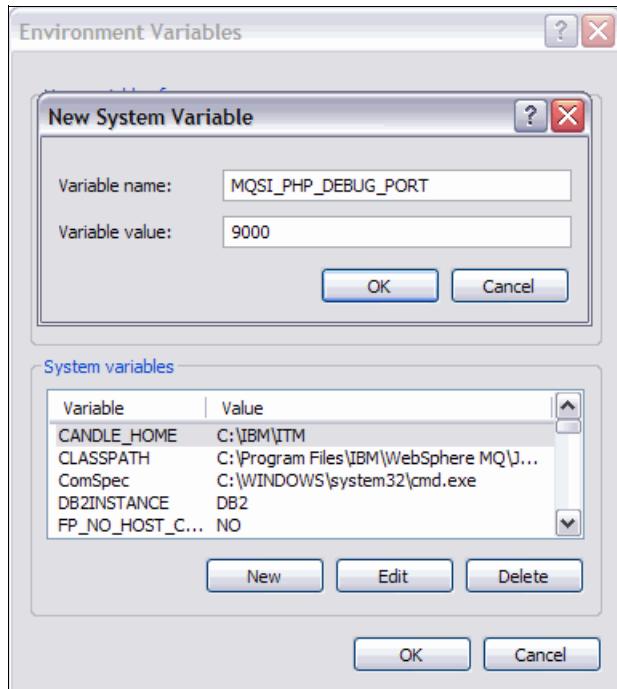


Figure 8-16 System Variables

To start the debugging session:

1. To open the dialog window, go to the workbench menus, and select **Run** → **Debug Configurations**.
2. Under PHP Web page, select **BrokerDebug**.
3. Click **Debug**.

When the debugging session starts, the PDT tries to open the *dummy.php* in the system default browser but fails with the error message Cannot find server or DNS Error, as shown in Figure 8-17 on page 306. It is because the PHP Web page debug launch first sends a request for *dummy.php* to a Web server that does not exist, *Default PHP Web Server* in this scenario, and then waits for a debug session from the PHPCompute node debug engine. Therefore you can ignore this error.

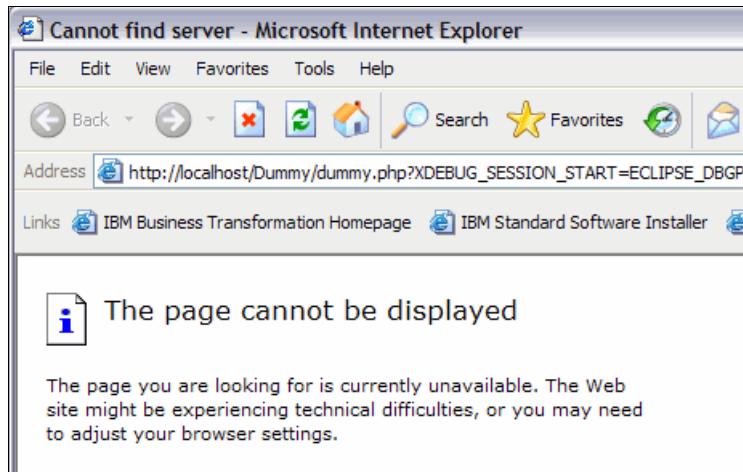


Figure 8-17 Server not found error in Internet Explorer

Now the WebSphere Message Broker Toolkit is ready for debugging. Afterward, each time PHP scripts are executed in the PHPCompute node, the PHP debugger stops the message flow processing at the first line as configured, as shown in Figure 8-18.

Note: You might be asked to open the PHP Debug perspective if you have not already done so, depending on your preference setting.

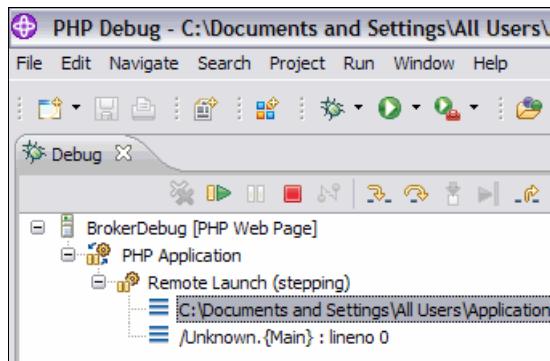


Figure 8-18 PHP Debug perspective

The debugging session remains active until the PHP Web page launch is terminated by the user. Use the right-click menu in the Debug view or the **Terminate** icon in the Debug view tool bar to issue the command to end the

debugging session. To disable the debugging mode in the PHPCompute node, remove the `MQSI_PHP_DEBUG_PORT` environment variable from the system variables.

Note: If the debugging mode in the PHPCompute node is left enabled even after the debugging session ends, it might have a performance impact on the execution of PHP scripts.

8.2 Scenario environment

We developed this scenario in an environment with the following software components:

- ▶ WebSphere Message Broker runtime 7.0.0.0
- ▶ WebSphere Message Broker Toolkit 7.0.0
- ▶ WebSphere MQ 7.0.1.0
- ▶ Dynamic Language Toolkit (DLTK) Core R1.0 1.0M4 200812290249
- ▶ PHP Development Tools (PDT) runtime 2.0.1 (2009/04/28)
- ▶ Microsoft® Windows XP SP2

We assume that the broker that the broker was created using the default configuration that is available in the WebSphere Message Broker Toolkit with the following names:

- ▶ Broker Name: MB7BROKER
- ▶ Queue Manager Name: MB7QMGR

It is also assumed that there is an FTP server that is available on the following address, listening on the specified port in the scenario:

`1kfydky.itso.ral.ibm.com:21`

8.3 Scenario

In this scenario, we use a pattern to implement message flows to process a series of records that are read from remote files.

8.3.1 File processing patterns

A pattern is a general solution to a particular problem and can be reused to increase development efficiency. It provides an architectural design to develop a quality solution and reusable assets and common implementation of aspects,

such as error handling and logging. A File Processing pattern is one of the patterns that is available from the WebSphere Message Broker Toolkit.

The File Processing patterns in WebSphere Message Broker provide support for activities, such as:

- ▶ Transforming and translating data that is held in files
- ▶ Splitting files into multiple individual transaction records
- ▶ Routing records
- ▶ Accumulating records into target files
- ▶ Routing entire files to specified locations and processors
- ▶ Routing records within files to specified locations and processors

At the time of this writing, only one example of the File Processing patterns is available from the toolkit, which is the Record Distribution to WebSphere MQ: one-way pattern. A Record Distribution pattern is an abstract pattern that can be used when records in local or remote files are processed and each record is sent to one or more transactional systems. This pattern bridges two styles of integration: file based and transaction based.

For this file processing scenario, we use this *Record Distribution to WebSphere MQ: one-way pattern* to generate a pattern instance, which provides the base template and required resources for the application development.

Patterns: For more information about patterns, refer to the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac67850_.htm

8.3.2 Scenario overview

Different applications require different input message formats. A fixed length format is common for mature applications, while many Java-based applications developed in recent years use XML as the input message format. WebSphere Message Broker can transform the message format of the messages to one supported by such applications without actually modifying the applications themselves.

In this scenario, we implement a message flow that reads records from a file and routes each record to a WebSphere MQ destination according to routing rules or to a default destination if a match is not found in the routing rules. The messages are transformed into a different format for each destination before being delivered.

Figure 8-19 provides a high-level overview of this scenario.

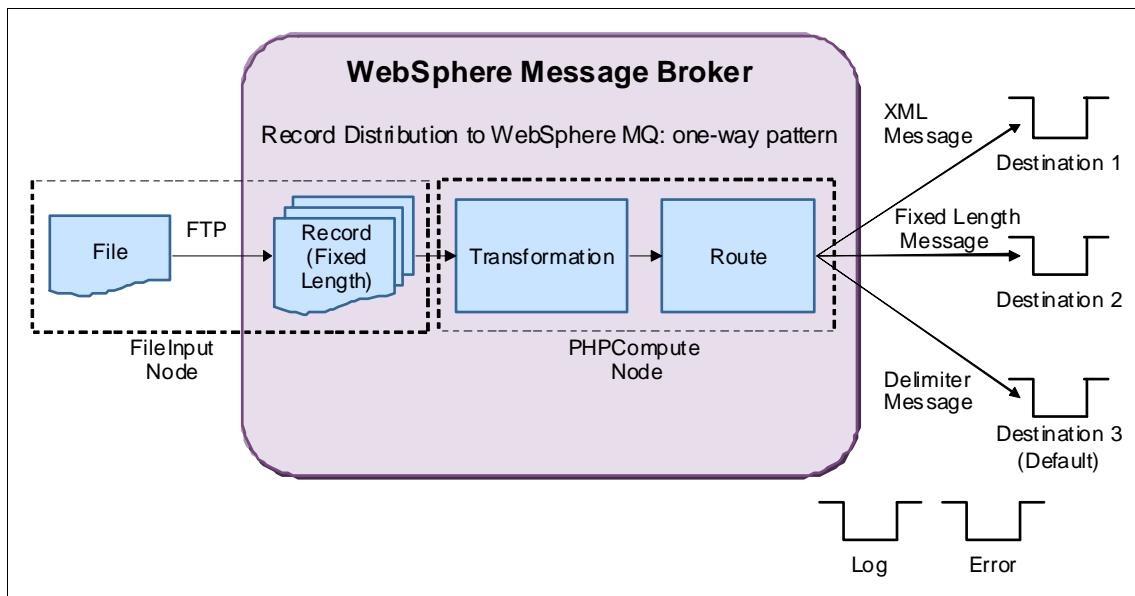


Figure 8-19 Scenario overview

The highlights of this scenario are:

1. WebSphere Message Broker uses the FileInput node to read a file from a remote FTP server. The FileInput node scans the remote directory for the input text file that contains data in a fixed length format. If the file is found, the file is transferred to the broker's local directory and processed.
2. Each record is propagated as a separate transactional message flow. The records are parsed by the MRM Custom Wire Format (CWF) parser using the message definition.
3. PHP scripts in the PHPCompute node transform the incoming messages that belong to the MRM domain into XML messages in the XMLNSC domain, or Delimited messages in the BLOB domain, and route them to the WebSphere MQ destinations based on the routing rule. The destination is dynamically determined from the fields in each record.

8.3.3 Record structure

In this scenario, a single line in the input text file is a single record. Table 8-1 on page 310 shows the structure of the record. Example 8-7 on page 310 shows the sample records.

Table 8-1 Record description

Seq.	Name	Length (Bytes)	Description
1	EmpNo	6	Employee number
2	FirstName	12	First name of employee
3	MidInit	1	Middle initial of employee
4	LastName	15	Last name of employee
5	WorkDept	3	ID of department in which the employee works
6	PhoneNo	4	Employee telephone number
7	HireData	8	Date of hire
8	Job	8	Job held by the employee
9	BirthDate	8	Date of birth
10	Salary	11	Yearly salary in dollars
11	Bonus	11	Yearly bonus in dollars
12	Comm	11	Yearly commission in dollars

Example 8-7 Sample records

```
000010CHRISTINE IHAAS          A00397819950101PRES
19630824+0152750.00+0001000.00+0004220.00
000020MICHAEL LTHOMPSON        B01347620031010MANAGER
19780202+0094250.00+0000800.00+0003300.00
000030SALLY AKWAN             C01473820050405MANAGER
19710511+0098250.00+0000800.00+0003060.00
000050JOHN BGEYER            E01678919790817MANAGER
19550915+0080175.00+0000800.00+0003214.00
000060IRVING FSTERN           D11642320030914MANAGER
19750707+0072250.00+0000500.00+0002580.00
000070EVA DPULASKI          D21783120050930MANAGER
20030526+0096170.00+0000700.00+0002893.00
...
```

8.4 Message flows

In this section, we describe how to develop message flow applications by using a pattern. In this scenario, we use message flows that are generated from the pattern and then adapt the message flow to include the PHPCompute node. PHP scripts are used to route incoming messages to their destinations in the scenario.

8.4.1 Preparing the environment

In this section, we provide the environmental configurations that are required prior to the pattern instance generation to run the scenario.

WebSphere MQ configuration

You must create the following queues for the message destinations in this scenario:

- ▶ `DEFINE QLOCAL('DEST1.LQ')`
- ▶ `DEFINE QLOCAL('DEST2.LQ')`
- ▶ `DEFINE QLOCAL('DEST3.LQ')`

The following queues also must be created for error messages and log messages:

- ▶ `DEFINE QLOCAL('ERROL.LQ')`
- ▶ `DEFINE QLOCAL('LOG.LQ')`

Environment variable

For this scenario, we use the environment variable `MQSI_FILENOES_ROOT_DIRECTORY` to use relative paths for the file nodes.

To set the environment variable in Windows:

1. From the Windows start menu, select **Start** → **Control Panel** → **System**.
2. In the **Advanced** tab, click **Environment Variables**.
3. In the system variable section, click **New** to add a new system variable called `MQSI_FILENOES_ROOT_DIRECTORY` with the value `C:\FileProcessing`.
4. Click **OK** to close the dialog.
5. Restart the broker. You might also need to restart your Windows to apply the change in the system environment before starting the broker.

Defining a security identity

To define a security identity that requires for the broker to access the FTP server in this scenario:

1. From the Windows start menu, select **Start → All Programs → IBM WebSphere Message Broker 7.0 → Command Console.**
2. Run the following command in the console to create a security identity called `ftpid`, as shown in Figure 8-20:

```
mqsisetdbparms MB7BROKER -n ftp::ftpid -u ftpuser -p ftppwd
```

```
MQSI 7.0.0.0
C:\Program Files\IBM\MQSI\7.0

C:\Program Files\IBM\MQSI\7.0>mqsisetdbparms MB7BROKER -n
ftp::ftpid -u ftpuser -p ftppwd
BIP8071I: Successful command completion.

C:\Program Files\IBM\MQSI\7.0>
```

Figure 8-20 Security identity definition

3. Restart all execution groups that require this new security identity.

Adding a new FtpServer configurable service

As the message flow reads a file from a FTP server, an FtpServer configurable service with a security identity is required to logon to the FTP server. You can define the FtpServer configurable service using the `mqsicreateconfigurableservice` command in the command console as described in 8.1.3, “FtpServer configurable service for remote file processing” on page 291.

You can also use the WebSphere Message Broker Explorer to define the configurable service. The WebSphere Message Broker Explorer is a graphical user interface that is based on the Eclipse platform for administering the brokers and adds the Brokers and Broker Archive Files folders to the MQ Explorer - Navigator view. We use the WebSphere Message Broker Explorer in this scenario.

To add a new FtpServer configurable service using the WebSphere Message Broker Explorer:

1. In the **Navigator** view, expand the broker on which you want to add a new configurable service, as shown in Figure 8-21 on page 313.

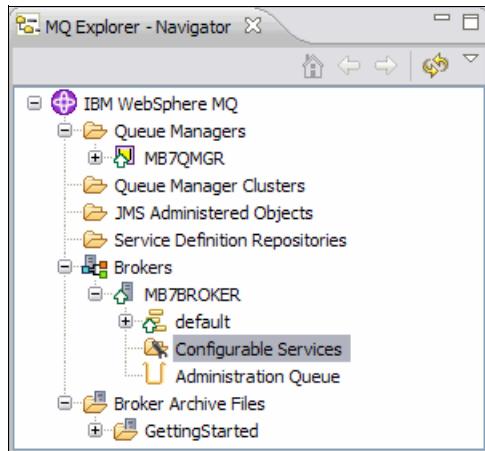


Figure 8-21 Navigator view

2. Right-click the **Configurable Services** folder under the expanded broker, and click **New** → **Configurable service**. The Configurable Service wizard window is displayed.
3. Provide a name for the configurable service. In this case, we use **FtpService** for the name, and select **FtpServer** as the type of configurable service to create.
4. Provide a security identity for the **securityIdentity** property. We use the security identity that was defined previously, **ftpid**, for this scenario.
5. Provide an IP address or a URL, of the FTP server, **1kfydky.itso.ral.ibm.com** in this scenario, for the **serverName** property, as shown in Figure 8-22 on page 314. For all of the other properties, we use the default values or the values that are provided from the node properties.
6. Click **Finish** to create the new **FtpServer** configurable service.
7. Restart the broker for the change to take effect.

The following command creates exactly the same **FtpServer** configurable service, and the command for this wizard configuration can also be seen in the bottom of the window, as also shown in Figure 8-22 on page 314:

```
mqsicreateconfigurableservice MB7BROKER -c FtpServer -o FtpService -n  
serverName,securityIdentity -v 1kfydky.itso.ral.ibm.com,ftpid
```

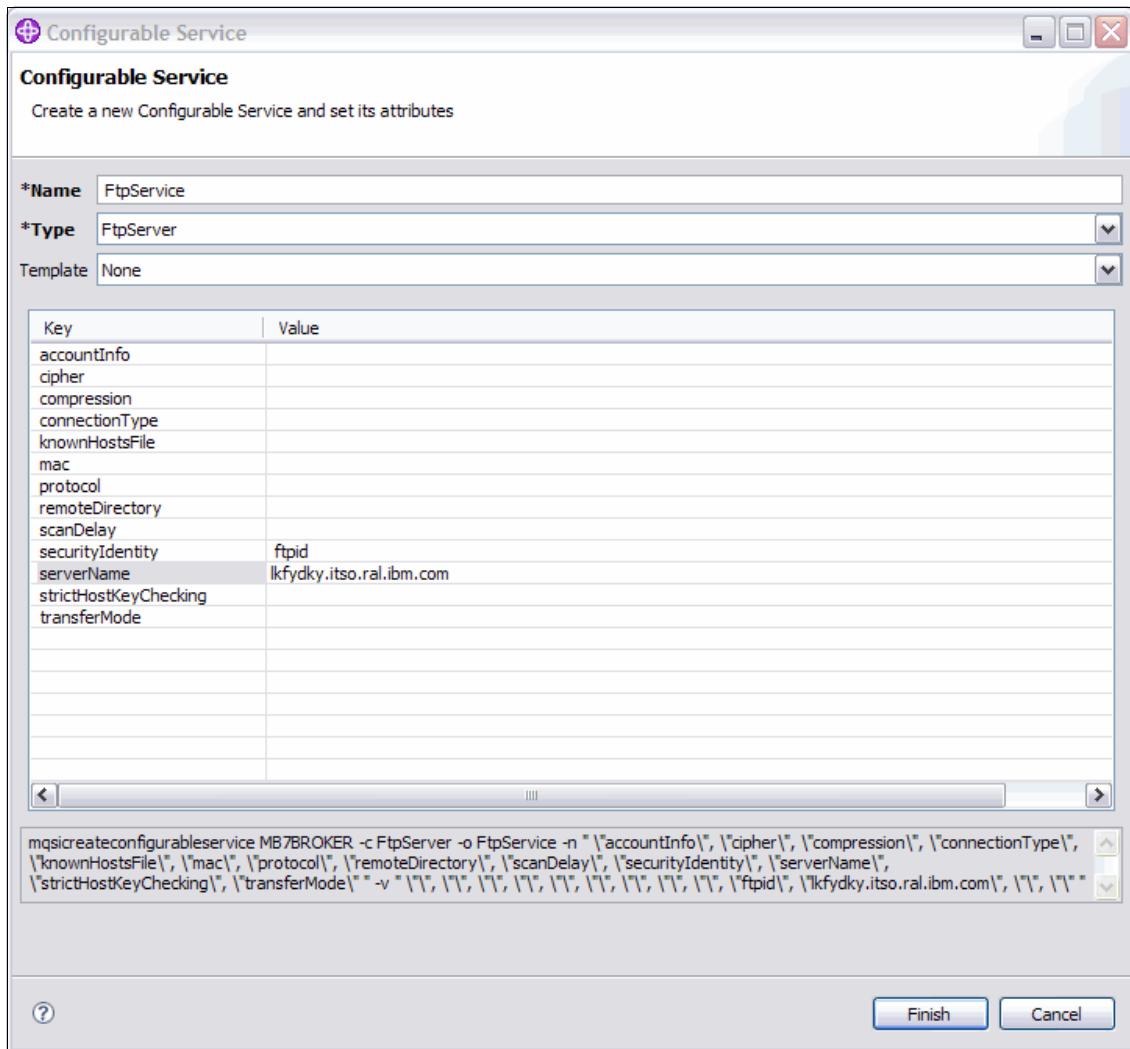


Figure 8-22 Configurable Service window

8.4.2 Creating a message definition

Because the pattern does not generate a message set for the message flows, you must create an empty message definition file and populate it.

To create a new message set for this scenario:

1. To create a new message set called **FileProcessingMsgSet**, navigate to **File → New → New Message Set**. In the wizard window, select **Binary data**

(for example, C or COBOL structures) as the type of message data from the selection list. The input data in this scenario is a fixed length format, as shown in Figure 8-23.

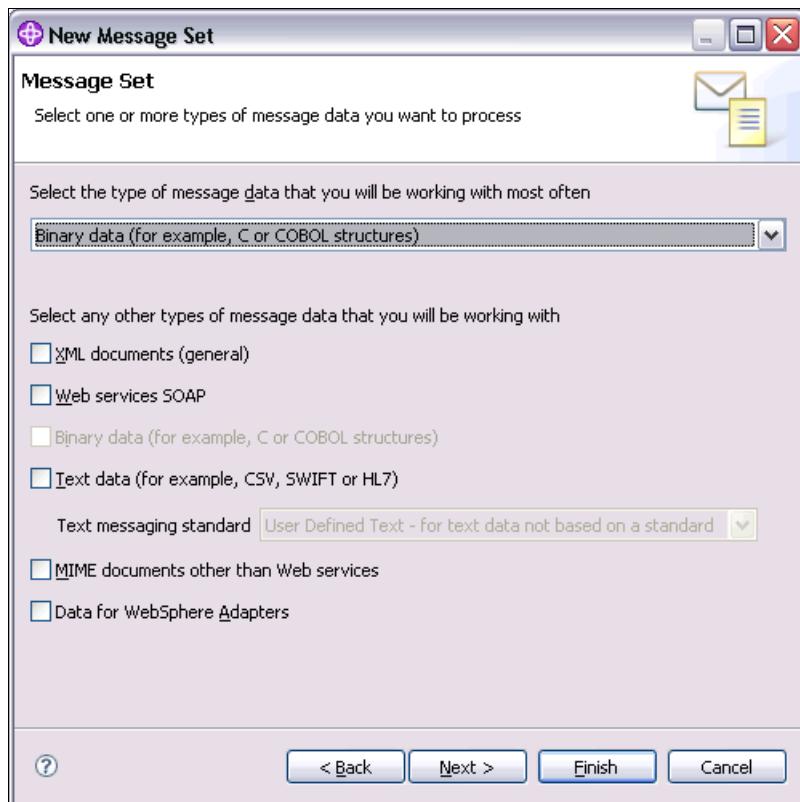


Figure 8-23 New Message Set window

A message set that supports the MRM message domain and a Custom Wire Format (CWF) called **Binary1** is created. Select **Binary1** as the Default wire format in the MRM domain. Figure 8-24 on page 316 shows the message set.

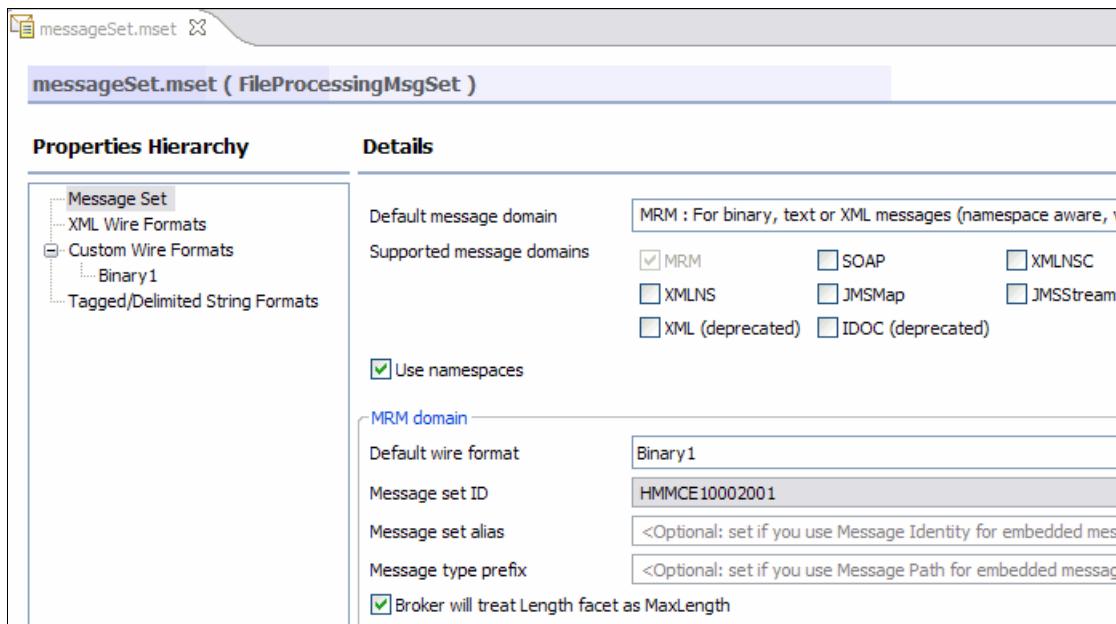


Figure 8-24 Message set definition

2. In the Broker Development view, right-click the **FileProcessingMsgSet** project, and select **New** → **Message Definition File** to open a New Message Definition File window.
3. Provide **employeeMsg**, for the Message definition file name, and click **Finish** to create **employeeMsg.mxsd** as shown in Figure 8-25. Double-click the **employeeMsg.mxsd** file to open a Message Definition Editor, if it has not already opened.

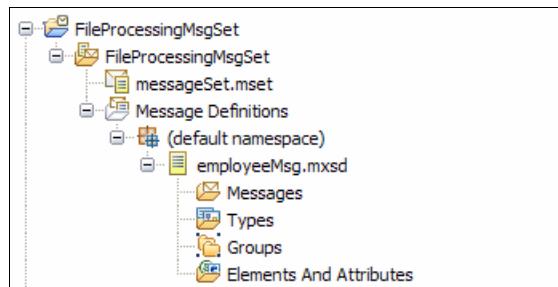


Figure 8-25 FileProcessingMsgSet Project

4. In the editor, under **employeeMsg.mxsd**, right-click **Messages**, and select **Add Message** to create a message called **employee**.

5. Right-click **employee** and use **Add Local Element** to add local elements to the message. The Data Type, Physical Type and Length for each element must match the message structure read from the file. Refer to “Record structure” on page 309 for record layout. See Figure 8-26 for the message definition for this scenario.

The screenshot shows a software interface titled "employeeMsg.mxsd". The main window is titled "Elements and Attributes". Below the title bar is a toolbar with icons for "New", "Open", "Save", "Print", "Exit", "Copy", "Paste", "Delete", "Find", "Replace", "Properties", and "Overview". The main area is a table with two columns: "Structure" and "Type". The "Structure" column contains a tree view of XML elements, and the "Type" column contains their corresponding XSD types. The tree view starts with "Messages", which contains "employee". "employee" has eleven child elements: "EmpNo", "FirstName", "MidInt", "LastName", "WorkDept", "PhoneNo", "HireDate", "Job", "BirthDate", "Salary", "Bonus", and "Comm". All these elements are of type "xsd:string". There are also three collapsed categories: "Types", "Groups", and "xsd:types". At the bottom of the table are tabs for "Overview" and "Properties".

Structure	Type
Messages	
employee	complexType
EmpNo	xsd:string
FirstName	xsd:string
MidInt	xsd:string
LastName	xsd:string
WorkDept	xsd:string
PhoneNo	xsd:string
HireDate	xsd:string
Job	xsd:string
BirthDate	xsd:string
Salary	xsd:string
Bonus	xsd:string
Comm	xsd:string
+ Types	
+ Groups	

Figure 8-26 Message definition

Figure 8-27 on page 318 shows the properties of the local element, *EmpNo*. The Physical Type is Fixed Length String.

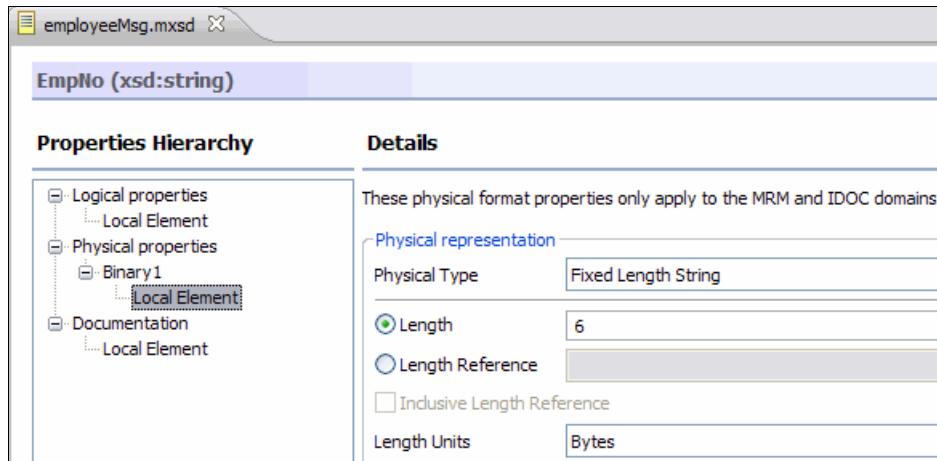


Figure 8-27 Local element EmpNo properties

8.4.3 Generating a pattern instance

In this scenario, the message flows are generated from the File Processing pattern. The Toolkit workbench creates the pattern resources based on the pattern parameters that are provided. With a different pattern parameter configuration, unique pattern instances can be created from each pattern. The resources for each pattern instance are contained within a single Pattern Instance project.

To generate a pattern instance, perform the following actions on the WebSphere Message Broker Toolkit:

1. In the WebSphere Message Broker Toolkit, switch to the Broker Application Development perspective.
2. Open the Patterns Explorer view, and select **Patterns** → **File Processing** → **Record Distribution** → **MQ one-way**, as shown in Figure 8-28 on page 319.

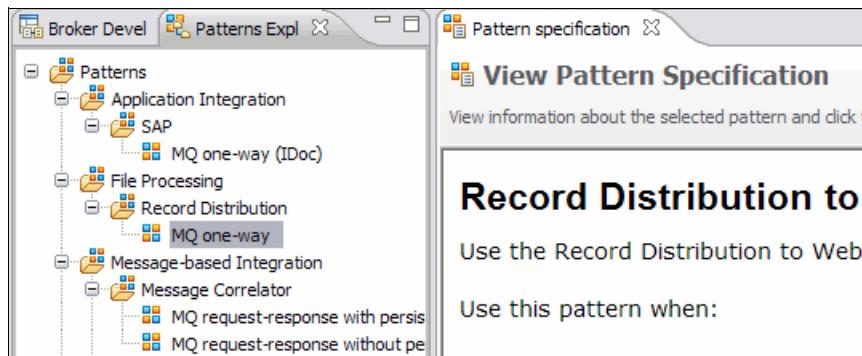


Figure 8-28 Pattern Explorer

3. The Pattern specification view displays information about the selected pattern. To open the New Pattern Instance wizard, go to the bottom of the Pattern specification view, and click **Create New Instance**.
4. Provide a name for the pattern instance, *FileProcessing* in this scenario, as shown in Figure 8-29, and click **OK**.

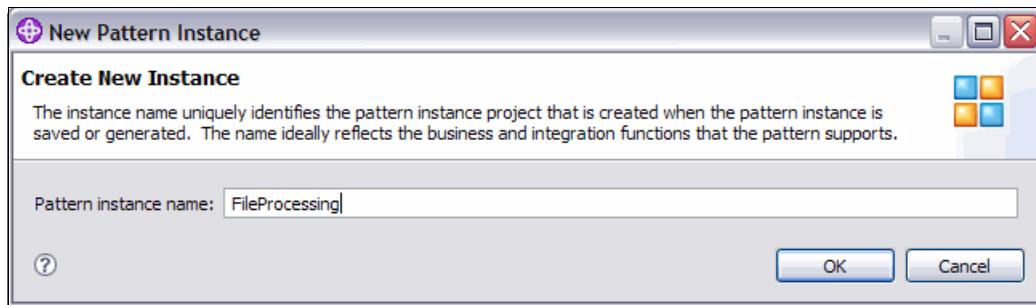


Figure 8-29 New Pattern Instance wizard

5. Now the Configure Pattern Parameters page is displayed and the **Generate** button is still inactive because all required parameters are not completed. The first message with a red x mark is The pattern parameter "Directory of input file" is mandatory but a value is not set, which indicates what is expected as the first parameter to start with, as shown in Figure 8-30 on page 320.
- Pattern Parameters Details are available from the right pane of the pattern instance configuration panel and in the toolkit help.

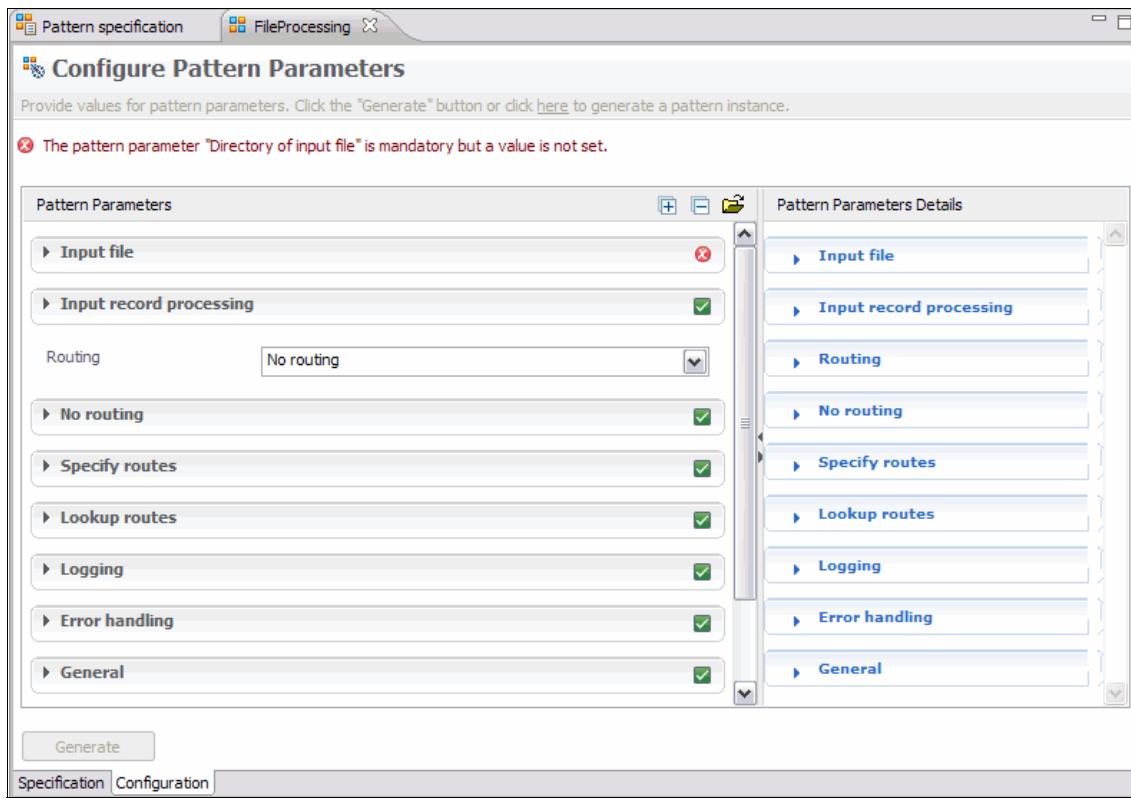


Figure 8-30 Configure Pattern Parameters view

6. Provide the required information for the Input file section, as shown in Figure 8-31.

A detailed view of the 'Input file' configuration dialog. It has a header 'Input file' with a checkmark icon. Below it is a sub-header 'Details of how to access the input data'. The fields are:

- 'Directory of input file *' with value '.\input'
- 'File pattern *' with value 'input.txt'
- 'Use FTP *' with checked checkbox
- 'FTP configurable service *' with value 'FtpService'

Figure 8-31 Input file parameters

In the window in Figure 8-31, complete these fields:

- a. Directory of input file: `./input`

This is not a remote directory in a FTP server. The value entered here defines the Input directory property on the Basic tab in the FileInput Node Properties view. The remote file is transferred to the local directory that is defined in this property before the actual processing starts.

- b. File pattern: input.txt
- c. Use FTP: Selected
- d. FTP configurable service: FtpService

We use the FTP configurable service that was created previously for this scenario.

7. Expand the Input record processing section, and select **Delimited (binary or text)** for Record detection because a single line in the input text file in this scenario is a single record, and each line is terminated by a carriage return (X'0D') and line feed (X'0A') pair of characters (on a Windows system). The parameters that are not required for this selection are automatically disabled. Provide the required information, as shown in Figure 8-32 on page 322:

- a. Record detection: **Delimited (binary or text)**
- b. Delimiter type: **Dos or UNIX line end**
- c. CCSID of file: **Broker system default** (Default)

You can also type the value in this field.

- d. Encoding of file: **Broker System Determined** (Default)

If **Fixed length (binary or text)** is selected for the record detection in this case, be careful about determining the length of a record. Even if the actual data of a record is 98 in this scenario, the value for the Record length property should be 100 bytes including 2 bytes of the Windows line end characters. The record length might be different in brokers that run on other operating systems because the line end characters vary across operating systems.

Because the scenario uses the FileInput node to segment the input file into messages that are to be parsed by the MRM Custom Wire Format (CWF) parser using a message definition, **Parsed (binary or text)** is also a possible option.

If this record detection option is selected, the Input message set, the Input message type, and Input message format parameters are activated, and available choices for these parameter values are displayed in the selection lists. However, this selection also requires a careful consideration on line end characters.

Input record processing

Specify how file records are to be processed.

Record detection	Delimited (binary or text)
Record length *	80
Delimiter type *	Dos or UNIX line end
Record delimiter *	
Input message set *	
Input message type *	
Input message format *	
CCSID of file *	Broker system default
Encoding of file *	Broker System Determined

Figure 8-32 Input record processing parameters

- As we implement the PHPCompute node to route messages to the destinations, keep **No routing** selected for Routing and keep the default values for the No routing section, as shown in Figure 8-33.

Pattern Parameters

Routing	No routing
No routing	
Specify the single output destination	
Output queue manager	
Output queue *	OUTPUT

Figure 8-33 No routing parameters

- Expand the Logging section, and select **Logging required**. Provide a name for Log queue, *LOG.LQ* in this case. The Log queue manager pattern parameter is left blank because the broker queue manager is used for logging. See Figure 8-34 on page 323.

Logging

Enable logging messages and specify their destination.

Logging required *

Log queue manager

Log queue * LOG.LQ

Figure 8-34 Logging parameters

10. In the Error handling section, the Error message required property is already selected as a default. Provide a name for Error Queue, *ERROR.LQ* in this scenario, as shown in Figure 8-35. The Error queue manager pattern parameter is left empty because the broker queue manager is used for logging.

Error handling

Enable error messages and specify their destination.

Error message required *

Error queue manager

Error queue * ERROR.LQ

Figure 8-35 Error handling parameters

11. Keep the default values for the parameters in the General section, as shown in Figure 8-36.

General

Specify naming conventions and description

Broker schema mqsi

Flow prefix

Flow suffix

Queue prefix

Queue suffix

Short description

Long description

Figure 8-36 General parameters

12. Click **Generate** to generate a pattern instance.

8.4.4 Generated artifacts

Upon successful completion, the pattern instance generation summary report is displayed along with generated artifacts. A Pattern Instance project is created in the workspace and is displayed in the Broker Development view, as shown in Figure 8-37.

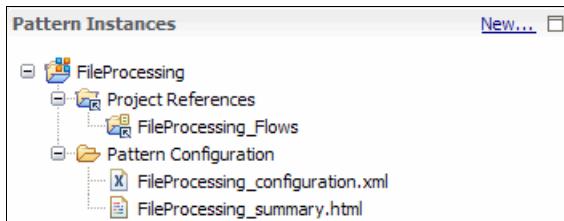


Figure 8-37 A Pattern Instance project

The Pattern Instance project contains Project References and Pattern Configuration. The Project References section shows the message flow project that was created by the pattern instance generation.

The Pattern Configuration contains a configuration file and a summary file. The summary file describes the tasks that were generated and the tasks that are still required to be completed. With the configuration file, you can regenerate the pattern instance, but any existing projects are deleted.

Figure 8-38 on page 325 shows the generated message flow project that contains message flows and ESQLs.

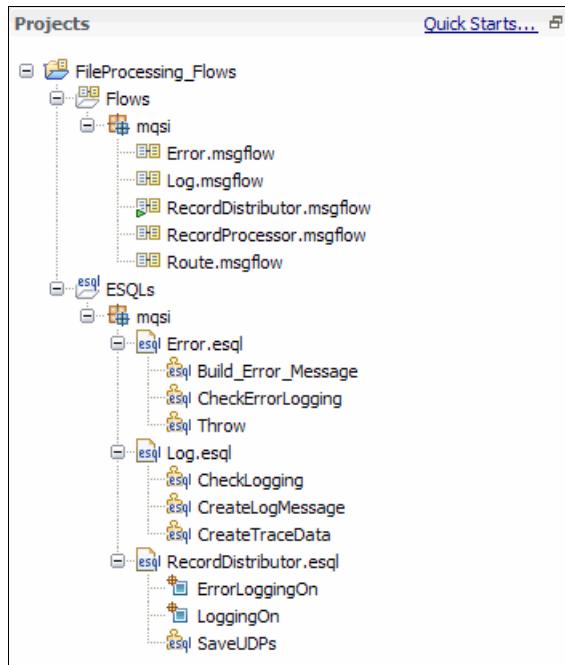


Figure 8-38 A message flow project

Record Distributor message flow

Figure 8-39 shows the Record Distributor message flow that is generated from the pattern.

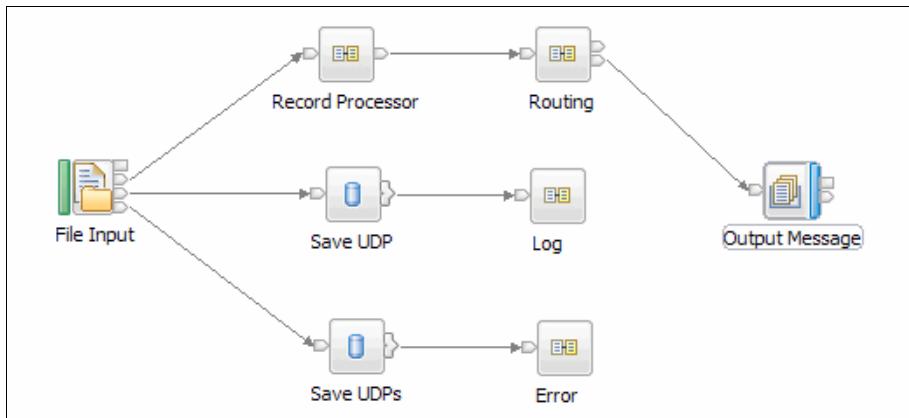


Figure 8-39 RecordDistributor.msgflow

The flow performs the following actions:

1. A remote file is selected for processing in accordance with the file input parameters.
2. Records are read from the file, as defined in the record detection parameters. Each record is propagated as a separate message flow.
3. A message is sent to the Record Processor subflow for further process.
4. The routing is called to determine the correct destination, but in this generated instance, no routing occurs. The output message from the Record Process subflow directly goes to the default destination.
5. The Error subflow is called if errors occur because the user-defined property ErrorLoggingOn that is saved in the database node is set to true. See Example 8-8.
6. At the end of the data processing, the Log subflow is called to record completion because the user-defined property LoggingOn that is saved in the database node is set to true.

Example 8-8 RecordDistributor.esql

```
BROKER SCHEMA mqsi
-- Generated by com.ibm.etools.mft.pattern.fp.rd.mq Version 1.0
-- $MQSI patternName=com.ibm.etools.mft.pattern.fp.rd.mq MQSI$
-- $MQSI patternVersion=1.0 MQSI$

DECLARE ErrorLoggingOn EXTERNAL BOOLEAN TRUE;
DECLARE LoggingOn EXTERNAL BOOLEAN TRUE;

-- This uses a database node as it is (currently) the most
-- efficient way of making changes to the local environment
CREATE DATABASE MODULE SaveUDPs
    CREATE FUNCTION Main() RETURNS BOOLEAN
        BEGIN
        -- This module saves the logging UDPs to the environment to make them
        available to any subflows
        -- UDPs are handled this way (rather than by promotion) to enable
        subflow substitution
        -- without imposing UDP definitions
            SET Environment.PatternVariables.ErrorErrorLoggingOn;
            SET Environment.PatternVariables.LoggingOn;

        RETURN TRUE;
    END;
END MODULE;
```

Record Processor subflow

Figure 8-40 shows the Record Processor subflow.

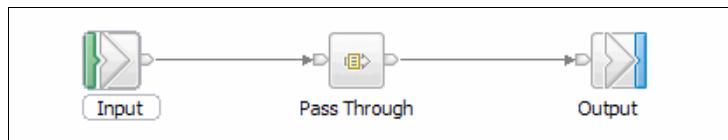


Figure 8-40 RecordProcessor.msgflow

This subflow contains only a Passthrough node and does nothing in this scenario. This subflow is, by default, designed to provide a place for customizing in a pattern instance without damaging the pattern structure when the instance is generated by the Record Distribution pattern.

Log subflow

Figure 8-41 shows the Log subflow.

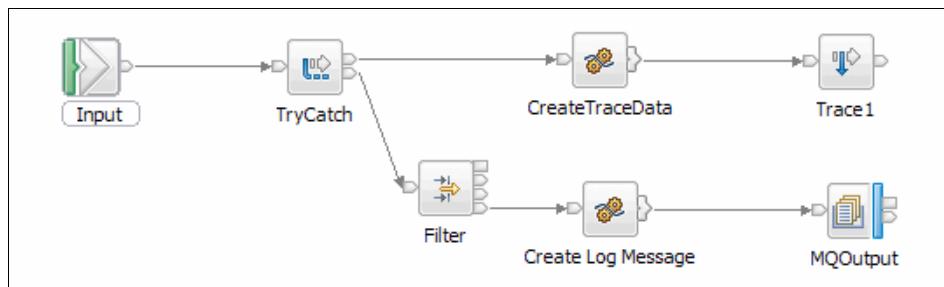


Figure 8-41 Log.msgflow

The Log subflow is called to write a log message when the file processing is completed. Logging is controlled by the user-defined property, LoggingOn, which is saved in the database node, Save UDP and Save UDPS, in the Record Distributor message flow. The database node saves the logging UDPS to the environment to make them available to any subflows.

Log messages are written as persistent WebSphere MQ messages to a WebSphere MQ queue. Example 8-9 on page 328 shows the sample of log messages. Example 8-10 on page 328 shows a sample ESQL log.

The MQRFH2 header of the log message contains the following information:

- ▶ Broker name
- ▶ Message flow name
- ▶ Time stamp

The XML body of the message contains the following information about the file and records:

- ▶ Directory
- ▶ File name
- ▶ Records total
- ▶ Records to default

Example 8-9 A log message example

```
<CodedCharSetId>437</CodedCharSetId>
<Encoding>546</Encoding>
<Log>
  <Directory>C:\FileProcessing\.\\input</Directory>
  <FileName>input.txt</FileName>
  <RecordTotal>42</RecordTotal>
</Log>
```

Example 8-10 Log.esql

```
BROKER SCHEMA mqsi
-- Generated by com.ibm.etools.mft.pattern.fp.rd.mq Version 1.0
-- $MQSI patternName=com.ibm.etools.mft.pattern.fp.rd.mq MQSI$
-- $MQSI patternVersion=1.0 MQSI$I$
CREATE Compute MODULE CreateLogMessage

CREATE FUNCTION main() RETURNS BOOLEAN
    BEGIN
        SET OutputRoot.Properties = NULL;
-- Create Headers
        CREATE FIRSTCHILD OF OutputRoot DOMAIN ('MQMD') NAME 'MQMD';
        DECLARE MQMDRef REFERENCE TO OutputRoot.MQMD;
        SET MQMDRef.Version = MQMD_CURRENT_VERSION;
        SET MQMDRef.CodedCharSetId =
InputRoot.Properties.CodedCharSetId;
        SET MQMDRef.Encoding = InputRoot.Properties.Encoding;
        SET MQMDRef.Format = MQFMT_RF_HEADER_2;
        DECLARE OutRef REFERENCE TO OutputRoot;
        CREATE NEXTSIBLING OF MQMDRef AS OutRef DOMAIN('MQRFH2') NAME
'MQRFH2';
        SET OutputRoot.MQRFH2.(MQRFH2.Field)Version = 2;
```

```

-- Define basic logging data
    SET OutRef.CodedCharSetId =
InputRoot.Properties.CodedCharSetId;
    SET OutRef.Encoding = InputRoot.Properties.Encoding;
    SET OutRef.usr.BrokerName = SQL.BrokerName;
    SET OutRef.usr.MessageFlowLabel = SQL.MessageFlowLabel;
    SET OutRef.usr.DTSTAMP = CURRENT_TIMESTAMP;
    CREATE NEXTSIBLING OF OutRef AS OutRef DOMAIN('XMLNSC') NAME
'XMLNSC';
-- Add file and record information
-- Do not log file contents as these are available in the archive
directory
    -- Log data on file and records in XMLNSC body
    SET OutputRoot.XMLNSC.Log.Directory =
InputLocalEnvironment.File.Directory;
    SET OutputRoot.XMLNSC.Log.FileName =
InputLocalEnvironment.File.Name;
    SET OutputRoot.XMLNSC.Log.RecordTotal =
InputLocalEnvironment.File.Record;

END;
END MODULE;

CREATE Compute MODULE CreateTraceData
CREATE FUNCTION main() RETURNS BOOLEAN BEGIN
    DECLARE EnvVarRef REFERENCE TO Environment.PatternVariables;
    SET EnvVarRef.DTSTAMP = CURRENT_TIMESTAMP;
    SET EnvVarRef.BrokerName = SQL.BrokerName ;
    SET EnvVarRef.MessageFlowlabel = SQL.MessageFlowLabel;
-- Add file and record information
    SET EnvVarRef.File.Directory = InputLocalEnvironment.File.Directory;
    SET EnvVarRef.File.FileName = InputLocalEnvironment.File.Name;
    SET EnvVarRef.File.RecordTotal = InputLocalEnvironment.File.Record;
    SET EnvVarRef.File.RecordsToDefault =
Environment.PatternVariables.DefaultRouting;

RETURN TRUE;
END;
END MODULE;

CREATE FILTER MODULE CheckLogging
CREATE FUNCTION main() RETURNS BOOLEAN BEGIN

RETURN Environment.PatternVariables.LoggingOn;

```

```
END;  
  
END MODULE;
```

Error subflow

This subflow writes an error message if an error is caught during processing in the message flow, depending on the user-defined property, ErrorLoggingOn, which is also saved in the database node. Figure 8-42 shows the Error subflow.

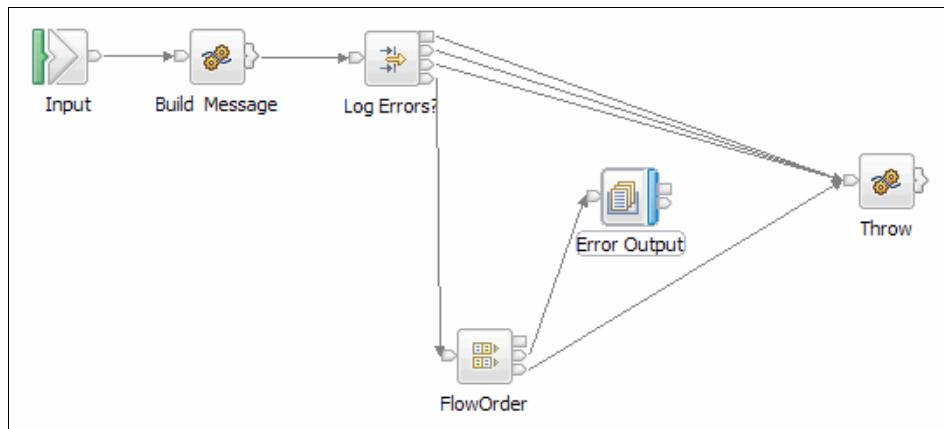


Figure 8-42 Error.msgflow

If an error occurs, an XML message that describes the details of the exception is created and written to the error queue. The sample of error messages is shown in Example 8-11 on page 331. Example 8-12 on page 331 shows the .esql log.

The XML message contains the following details:

- ▶ Broker name
- ▶ Message flow name
- ▶ Time stamp
- ▶ File and records information:
 - directory
 - File name
 - Record number, when error occurred
- ▶ Summary of the exception data:
 - Message flow label

- Error number
- Text description
- A list of inserts giving further details.

Example 8-11 An error message sample

```
<Error>
  <BrokerName>MB7BROKER</BrokerName>
  <MessageFlowLabel>mqsi.RecordDistributor</MessageFlowLabel>
  <DTSTAMP>2009-12-9T20:26:03.676078</DTSTAMP>
  <File>
    <Directory>C:\FileProcessing\.\


---



```

Example 8-12 Error.esql

```
BROKER SCHEMA mqsi
-- Generated by com.ibm.etools.mft.pattern.fp.rd.mq Version 1.0
-- $MQSI patternName=com.ibm.etools.mft.pattern.fp.rd.mq MQSI$
-- $MQSI patternVersion=1.0 MQSI$

CREATE FILTER MODULE CheckErrorLogging
  CREATE FUNCTION Main() RETURNS BOOLEAN
  BEGIN
    RETURN Environment.PatternVariables.ErrorLoggingOn;
  END;
  END MODULE;

CREATE COMPUTE MODULE Build_Error_Message
  CREATE FUNCTION Main() RETURNS BOOLEAN
  BEGIN
    SET OutputRoot.Properties = NULL;
    -- Create MQMD
    DECLARE MQMDRef REFERENCE TO OutputRoot.MQMD;
    CREATE FIRSTCHILD OF OutputRoot AS MQMDRef DOMAIN ('MQMD') NAME
'MQMD';
```

```

SET MQMDRef.Version = MQMD_CURRENT_VERSION;
SET MQMDRef.ApplIdentityData = SQL.BrokerName;
SET MQMDRef.CodedCharSetId = InputRoot.Properties.CodedCharSetId;
SET MQMDRef.Encoding = InputRoot.Properties.Encoding;

DECLARE OutRef REFERENCE TO OutputRoot.XMLNSC;
CREATE NEXTSIBLING OF MQMDRef AS OutRef DOMAIN('XMLNSC') NAME
'XMLNSC';
-- Define the standard error fields
SET OutRef.Error.BrokerName = SQL.BrokerName;
MOVE OutRef TO OutputRoot.XMLNSC.Error;

SET OutRef.MessageFlowLabel = SQL.MessageFlowLabel;
SET OutRef.DTSTAMP = CURRENT_TIMESTAMP;
-- and some file info
SET OutRef.File.Directory = InputLocalEnvironment.File.Directory;
SET OutRef.File.FileName = InputLocalEnvironment.File.Name;
SET OutRef.File.RecordNumber = InputLocalEnvironment.File.Record;
DECLARE I Integer 0;
-- Add exception data
Call AddExceptionData();
END;

CREATE PROCEDURE AddExceptionData() BEGIN

DECLARE ERef REFERENCE TO OutputRoot.XMLNSC.Error;
-- Add some exception data for error and fault
DECLARE Error INTEGER;
DECLARE Text CHARACTER;
DECLARE Label CHARACTER;
Declare FaultText CHARACTER 'Exception data:';
DECLARE I INTEGER 1;
DECLARE K INTEGER;
DECLARE start REFERENCE TO InputExceptionList.*[1];

WHILE start.Number IS NOT NULL DO
    SET Label = start.Label;
    SET Error = start.Number;
    IF Error = 3001 THEN
        SET Text = start.Insert.Text;
    ELSE
        SET Text = start.Text;
    END IF;
    -- Don't include the "Caught exception and rethrowing message"
    IF Error <> 2230 THEN

```

```

-- Process inserts
DECLARE Inserts Character;
DECLARE INS Integer;
SET Inserts = '';
-- Are there any inserts for this exception
IF EXISTS (start.Insert[]) THEN
    -- If YES add them to inserts string
    SET Inserts = Inserts ||
COALESCE(start.Insert[1].Text,'NULL')|| ' / ';
    SET K = 1;
    INSERTS: LOOP
        IF CARDINALITY(start.Insert[])> K
        THEN
            SET Inserts = Inserts ||
COALESCE(start.Insert[K+1].Text,'NULL')|| ' / ';
            -- No more inserts to process
            ELSE LEAVE INSERTS;
            END IF;
            SET K = K+1;
            END LOOP INSERTS;
        END IF;
        SET ERef.Exception[I].Label = Label;
        SET ERef.Exception[I].Error = Error;
        SET ERef.Exception[I].Text = Text;
        Set ERef.Exception[I].Inserts = COALESCE(Inserts, '');
        SET FaultText = FaultText || ' Label: ' || COALESCE(Label,
        '');
        SET FaultText = FaultText || ' Error: ' ||
COALESCE(CAST(Error AS CHARACTER), '');
        SET FaultText = FaultText || ' Text: ' || COALESCE(Text,
        '');
        SET FaultText = FaultText || ' Inserts: ' ||
COALESCE(Inserts, '');

        SET I = I+1;
    END IF;
    -- Move start to the last child of the field to which it
    currently points
    MOVE start LASTCHILD;
END WHILE;
SET Environment.PatternVariables.FaultText = FaultText;
END;
END MODULE;
CREATE COMPUTE MODULE Throw

```

```
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
    THROW USER EXCEPTION SEVERITY 3 MESSAGE 2372
VALUES(Environment.PatternVariables.FaultText);
    RETURN FALSE;
END;
END MODULE;
```

Routing subflow

Figure 8-43 shows the Routing subflow.

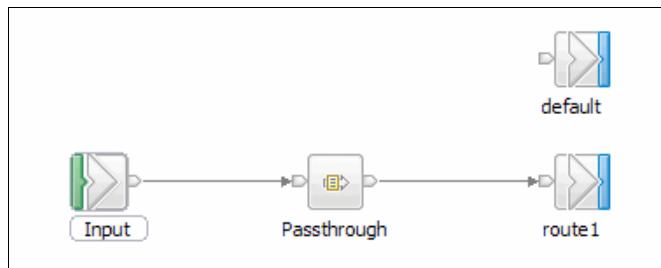


Figure 8-43 Route.msgflow

The routing that is generated in this scenario contains only a Passthrough node because no routing is selected. The subflow also includes both default and route1 terminals to maintain the consistency with all other types of routing subflow, but the single route1 terminal is the only node that is connected. The PHPCompute node is used instead of the pass-through node to make the actual routing occur in this scenario.

8.4.5 Implementing message flow

The generated message flows use a message set that contains a message definition to parse records that are read from the file. The message definition describes the MRM message structure for the flows, and the PHPCompute node uses the data that is parsed using the message definition to route incoming messages to their destinations.

Modifying the Record Distributor message flow

In this scenario, the Record Distributor message flow uses label nodes for message routing.

To modify the Record Distributor message flow:

1. The message set project for this scenario, *FileProcessingMsgSet*, should be referenced by the generated message flow project, *FileProcessing_Flows*. In the Broker Development view, right-click the **FileProcessing_Flows project**, and select **Properties**. In Project References on the left pane of the Properties for FileProcessing_Flows window, Figure 8-44, ensure that the *FileProcessingMsgSet* project is selected.

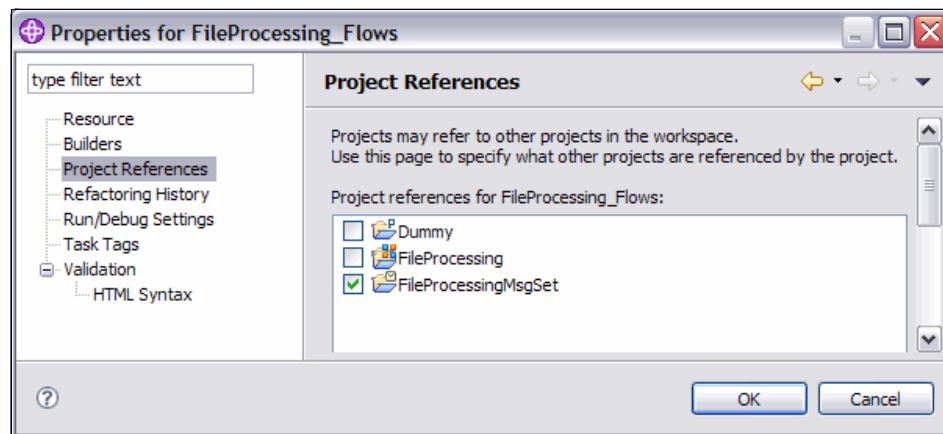


Figure 8-44 Project References

2. Open the Record Distributor message flow with Message Flow Editor. In the Input Message Parsing tab on the FileInput Node Properties view, select the required message information, as shown in Figure 8-45 on page 336. The values that you entered for Message coded character set ID and Message encoding in the pattern parameters in the pattern instance generation are already selected as the default values:
 - Message set: **FileProcessingMsgSet**
 - Message type: **employee**
 - Message Format: **Binary1**If the message set project is not referenced by the message flow project, the choices might not be available in the selection lists.

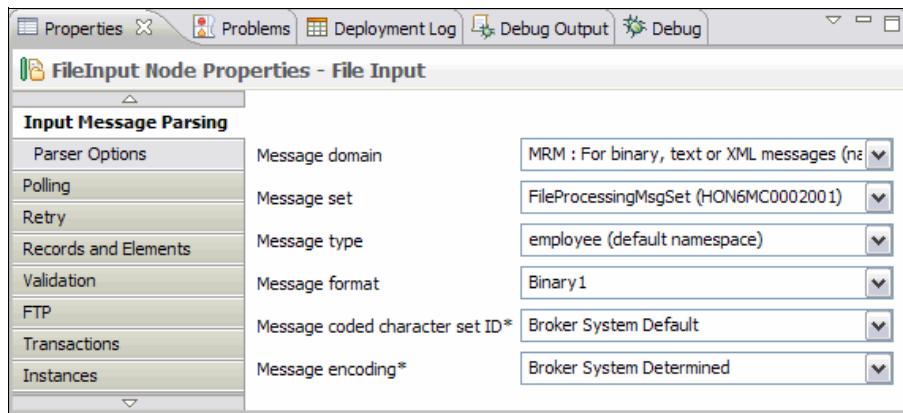


Figure 8-45 FileInput node properties for Input Message Parsing

3. On the FileInput Node Properties view, in the FTP tab, provide the required information, as shown in Figure 8-46. The values that you entered for the pattern parameters in the pattern instance generation are already provided to the corresponding fields:

 - Server directory: `./input`
 - Transfer mode: **ASCII**

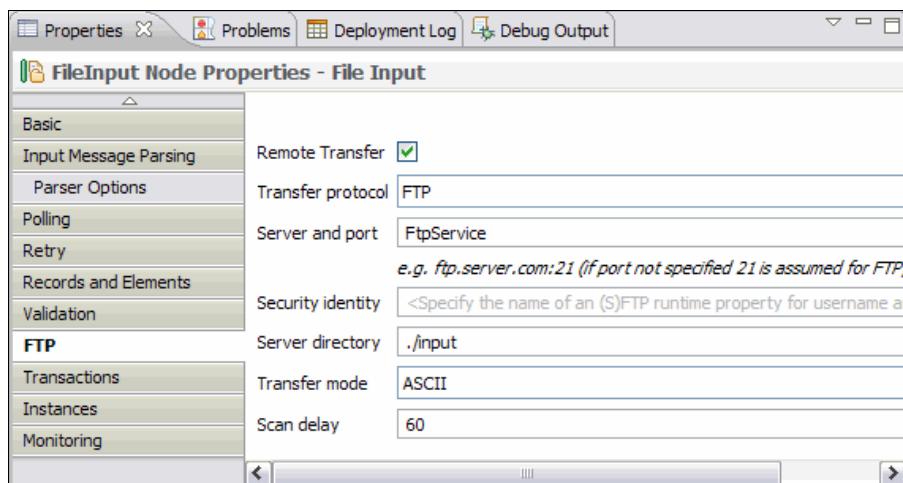


Figure 8-46 FileInput node properties for FTP

4. Remove the connection between the Routing subflow node and the Output Message node.

5. Rename the MQOutput node called Output Message node to *DEST1.LQ* and provide a name for the Queue name property in the Basic tab on the MQOutput Node Properties view, *DEST1.LQ* in this case.
6. Create two additional MQOutput nodes called *DEST2.LQ* and *DEST3.LQ*. The queue names are *DEST2.LQ* and *DEST3.LQ* respectively.
7. Create three Label nodes called *Dest1*, *Dest2*, and *Dest3*. The Label names are also called *Dest1*, *Dest2*, and *Dest3* respectively. Connect these Label nodes with the MQOutput nodes, as shown in Figure 8-47.

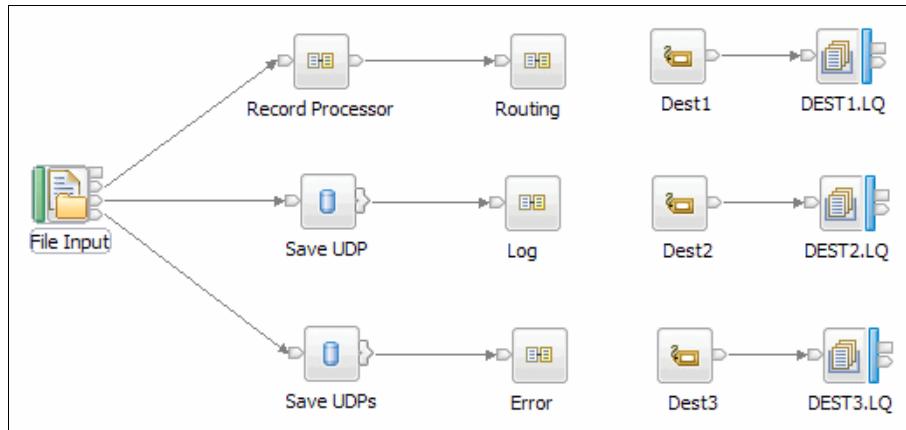


Figure 8-47 Record distributor message flow with label nodes

Routing message subflow modification

The scenario uses the PHPCompute node to route incoming messages to the destination. The PHP script propagates the messages directly to Label nodes.

To modify the Route message subflow called Routing:

1. Open the Route message subflow with the Message Flow Editor, and remove the Passthrough node, the default Output node, and the route1 Output node, and replace them with a PHPCompute node.
2. Connect the InputTerminal Input node with the PHPCompute node, as shown in Figure 8-48.

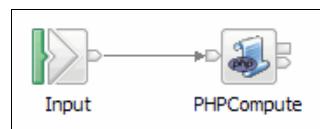


Figure 8-48 Route message flow with the PHPCompute node

3. In the Broker Development view, right-click the **FileProcessing_Flows** project, and select **New** → **Other** → **PHP** → **PHP File** to create a new PHP file called *route.php*. Use a PHP template called **New simple PHP file**, and click **Finish**.
4. Write PHP scripts for routing, as shown in Example 8-13.

Example 8-13 route.php

```
<?php
class Route {

    /**
     * @MessageBrokerSimpleTransform
     * @MessageBrokerRouter
     */
    function evaluate($output_assembly, $input_assembly) {
        // to propagate the Properties and MQMD folders
        // from the inbound message assembly
        $output_assembly->Properties = $input_assembly->Properties;
        $output_assembly->MQMD = $input_assembly->MQMD;

        // to store a reference to the input sub-tree
        $mrm = $input_assembly->MRM;

        $salary = $mrm->Salary->getValue();
        $bonus = $mrm->Bonus->getValue();
        $comm = $mrm->Comm->getValue();

        $sum = $salary + $bonus + $comm;

        switch ($sum) {
            case ($sum > 90000) :
                // to build the XMLNSC/employee structure
                $output_assembly->XMLNSC->employee->EmpNo =
trim($mrm->EmpNo->getValue());

                // to store a reference to the output sub-trees
                $employee = $output_assembly->XMLNSC->employee;

                $employee->FirstName =
trim($mrm->FirstName->getValue());
                $employee->MidInt = trim($mrm->MidInt->getValue());
                $employee->LastName = trim($mrm->LastName->getValue());
                $employee->WorkDept = trim($mrm->WorkDept->getValue());
                $employee->PhoneNo = trim($mrm->PhoneNo->getValue());
        }
    }
}
```

```

$employee->HireDate = trim($mrm->HireDate->getValue());
$employee->Job = trim($mrm->Job->getValue());
$employee->BirthDate =
trim($mrm->BirthDate->getValue());
$employee->Salary = trim($mrm->Salary->getValue());
$employee->Bonus = trim($mrm->Bonus->getValue());
$employee->Comm = trim($mrm->Comm->getValue());

$output_assembly->routeToLabel('dest1');
break;

case ($sum > 50000) :
$output_assembly->MRM = $mrm;
$output_assembly->routeToLabel('dest2');
break;

default :
$record = trim($mrm->EmpNo->getValue());
$record = $record.";" . trim($mrm->FirstName->getValue());
$record = $record.";" . trim($mrm->MidInt->getValue());
$record = $record.";" . trim($mrm->LastName->getValue());
$record = $record.";" . trim($mrm->WorkDept->getValue());
$record = $record.";" . trim($mrm->PhoneNo->getValue());
$record = $record.";" . trim($mrm->HireDate->getValue());
$record = $record.";" . trim($mrm->Job->getValue());
$record = $record.";" . trim($mrm->BirthDate->getValue());
$record = $record.";" . trim($mrm->Salary->getValue());
$record = $record.";" . trim($mrm->Bonus->getValue());
$record = $record.";" . trim($mrm->Comm->getValue());
blob = new MbsBlob();
blob->setValue($record);

// to build a BLOB message, add a child element, also
// called BLOB,
// to the BLOB domain element
$output_assembly->BLOB->BLOB = $blob;
$output_assembly->routeToLabel('dest3');
}
}
}
?>

```

In the PHP script, route.php, in Example 8-13 on page 338, the class name is Route. The class name must match the name of the PHP script, but the name is not case sensitive.

For annotation of the evaluation method, we specify @MessageBrokerSimpleTransform and @MessageBrokerRouter annotation in a JavaDoc style documentation block, as described in “Annotations” on page 296. The @MessageBrokerSimpleTransform annotation passes two parameters to the evaluate method. The first parameter is a reference to the output assembly, and the second parameter is a reference to the input assembly. The @MessageBrokerCopyTransform annotation is not required in this scenario because we create new message bodies for each destination.

The @MessageBrokerRouter annotation prevents the PHPCompute node from automatically propagating the output assembly to the ‘out’ terminal of the node at the end of the evaluate method, which is not necessary in this scenario. The routeToLabel() function in this example explicitly routes the output assembly to the label nodes.

The incoming messages belong to the MRM domain. These messages are switched to the different block of transformation codes based on the sum of three fields, Salary, Bonus, and Commission:

- Messages that meet the first condition (The sum is greater than 9000) are transformed to XML messages that belong to the XMLNSC domain.
 - Messages that meet the second condition (The sum is greater than 5000) are routed to their destination with no change.
 - All other messages are transformed to delimited messages that belong to the BLOB message domain.
5. In the PHPCompute Node Properties view, click the **Basic** tab, and select **Browse** for PHP Script Selection, and select *route.php*.
 6. Save all of the changes made, and run the test.

8.5 Testing the scenario

To test the message flows, create a Message Broker Archive file, and deploy the file to the broker runtime. The archive file should include the message flows, the message set, and the php file created for routing, as shown in Figure 8-49 on page 341.

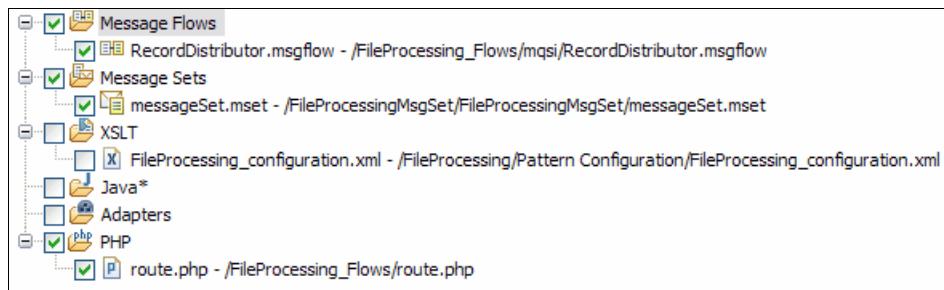


Figure 8-49 Message broker archive file creation

You can turn the error logging on or off by selecting the **ErrorLoggingOn** and **LoggingOn** options in the broker archive file, as shown in Figure 8-50.

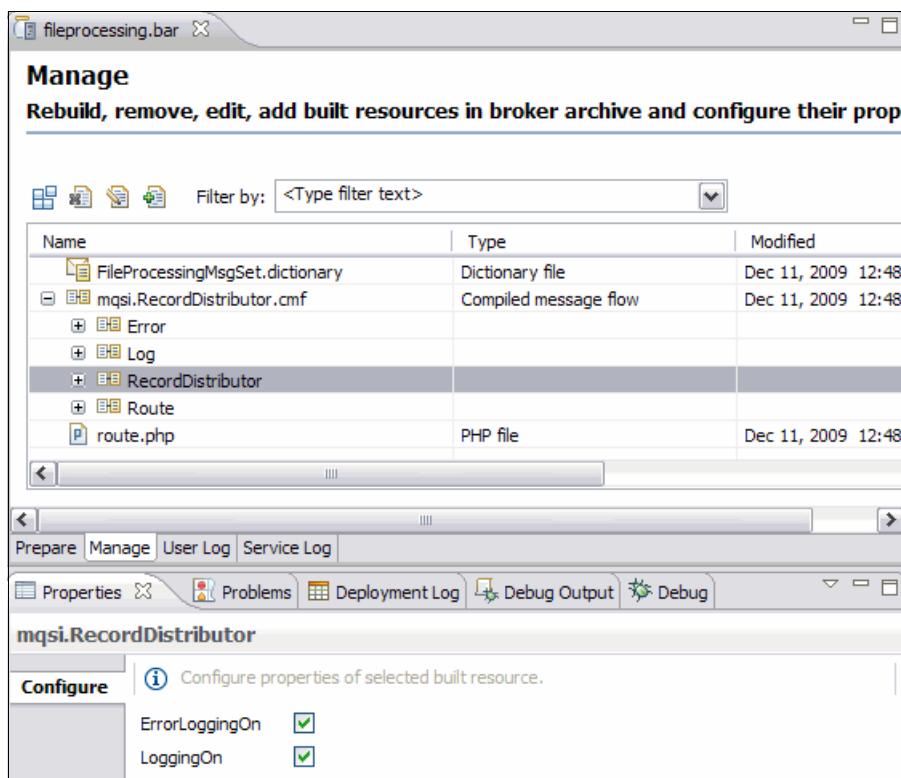


Figure 8-50 Manage and configure in the broker archive editor view

After deploying the archive file to the broker runtime, the **FileInput** node scans the remote directory for the input file every 60 seconds, which you can monitor from the FTP server log. After a file is found, the **FileInput** node transfers the file to the

local directory and the processing begins. In this scenario, each message is transformed to three different types of message formats and routed to the different destinations separately.

Figure 8-51 shows the successful test result. No error message is in *ERROR.LQ*.

Queue name	Queue type	Open input count	Open output count	Current queue depth	Max queue depth
DEST1.LQ	Local	0	1	5	5000
DEST2.LQ	Local	0	1	20	5000
DEST3.LQ	Local	0	1	17	5000
ERROR.LQ	Local	0	0	0	5000
LOG.LQ	Local	0	1	1	5000

Figure 8-51 Test Result

Example 8-14 shows the sample of messages in the *DEST1.LQ*.

Example 8-14 DEST1.LQ

```
<employee><EmpNo>000010</EmpNo><FirstName>CHRISTINE</FirstName><MidInt>
I</MidInt><LastName>HAAS</LastName><WorkDept>A00</WorkDept><PhoneNo>397
8</PhoneNo><HireDate>19950101</HireDate><Job>PRES</Job><BirthDate>19630
824</BirthDate><Salary>+0152750.00</Salary><Bonus>+0001000.00</Bonus><C
omm>+0004220.00</Comm></employee>
```

Example 8-15 shows the sample of messages in the *DEST2.LQ*.

Example 8-15 DEST2.LQ

```
000050JOHN          BGEYER           E01678919790817MANAGER
19550915+0080175.00+0000800.00+0003214.00
```

Example 8-16 shows the sample of messages in the *DEST3.LQ*.

Example 8-16 DEST3.LQ

```
000170;MASATOSHI;J;YOSHIMURA;D11;2890;19990915;DESIGNER;19810105;+00446
80.00;+0000500.00;+0001974.00
```

Example 8-17 shows a log message in the *LOG.LQ*.

Example 8-17 LOG.LQ

```
RFH.....<CodedCharSetId>437</CodedCharSetId><Encoding>546</Encoding><Log><Directory>C:\FileProcessing\.\input</Directory><FileName>input.txt</FileName><RecordTotal>42</RecordTotal></Log>
```



Scenario: Publish/subscribe using WebSphere Message Broker V7.0

In the scenario presented in this chapter, we describe and demonstrate the following scenarios:

- ▶ “Introduction to publish/subscribe” on page 344
- ▶ “Scenario 1: Comparison and difference” on page 349
- ▶ “Scenario 2: Migration” on page 379
- ▶ “Scenario 3: Content-based filtering” on page 402

9.1 Introduction to publish/subscribe

Publish/subscribe is a style of asynchronous messaging where the senders (publishers) of information are decoupled from the consumers (subscribers) of that information. Publishers publish messages without any knowledge of what or who can get these messages. Subscribers receive messages that are of interest to them without any knowledge of who published these messages. This decoupling of publishers and subscribers can allow for greater scalability and a more dynamic network topology.

Publish/subscribe applications are typically intended for situations where a single message is required by multiple users. A publish/subscribe application has one or more *publishers* who publish messages from an application to a *broker*, and a group of *subscribers* who subscribe to some or all of those published messages that are held on the broker. The system matches the publications to the subscribers and ensures that all of the messages are made available and delivered to all of the subscribers in a timely manner.

Figure 9-1 shows Publisher1 and Publisher2 publishing messages that are concerned with schedule information onto the broker. Subscribers can choose to subscribe or unsubscribe to that information that is available on the broker, as necessary.

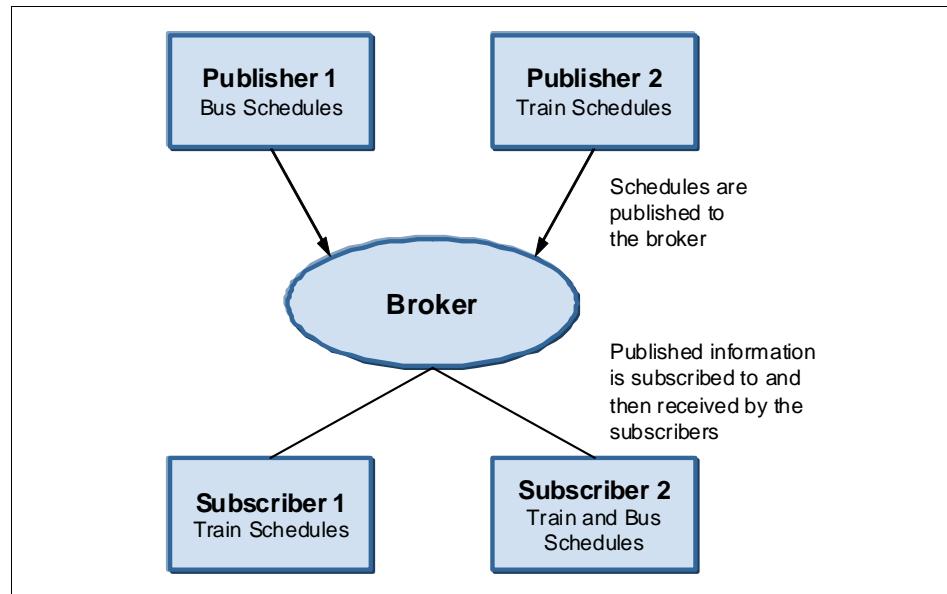


Figure 9-1 Basic Publish/subscribe

Multiple brokers can simply be connected together, enabling brokers to exchange messages, which allows subscribers to one of the brokers to pick up messages that were published to another broker, further freeing the subscriber from the constraints of using the same broker as the publisher, as shown in Figure 9-2.

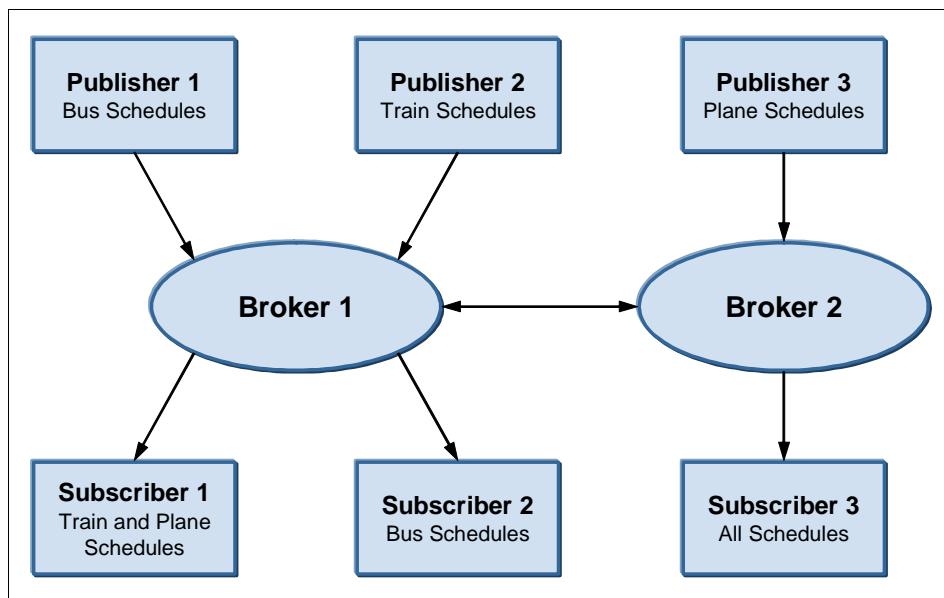


Figure 9-2 Extended Publish/Subscribe

The subscribers can choose between looking at all of the messages that are published or just some of the messages based upon various criteria that are important to them. This concept is known as either content-based or topic-based subscription, which we describe in more detail later in this chapter.

Publish/subscribe applications are widely used as a means of easily disseminating information to multiple users who might be interested in some or all of the information that is available. Additional subscribers can choose to either subscribe or unsubscribe as time goes by, and all of the subscribers are completely independent of each other. It's also worth noting that when a subscriber receives information, it can then go on to publish that material itself, possibly in a modified form, either back to the broker it got it from or to another broker.

In traditional systems, using point-to-point connections to provide this kind of service, the matrix of connections can grow exponentially, becoming very difficult to control and maintain. The root cause of the problem is that the messaging application always needs to know something about where the message is going, such as a queue manager name or a queue name. When high numbers of destinations are possible, the network view can become extremely complex.

A big advantage of a publish/subscribe system is that it can remove this unmanageable aspect of a point-to-point network and replace it with a very simple network of a publisher, a broker, and all of the subscribers. New subscription clients or services can simply be added without any impact or interruption in the service to other users, which provides a superior means of streamlined and efficient integration and growth across an enterprise, and because WebSphere MQ is used as the backbone for message delivery, all of the benefits and features of WebSphere MQ are inherited.

You can take advantage of the following capabilities of WebSphere MQ-based WebSphere Message Broker publish/subscribe:

- ▶ Enhanced publish/subscribe function through exploitation of structured topic names, access control, content-based subscriptions, and subscription points.
- ▶ Enhancement of message processing through the addition of new message processing nodes to complement or replace the previously supplied nodes.
- ▶ Interfaces that allow messages to be enriched with information from a database or to be stored in a database.

Here are some of the key features that can be used with WebSphere MQ Publish/Subscribe applications:

- ▶ Retained publications

By default, a queue manager discards a publication when it is sent to all interested subscribers. However, a publisher can specify that it wants the queue manager to keep a copy of a publication, which is then called a *retained publication*. The copy can be sent by the broker to subsequent subscribers who register an interest in the topic.

- ▶ Message persistence

Persistent messages in MQ are logged by the queue manager and are preserved when the queue manager restarts. Non-persistent messages might be lost if the queue manager fails, but are only delivered one time at the most, whereas persistent messages are guaranteed to be delivered exactly one time.

- ▶ Topic-based or content-based subscriptions

When you create a subscription, you subscribe to a specific topic, such as Weather, as in our Weather reporting application example. But you can also subscribe to specific content, which for example might be Weather at a particular zip code. The difference here is that the content-based subscription is filtering out information from the topic, giving the user a much more refined result. In the older versions of WebSphere Message Broker, this filtering took place only in the broker. However, in WebSphere MQ V7.0.1, some filtering capability was added. While MQ V7.0.1 engine supports filtering on the

header of the message only, WebSphere Message Broker supports filtering on the entire message, including the body of the message.

- ▶ Temporary subscriptions

It is possible that you can create a temporary subscription by specifying a temporary dynamic queue as the subscriber queue. When the subscribing application ends, the queue is removed and the subscription is therefore removed.

- ▶ Durable subscriptions

With durable subscriptions, a client registering the subscription can go away and come back later without missing any published messages.

- ▶ Security

Use of a topic can be restricted by setting the appropriate permissions on the associated topic objects using Object Authority Manager (OAM) and Access Control Lists (ACL).

- ▶ Expiration

Messages can be created with an expiration date and time, which means that after that assigned time arrives, the data is no longer available to subscribers.

The WebSphere MQ based publish/subscribe system is a subject-based publish/subscribe system, meaning that a publisher application creates a message and publishes it on the queue manager with a topic string that best fits the subject of the publication. To receive this publication, a subscriber creates a subscription on the queue manager with a pattern that matches the topic string. The queue manager then delivers the publication to the subscriber that matches the publication topic and is authorized to receive the publication.

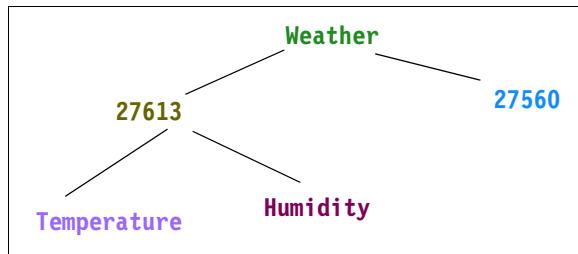
Typically, topics are organized hierarchically, into topic trees, using the '/' character to create subtopics in the topic strings. Wild card characters '+' and '#' can be used in the topic hierarchy when defining topics. Example 9-1 shows a topic tree with one root topic.

Example 9-1 A topic tree with one root topic

Root topic : Weather

Sub topic : 27613

Sub topic : 27560



Valid topics in the topic tree shown Example on page 347 are:

Weather

Weather/27613

Weather/27560

Weather/27613/Temperature

Weather/27613/Humidity

Weather/#

Topics and Topic trees: For details about Topics and Topic trees, refer to the WebSphere MQ Information Center at:

http://www publib boulder ibm com infocenter wmqv7/v7r0/topic/com.ibm.mq.amqnar.doc/ps22040_.htm

9.2 Scenario environment

The scenarios in this chapter were developed in an environment with the following components:

- ▶ WebSphere MQ V7.0.1
- ▶ WebSphere Message Broker run time V7.0
- ▶ WebSphere Message Broker Toolkit V7.0
- ▶ WebSphere Message Broker Toolkit V6.1.0.5
- ▶ IH03 SupportPac for WebSphere Message Broker, which SupportPac provides the rfutil.exe
- ▶ Windows XP Professional Version 5.1 SP2

WebSphere Message Broker: For more information about WebSphere Message Broker SupportPacs, refer to the following Web page:

<http://www-1.ibm.com/support/docview.wss?uid=swg27007197>

9.3 Scenario 1: Comparison and difference

In WebSphere Message Broker and WebSphere MQ V6, the interactions between the publishers and subscribers were all controlled by the message broker. The message broker contained the publish/subscribe engine that received messages from the publishers and subscription requests from the subscribers. The message broker stored the information about the subscribers and their subscription details in the broker database and routed the published data to the target subscribers.

However, WebSphere Message Broker V7 uses a publish/subscribe engine that WebSphere MQ V7.0.1 provides to control the interactions between the publishers and subscribers. It is the publish/subscribe engine in the queue manager that receives the messages from publishers, subscriptions from the subscribers, and eventually routes the published data to the target subscribers.

Although the internals changed, this does not significantly effect the external working of WebSphere Message Broker MQ-based publish/subscribe, which means that if you are already a user of publish/subscribe, you can continue to use your current configuration after upgrading to WebSphere MQ V7.0.1 and Message Broker V7 without making extensive changes to your applications or configuration.

WebSphere MQ V6.x versus WebSphere Message Broker V6.x

Figure 9-3 on page 350 illustrates the differences between publish/subscribe functionalities that WebSphere MQ V6 and WebSphere Message Broker V6 handle.

Publish Subscribe function	MQ V6.x	Message Broker V6.x
Topic based matching	Yes	Yes
Retained publication support	Yes	Yes
RFH (1) Support	Yes	Yes
RFH 2 Support	No	Yes
Security / Access Control (ACLs)	No	Yes
Centralized GUI	No	Yes
High availability option	No	Yes
High performance non-persistent option	No	Yes
Available on all supported platforms	Not supported on z/OS	Yes

Figure 9-3 Comparison chart between PubSub features of MQ V6 and WebSphere Message Broker V6

The comparisons of WebSphere MQ V6 and WebSphere Message Broker V6 are:

- ▶ Support for MQRFH2 headers: WebSphere MQ V6 provided support for the MQRFH1, JMS, AMI, and PCF command based interfaces. MQRFH2 support was added in MQ V7.0.1. MQRFH2 header follows MQ Message descriptor (MQMD) and precedes the message body in a message. MQRFH2 headers are supported in both WebSphere Message Broker V6.1 and V7. WebSphere MQ V7.0.1 supports three formats/routes into the queue manager when using publish/subscribe messaging: RFH1, RFH2, and putting messages directly to the queue manager.
- ▶ Security/ACLs: While WebSphere MQ V6 supported ACL creation and configuration at the queue manager level, MQ V7.0.1 extends it further to publish/subscribe messaging, which tightens the security. In WebSphere MQ, the default is that no user ID has access to any topic unless the ACL explicitly authorizes the access. But in WebSphere Message Broker V6.x, the default is that all user IDs have permission to access any topic unless the ACL explicitly

denies the access. Because of this difference in security approach, the migration process cannot directly migrate the ACLs from Message Broker V6.x to MQ V7.0.1; instead, the migration process produces a security command file that lists the security commands to be executed manually on the queue manager to create the equivalent ACLs. The details can be seen in section 9.4 Migrating from WebSphere Message Broker V6 to Message Broker V7.

- ▶ Centralized graphical configuration: WebSphere MQ V6 did not provide any native interfaces for viewing topics and subscriptions. SupportPac MS0Q extended MQ V6 Explorer to provide a limited topic-based view of the WebSphere MQ publish/subscribe broker. WebSphere Message Broker V6 also provides the Toolkit view that lists, creates, and deletes topics, and lists and deletes subscriptions. WebSphere MQ V7.0.1 Explorer not only provides the ability to browse the defined topics and subscriptions, it also enables the you to test their publications on the fly.
- ▶ High-availability option: Traditionally, WebSphere MQ Publish/Subscribe messaging did not support high-availability (HA). WebSphere Message Broker V6 publish/subscribe messaging had to be used for HA support. However, in MQ V7.0.1, multi-instance queue managers provides HA support by switching to a standby server incase of an active queue manager instance failure.
- ▶ High performance non-persistent option: In WebSphere Message Broker V6, Real time nodes were provided that received messages from Java Message Service (JMS) applications using WebSphere MQ Real-time transport, which is a lightweight protocol that is optimized for use with non-persistent messaging. This protocol is for the applications that relied on the quality of service provided by TCP/IP but do not need persistent delivery. WebSphere MQ V6 relied on the Real time nodes in Message Broker to achieve this. However, in WebSphere Message Broker V7.0, the Real-timeInput node and Real-timeOptimizedFlow nodes are removed. In WebSphere MQ V7.0.1, this functionality is now provided by MQ streaming client technology.
- ▶ Support on z/OS: WebSphere MQ supports publish/subscribe for the first time on z/OS in MQ V7.0.1.

WebSphere MQ V7.x versus WebSphere Message Broker V6.x

Figure 9-4 on page 352 shows the comparison between publish/subscribe functionalities handled by WebSphere Message Broker V7, which now uses WebSphere MQ V7.0.1 for publish/subscribe messaging, and WebSphere Message Broker V6.x.

Publish Subscribe function	MQ V7.x	Message Broker V6
Topic based matching	Yes	Yes
Retained publication support	Yes	Yes
RFH (1) Support	Yes	Yes
RFH 2 Support	Yes	Yes
Security / Access Control (ACLs)	Yes	Yes
Centralized GUI	Yes	Yes
High availability option	Yes	Yes
High performance non-persistent option	Yes	Yes
Available on all supported platforms	Yes	Yes
"Put to topic"	Yes	No

Figure 9-4 Comparison chart between pubsub features of MQ V7 and Message Broker V6

WebSphere MQ V7.0.1 Explorer provides the ability to test the publications and subscriptions on a given topic. WebSphere Message Broker V6 does not provide this capability.

WebSphere Message Broker Version 6.1 provides the ability to define topic trees in the Message Broker Toolkit, but there is no capability to set specific attributes for a particular individual topic in a topic tree. WebSphere MQ Version 7.0.1 supports the concept of topic objects that allow you to set specific, non-default attributes for a topic.

Additionally, native support for non-durable subscriptions was also added in WebSphere MQ V7.0.1, which allows the unconsumed messages and unnecessary subscriptions to be removed at disconnection. This functionality removes the need that existed in WebSphere MQ V6 for JMS to clean up the non-durable subscriptions to meet the JMS specification requirements.

Because WebSphere MQ made significant enhancements to its publish/subscribe engine that covered all of the aspects of publish/subscribe

messaging, it was imperative to move this capability to a single publish/subscribe engine contained in WebSphere MQ. This integration of publish/subscribe capabilities leads to simplification in the workings between WebSphere MQ and WebSphere Message Broker.

This synergy between WebSphere MQ and WebSphere Message Broker reduces the administration overhead for users:

- ▶ Removal of User Name Server

The topic-based security is now handled by WebSphere MQ. Hence the component is no longer needed in WebSphere Message Broker. WebSphere MQ uses the security services that the underlying operating system provides.

- ▶ Removal of message broker database

The subscriber information is not kept in the message broker database any longer but is handled by WebSphere MQ, which, among other reasons, eliminates the need for the broker database.

- ▶ Removal of Configuration Manager

Because the subscription information is not handled in the message broker now, it does not need to exchange the information with the configuration manager any more as in previous versions. WebSphere MQ now handles the subscriber information.

Publication node

Other than the previously mentioned changes in WebSphere Message Broker V7, the Publication node was re-written to interface with the publish/subscribe engine in WebSphere MQ directly rather than the message broker in previous versions, as shown in Figure 9-5.

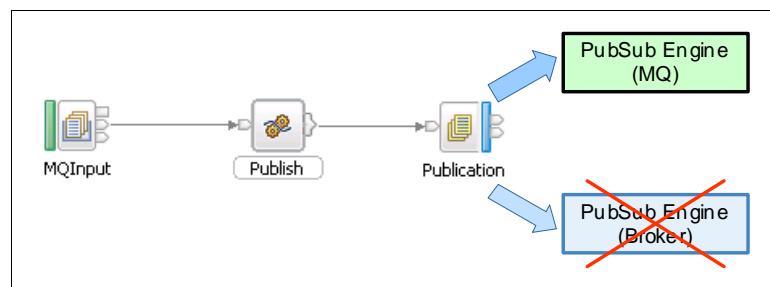


Figure 9-5 Publication node change

The Publication node contains a *NoMatch* terminal explicitly for no publication recipients and an *Out* terminal for further routing and propagation of messages

within or outside of the message flow. The NoMatch terminal is meant for propagating messages that have no matching subscriptions.

Even though all of the subscription matching and routing needs are met with WebSphere MQ, externally the publication node still continues to exhibit the same behavior.

9.3.1 Demonstrating publish/subscribe in WebSphere MQ V7.0.1

In this section, we use a sample that demonstrates the administration of publish/subscribe-related objects using only WebSphere MQ Explorer V7.0.1. You will see the use of MQ Explorer Test publication and test subscriptions features for publishing and subscribing to topic strings. In this example, we create a topic using MQ Explorer, create and register a new subscription against this topic, and a test publication message is put on the topic, which is then delivered to the destination that the subscriber specified:

1. Start the WebSphere MQ Explorer, and in the Navigator pane, click the **Topics** folder. No topic objects are defined yet, as shown in Figure 9-6.

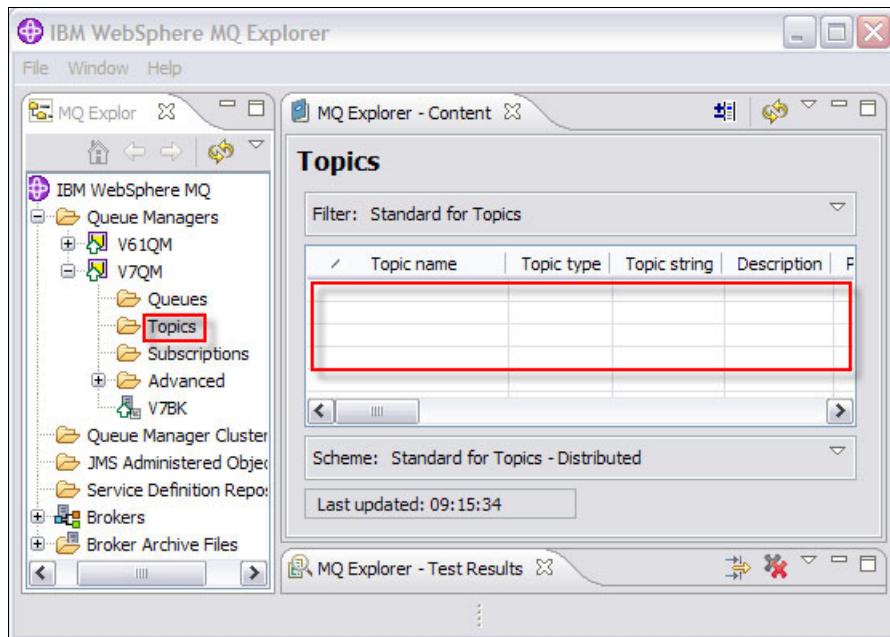


Figure 9-6 WebSphere MQ Explorer Topics view

2. Create a new topic by right-clicking **Topics** → **New** → **Topic**, as shown in Figure 9-7 on page 355.

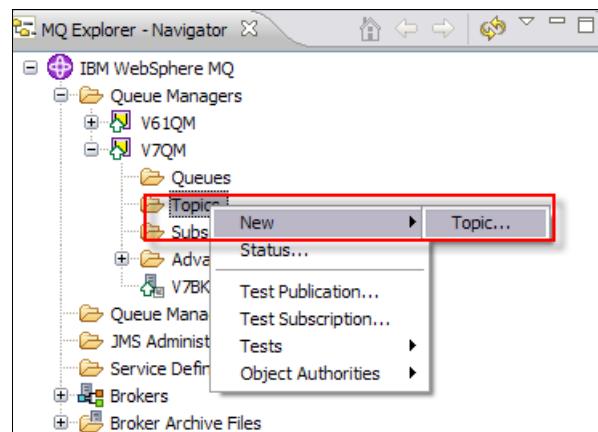


Figure 9-7 Create new Topic

3. In the Name field, type the Topic name Weather.27613, and click **Next**, as shown in Figure 9-8.

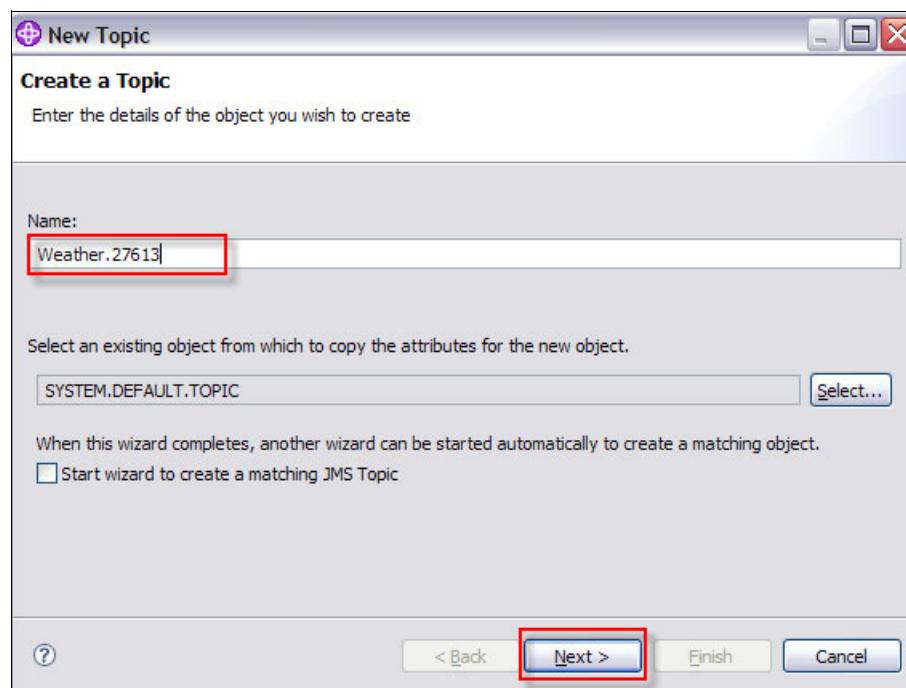


Figure 9-8 Enter Topic name

4. In the Topic string field, type Weather/27613. Accept the default for all of the other property values, and click **Finish**, as shown in Figure 9-9.

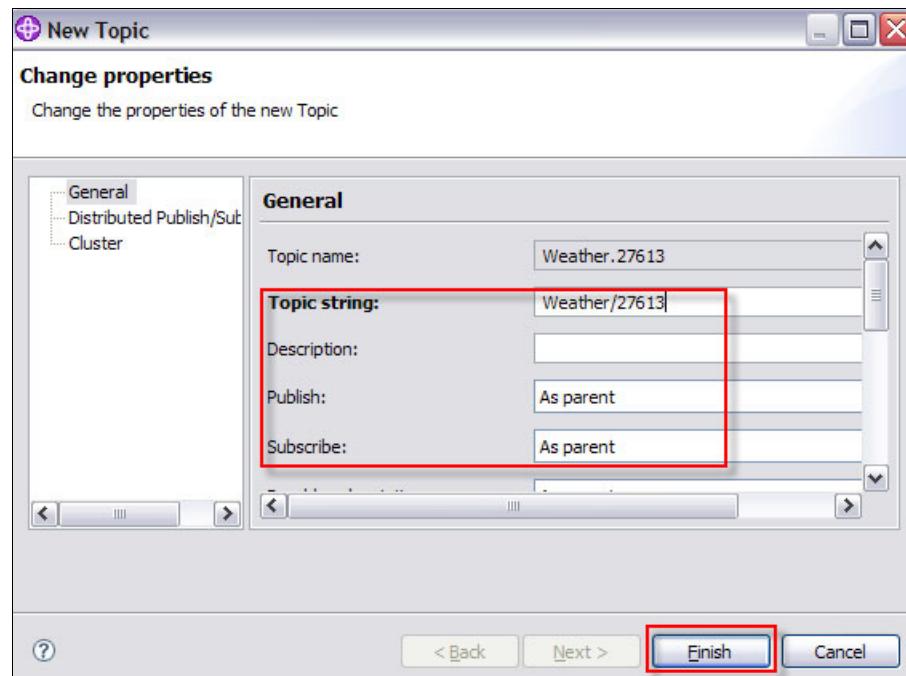


Figure 9-9 Enter Topic details

5. Create a few more topics. Your newly created topics are displayed, as shown in Figure 9-10 on page 357.

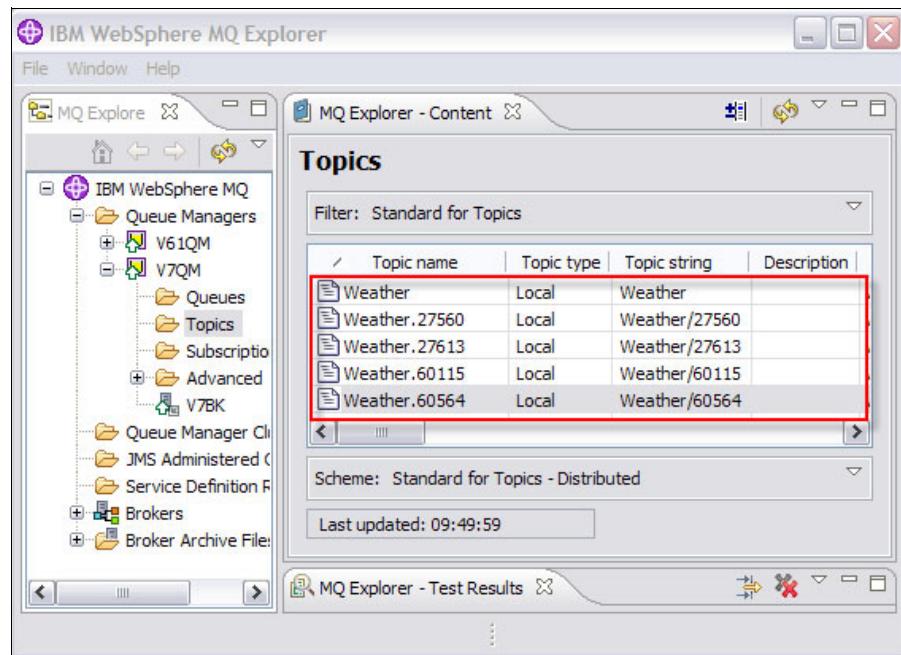


Figure 9-10 Create a few topics

6. Register a new subscription by right-clicking **Subscriptions** → **New** → **Subscriptions**, as shown in Figure 9-11.

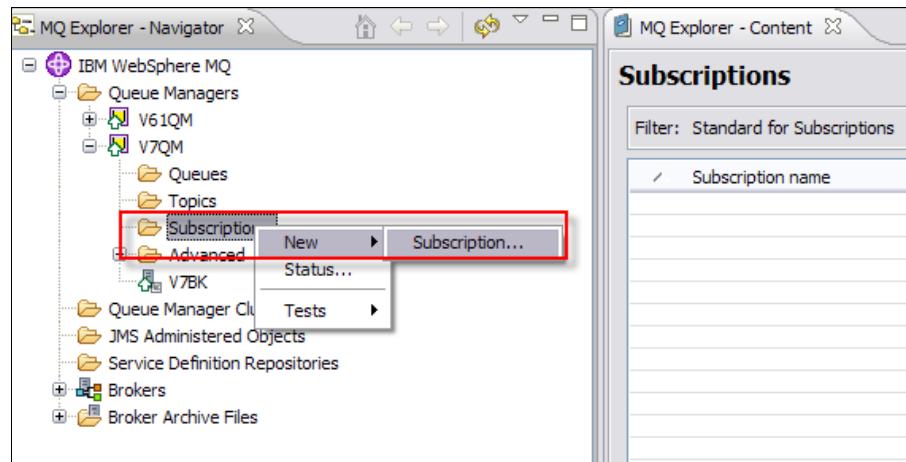


Figure 9-11 Create new subscription

7. In the Name field, provide the subscription name Weather_SUB, and click **Next** as shown in Figure 9-12.

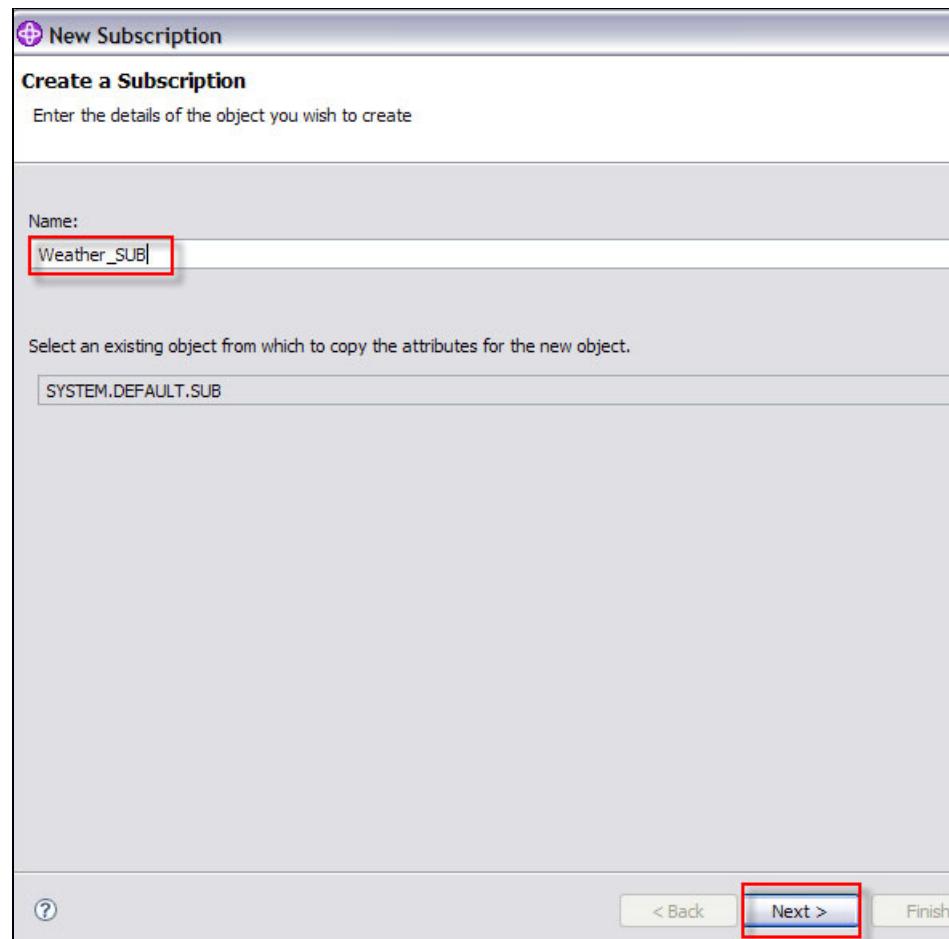


Figure 9-12 Enter Subscription name

8. Click **Select** to browse to the created Topic names, as shown in Figure 9-13 on page 359.

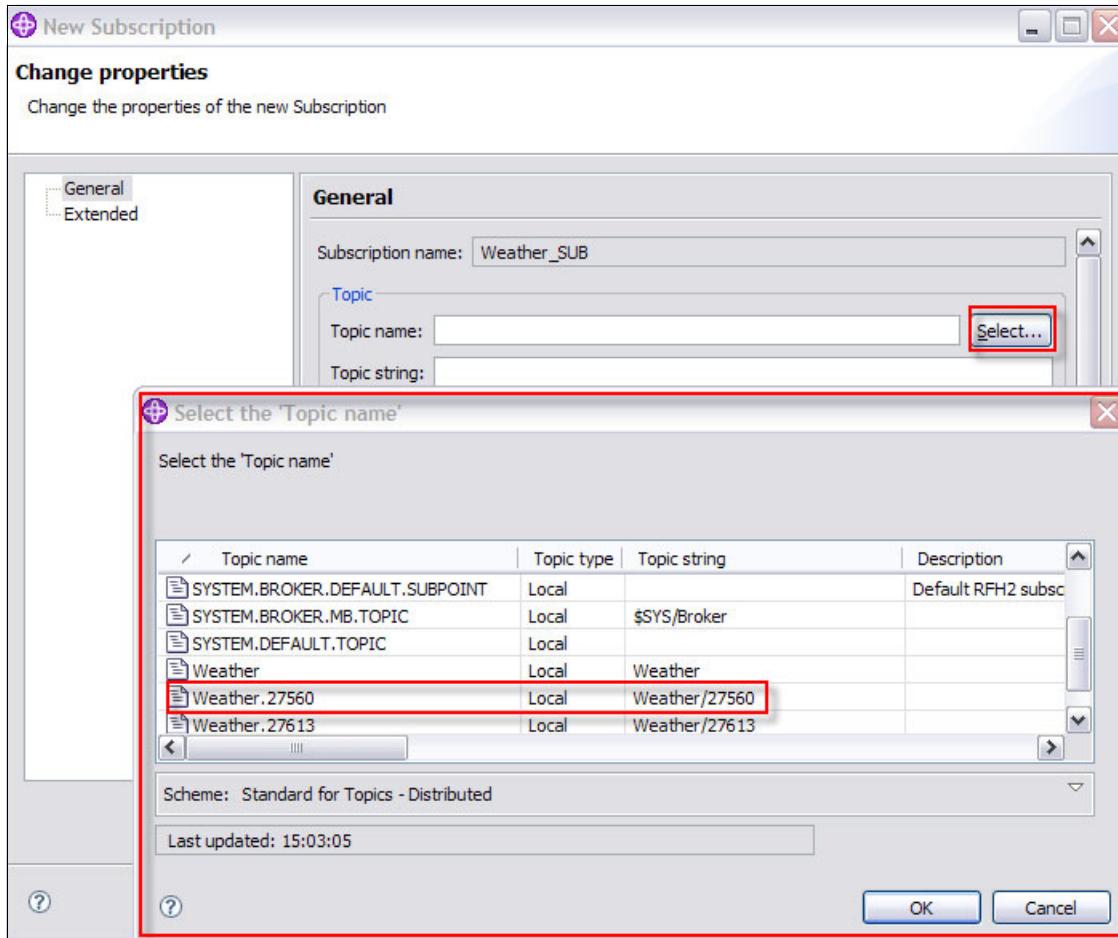


Figure 9-13 Select the Topic name

9. Select the Topic name Weather.27560, and provide the Destination queue manager V7QM and Destination queue SUB_Q, and click **Finish**, as shown in Figure 9-14 on page 360.

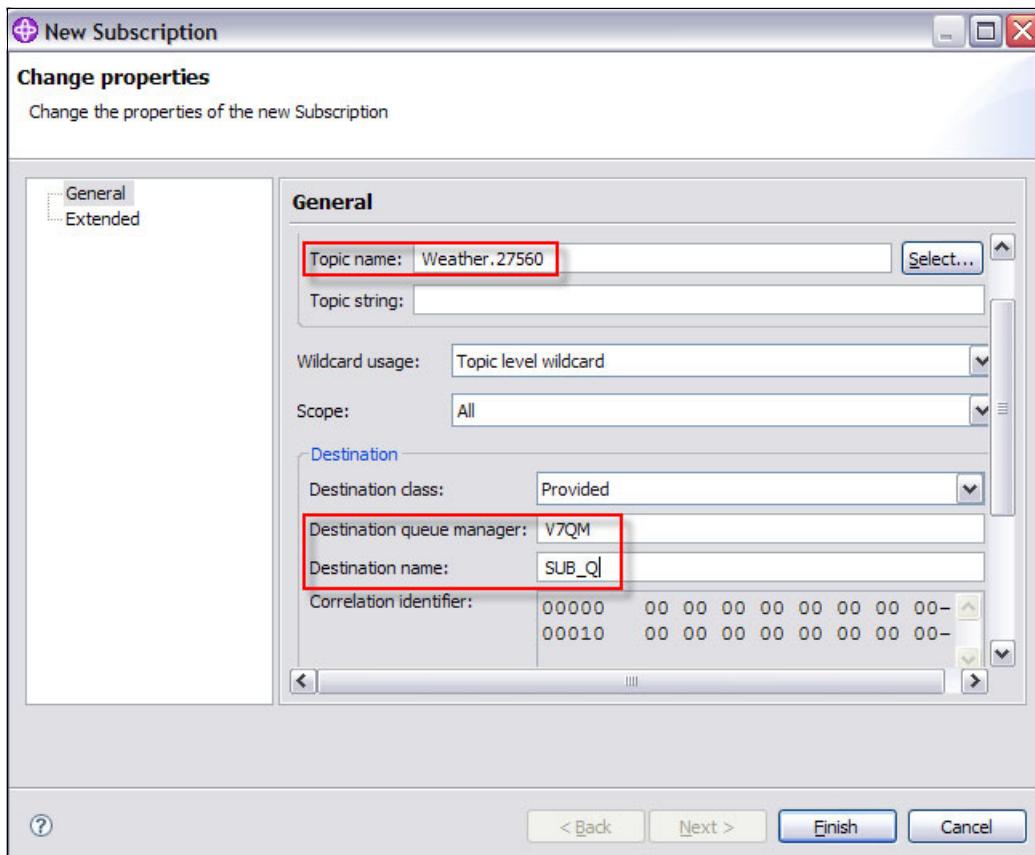


Figure 9-14 Enter subscription details

10. The Subscriptions pane now shows the created subscription Weather_SUB, as shown in Figure 9-15 on page 361.

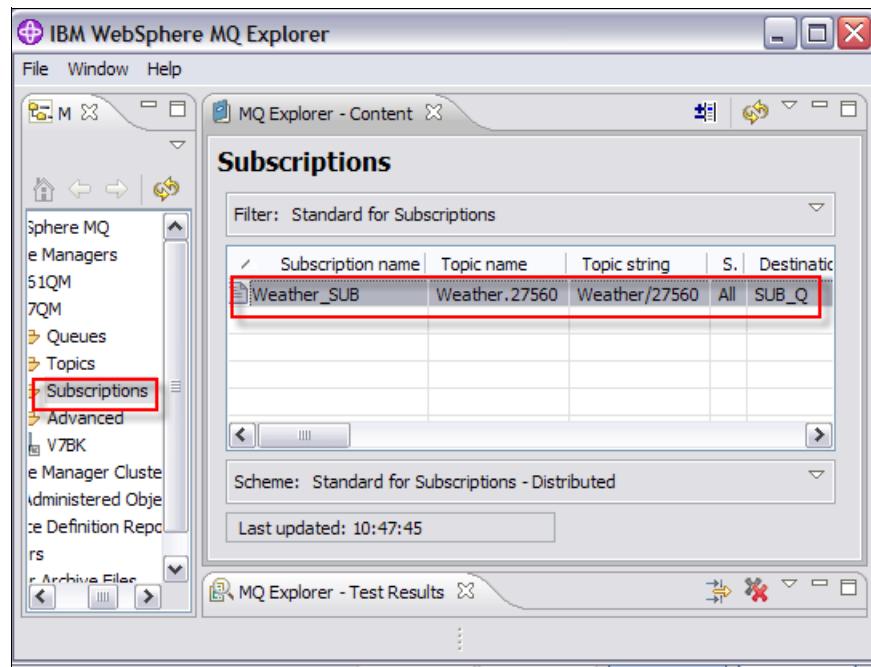
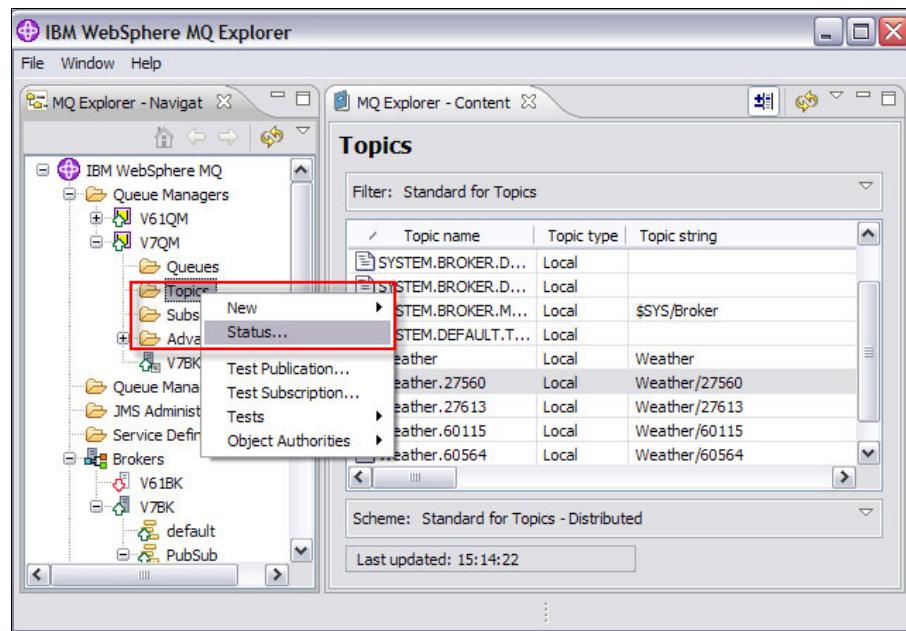


Figure 9-15 View the created Subscription

11.Right-click **Topics**, and select **Status**, as shown in Figure 9-16 on page 362.



Topic name	Topic type	Topic string
SYSTEM.BROKER.D...	Local	
SYSTEM.BROKER.D...	Local	\$SYS/Broker
STEM.BROKER.M...	Local	
STEM.DEFAULT.T...	Local	
weather	Local	Weather
weather.27560	Local	Weather/27560
weather.27613	Local	Weather/27613
weather.60115	Local	Weather/60115
weather.60564	Local	Weather/60564

Figure 9-16 Topic status

12. Because the subscription Weather_SUB is registered on the topic Weather.27560, the Sub count increments; however, the topics against which the subscriptions are not yet registered still list the Sub count as 0, as shown in Figure 9-17 on page 363.

V7QM - Topic Status

Queue Manager: V7QM

Topic status:

Topic string	Publish	Subscr...	Durabl...	Model non-durable queue	Admin topic n...	Sub count	Pub count
[Empty]	Allowed	Allowed	Allowed	SYSTEM.NDURABLE.MO...	SYSTEM.BASE....	0	0
\$SYS	Allowed	Allowed	Allowed	SYSTEM.NDURABLE.MO...		0	0
Weather	Allowed	Allowed	Allowed	SYSTEM.NDURABLE.MO...	Weather	0	0
+ 27560	Allowed	Allowed	Allowed	SYSTEM.NDURABLE.MO...	Weather.27560	1	0
+ 27613	Allowed	Allowed	Allowed	SYSTEM.NDURABLE.MO...	Weather.27613	0	0
+ 60115	Allowed	Allowed	Allowed	SYSTEM.NDURABLE.MO...	Weather.60115	0	0
+ 60564	Allowed	Allowed	Allowed	SYSTEM.NDURABLE.MO...	Weather.60564	0	0
+ NC	Allowed	Allowed	Allowed	SYSTEM.NDURABLE.MO...		0	0

Scheme: Standard for Topic Status - Distributed

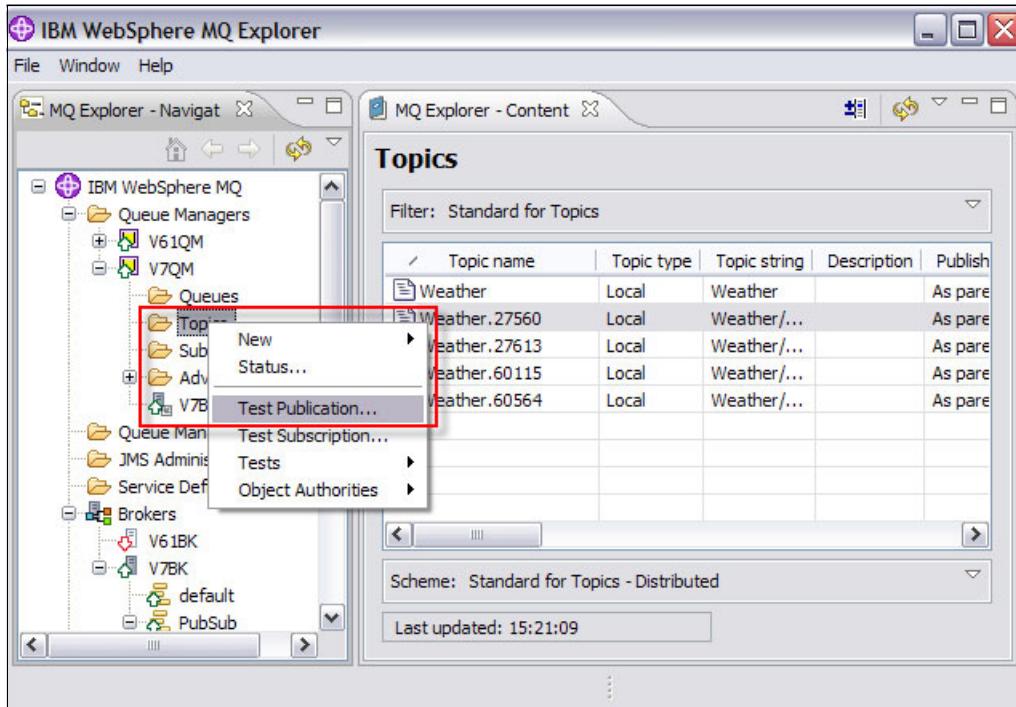
Table last refreshed: 10:52:20 Selected item last updated: 10:52:27

?

Refresh Refresh All Close

Figure 9-17 Topic view showing subscription status

13. To publish a message in MQ Explorer, right-click **Topics**, and select **Test Publication**, as shown in Figure 9-18 on page 364.



Topic name	Topic type	Topic string	Description	Publish
Weather	Local	Weather		As pare
Weather.27560	Local	Weather/...		As pare
Weather.27613	Local	Weather/...		As pare
Weather.60115	Local	Weather/...		As pare
Weather.60564	Local	Weather/...		As pare

Figure 9-18 Test Publication

14. Provide the Topic string Weather/27560 against which the subscription was registered, and also fill in the Message data with some sample text, as shown in Example 9-2.

Example 9-2 Message data sample text

This is test publication. The temperature is 55 degrees.

15. Click Publish message, as shown in Figure 9-19 on page 365.

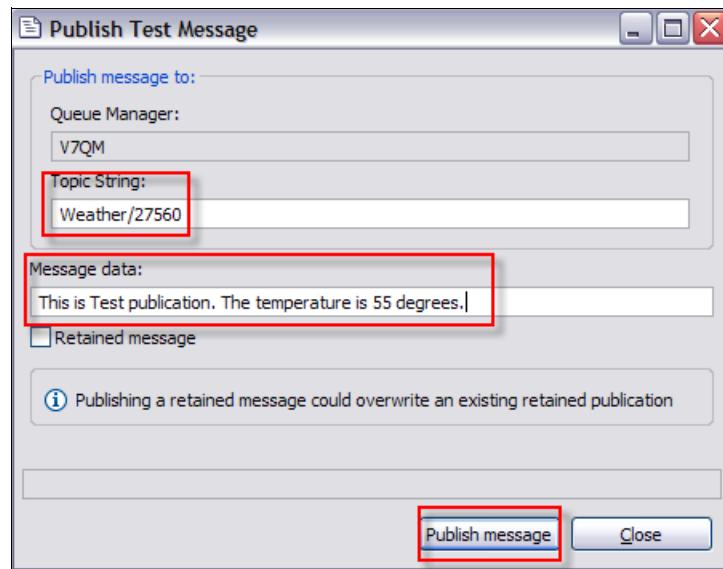


Figure 9-19 Publish test message

16. The published message went to the subscriber queue SUB_Q, as shown in Figure 9-20 on page 366.

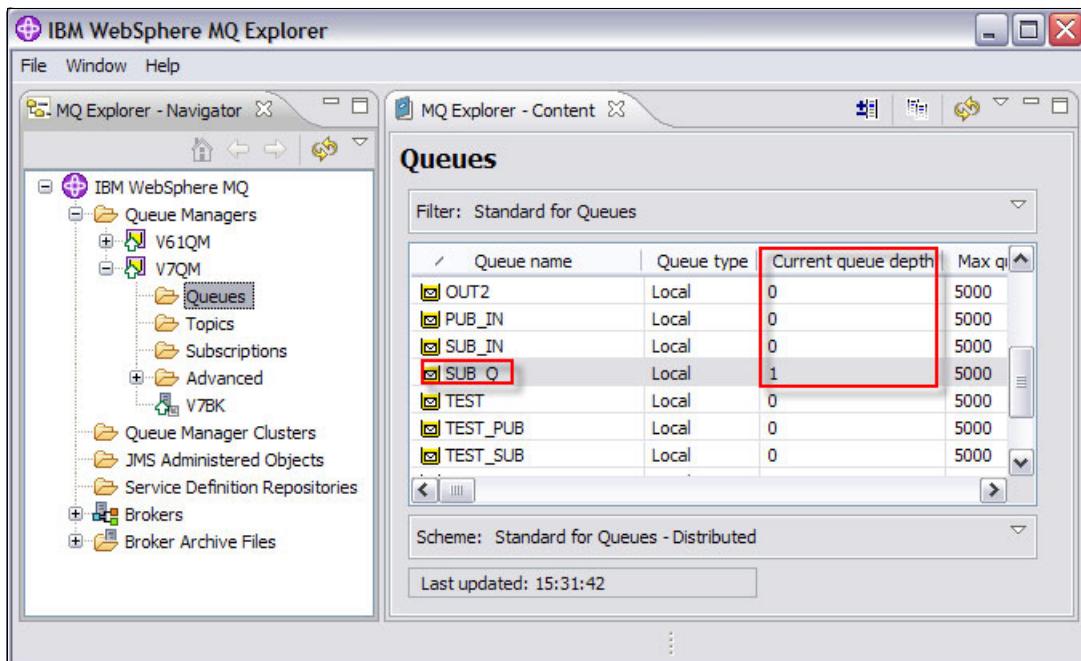


Figure 9-20 Current queue depth of subscriber queue

17. The message browser displays the published message text This is test publication. The temperature is 55 degrees., as shown in Figure 9-21.

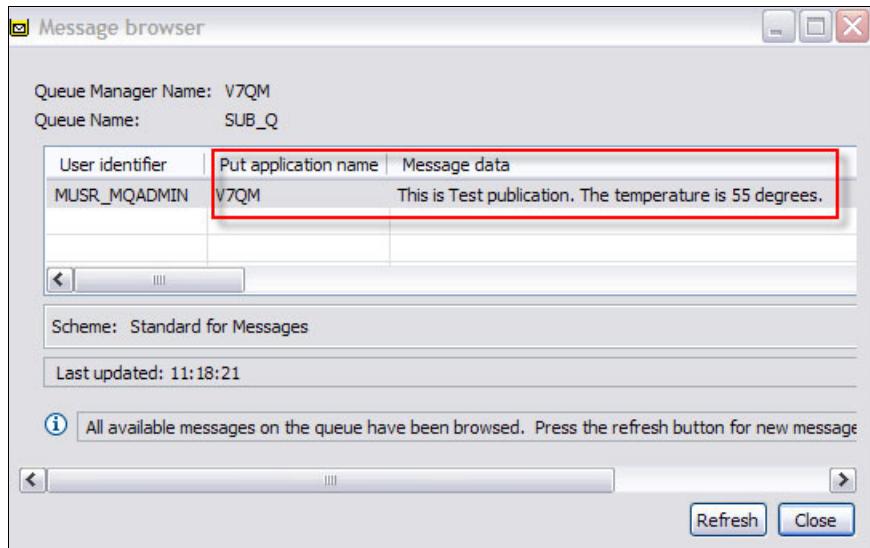


Figure 9-21 Message browser

The publish/subscribe messaging can also be tested from the command line in WebSphere MQ.

Publish/subscribe: For further details about publish/subscribe, refer to the WebSphere MQ Information Center at:

http://www.publib.boulder.ibm.com/infocenter/wmqv7/v7r0/topic/com.ibm.mq.amqnar.doc/ps19080_.htm

9.3.2 Demonstrating publish/subscribe in WebSphere Message Broker V6

In the following sample, we demonstrate the basic publish/subscribe messaging using WebSphere Message Broker V6. The sample registers a subscription on a topic using an application *rfhutil.exe*. The publish message flow then publishes a message on the topic for which the subscription is registered. See Appendix A, “Additional material” on page 533, for a sample Project Interchange V61_flows.zip containing the message flow V61_publish.msgflow:

1. An application registers a subscription on the broker on a topic *Weather/#* using *rfhutil.exe*, as shown in Figure 9-22 on page 368. The subscription queue manager *V61QM* and subscription queue *OUT* are provided.

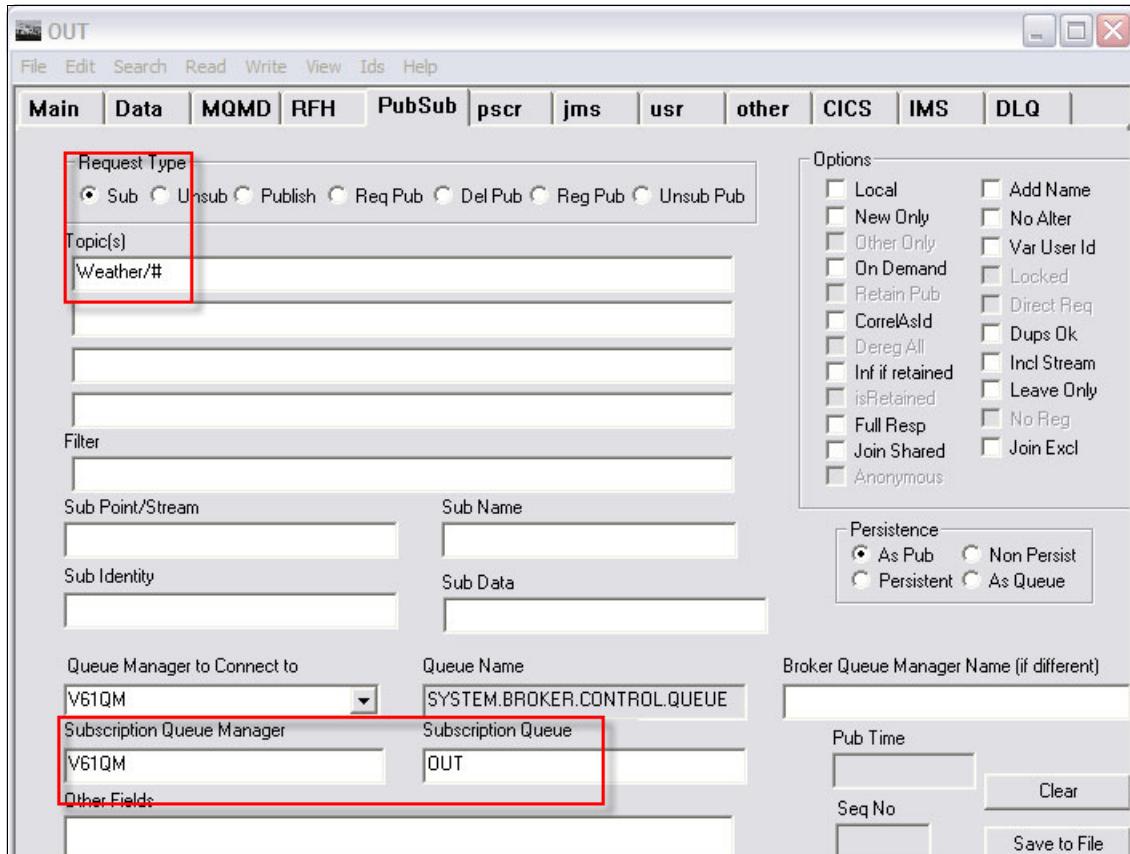


Figure 9-22 Application registering subscription

2. In the toolkit, select the **Subscriptions** tab. Click **Query**, and ensure that the registered subscription Weather/# shows in the pane, as shown in Figure 9-23 on page 369.

Subscriptions

Topics	All Topics	Brokers	All Brokers																															
Users	All Users	Subscription Points	All Subscription Points																															
Registration Date between Any Date and Any Date (The date format is: December 10, 2009)				<input type="button" value="Query"/>																														
<table border="1" style="width: 100%; border-collapse: collapse; font-size: small;"> <thead> <tr> <th style="width: 20%;">Topic</th> <th style="width: 20%;">Broker</th> <th style="width: 20%;">Registration D...</th> <th style="width: 20%;">Client</th> <th style="width: 20%;">Content Filter</th> </tr> </thead> <tbody> <tr><td>\$SYS/Broker/+/warning/expi...</td><td>V61BK</td><td>May 16, 2009</td><td>mqrffh2:V61QM:SYST...</td><td></td></tr> <tr><td>\$SYS/Broker/+/Subscription/#</td><td>V61BK</td><td>May 16, 2009</td><td>mqrffh2:V61QM:SYST...</td><td></td></tr> <tr><td>\$SYS/Broker/+/Status</td><td>V61BK</td><td>May 16, 2009</td><td>mqrffh2:V61QM:SYST...</td><td></td></tr> <tr><td>\$SYS/Broker/+/Status/Exec...</td><td>V61BK</td><td>May 16, 2009</td><td>mqrffh2:V61QM:SYST...</td><td></td></tr> <tr style="background-color: #cccccc;"><td>Weather/#</td><td>V61BK</td><td>December 7, 2...</td><td>mqrffh2:V61QM:OUT</td><td></td></tr> </tbody> </table>					Topic	Broker	Registration D...	Client	Content Filter	\$SYS/Broker/+/warning/expi...	V61BK	May 16, 2009	mqrffh2:V61QM:SYST...		\$SYS/Broker/+/Subscription/#	V61BK	May 16, 2009	mqrffh2:V61QM:SYST...		\$SYS/Broker/+/Status	V61BK	May 16, 2009	mqrffh2:V61QM:SYST...		\$SYS/Broker/+/Status/Exec...	V61BK	May 16, 2009	mqrffh2:V61QM:SYST...		Weather/#	V61BK	December 7, 2...	mqrffh2:V61QM:OUT	
Topic	Broker	Registration D...	Client	Content Filter																														
\$SYS/Broker/+/warning/expi...	V61BK	May 16, 2009	mqrffh2:V61QM:SYST...																															
\$SYS/Broker/+/Subscription/#	V61BK	May 16, 2009	mqrffh2:V61QM:SYST...																															
\$SYS/Broker/+/Status	V61BK	May 16, 2009	mqrffh2:V61QM:SYST...																															
\$SYS/Broker/+/Status/Exec...	V61BK	May 16, 2009	mqrffh2:V61QM:SYST...																															
Weather/#	V61BK	December 7, 2...	mqrffh2:V61QM:OUT																															

Figure 9-23 Toolkit subscriptions view

3. Create a message flow V61_publish.msgflow that publishes the message on the broker, as shown in Figure 9-24.

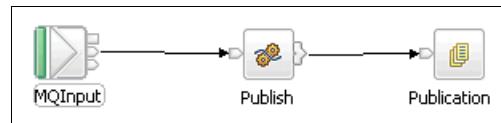


Figure 9-24 Publication Message flow

4. The compute node Publish is coded with ESQL V61_publish.esql, as shown in Figure 9-25 on page 370.

```

CREATE COMPUTE MODULE V61_publish_Compute
  CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
      | CALL CopyMessageHeaders();
      -- CALL CopyEntireMessage();
      -- Set up the Publication
      SET OutputRoot.MQRFH2.psc.Command = 'Publish';
      SET OutputRoot.MQRFH2.psc.PubOpt = 'Local';
      SET OutputRoot.MQRFH2.psc.Topic = InputRoot.XMLNSC.Request.Topic;
      SET OutputRoot.XMLNSC.Text = InputRoot.XMLNSC.Request.Text;

      RETURN TRUE;
    END;

    CREATE PROCEDURE CopyMessageHeaders() BEGIN
      DECLARE I INTEGER 1;
      DECLARE J INTEGER;
      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;

```

Figure 9-25 Compute node ESQL

5. The compute node Publish is coded with ESQL V61_publish.esql, as shown in Example 9-3.

Example 9-3 ESQL V61_publish.esql

```

CREATE COMPUTE MODULE V61_publish_Compute
  CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
      CALL CopyMessageHeaders();
      -- CALL CopyEntireMessage();
      --Set up the Publication
      SET OutputRoot.MQRFH2.psc.Command = 'Publish';
      SET OutputRoot.MQRFH2.psc.PubOpt = 'Local';
      SET OutputRoot.MQRFH2.psc.Topic = InputRoot.XMLNSC.Request.Topic;
      SET OutputRoot.XMLNSC.Text = InputRoot.XMLNSC.Request.Text;
      RETURN TRUE;
    END;

    CREATE PROCEDURE CopyMessageHeaders() BEGIN
      DECLARE I INTEGER 1;

```

```

DECLARE J INTEGER;
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 MQInput node MQInput in the V61_publish.msgflow is configured for the queue name PUB_IN and the XMLNSC domain, as shown in Figure 9-26. All other parameters are set to default.

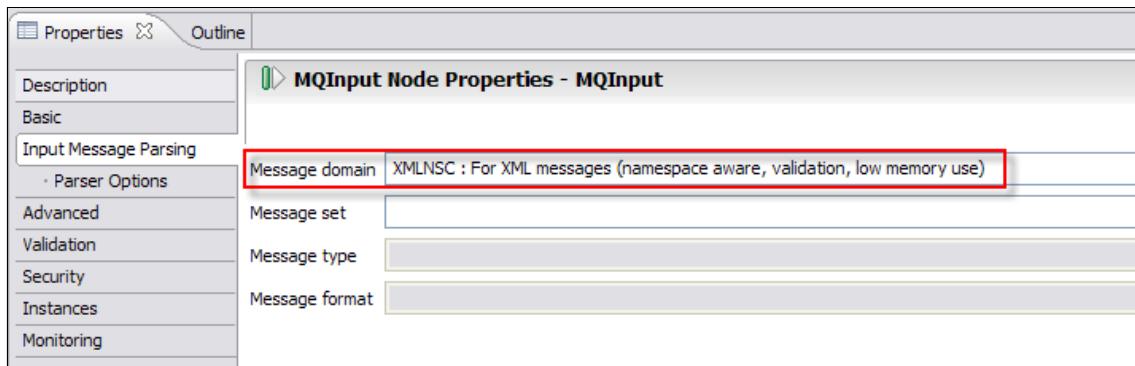


Figure 9-26 MQInput Node Properties

- From the Toolkit, deploy the message flow V61_publish.msgflow to the execution group, as shown in Figure 9-27.

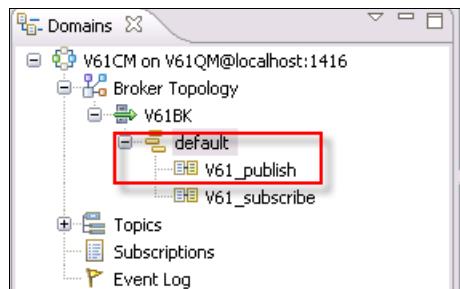


Figure 9-27 Message flow deploy

7. Put the Input message, as shown in Example 9-4, on the input queue PUB_IN of the message flow.

Example 9-4 Input message

```
<?xml version="1.0" encoding="utf-8"
?><Request><Topic>Weather</Topic><Text>This is Test Weather sample
for V6.1</Text></Request>
```

The subscription queue OUT received the published message based on the registered subscription, as shown in Figure 9-28.

Queues		
Filter: Standard for Queues		
/	Queue name	Open input count
	OUT	0
	PUB IN	1

Figure 9-28 Current depth on subscriber queue

When this message flow project V61_publish.msgflow is migrated to WebSphere Message Broker V7, it will still work in the same way. The complete migration details and scenario are demonstrated in 9.4, “Scenario 2: Migration” on page 379 of this chapter.

9.3.3 Demonstrating publish/subscribe in WebSphere MQ V7.0.1 using WebSphere Message Broker V7

The same example can be demonstrated in WebSphere MQ V7.0.1, in the following way:

1. A sample application rfutil.exe registers a subscription on the broker. The subscription queue manager V7QM and subscription queue OUT are defined, as shown in Figure 9-29 on page 373.

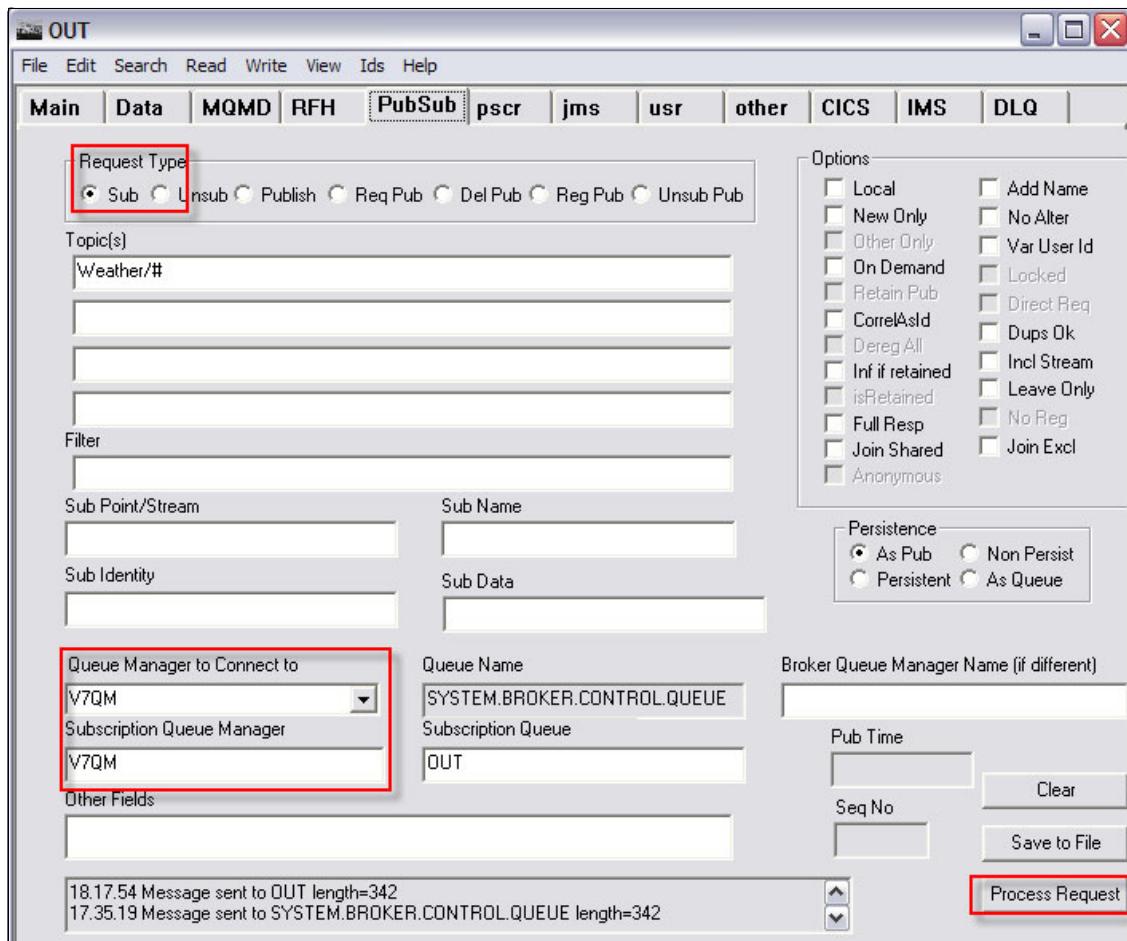


Figure 9-29 Subscribing application

2. In the MQ Explorer, select the **Subscriptions** tab, as shown in Figure 9-30 on page 374.

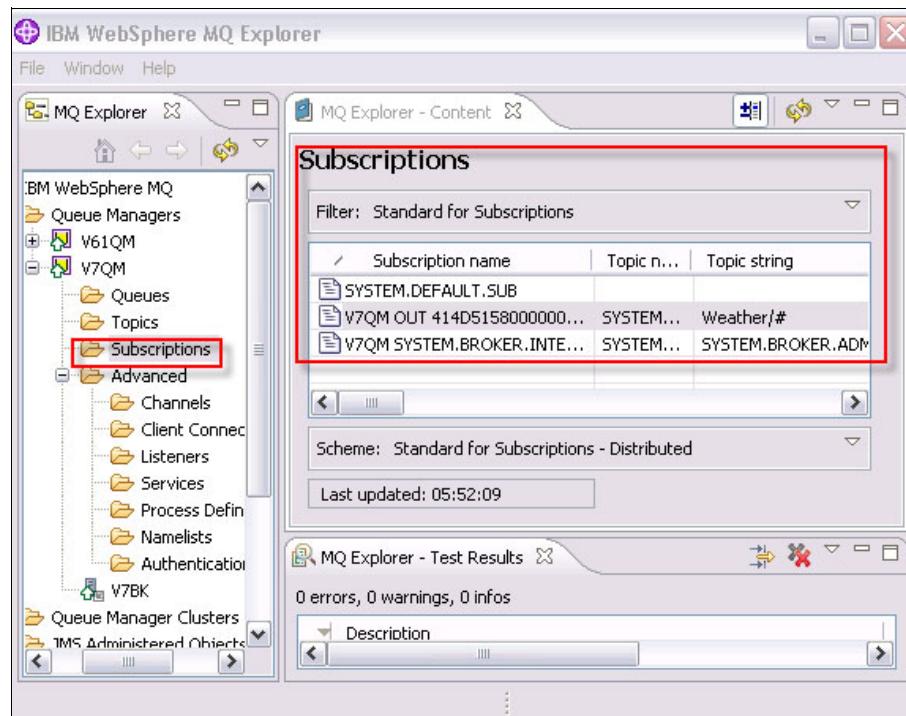


Figure 9-30 Subscriptions view in MQ Explorer

3. The publication message flow V61_publish.msgflow can simply be exported in the Message Broker V6.1 Toolkit from **File** → **Export**. The project can be exported as a Project Interchange V61_pubsub, as shown in Figure 9-31 on page 375.



Figure 9-31 Export as Project Interchange

4. Import this Project Interchange V61_pubsub in WebSphere Message Broker V7 Toolkit from **File** → **Import** → **Other** → **Project Interchange**, and browse to the project V61_pubsub, as shown in Figure 9-32 on page 376.

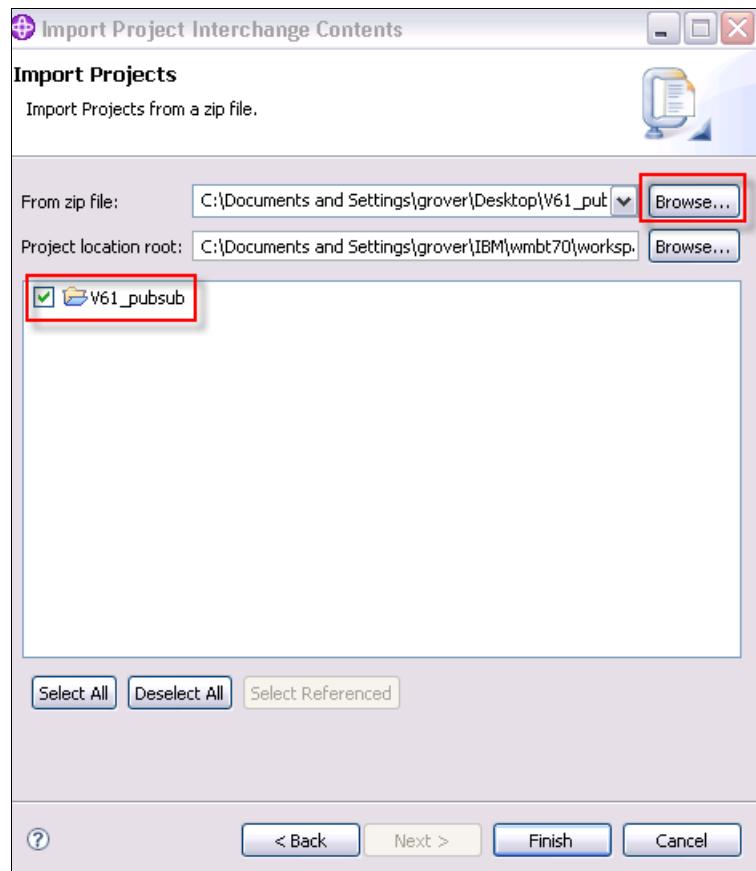


Figure 9-32 Import as Project Interchange

5. Deploy this message flow to the Execution Group default in WebSphere Message Broker V7 broker V7BK. The deploy can be performed from the Toolkit, as shown in Figure 9-33 on page 377, Message Broker Explorer, or using `mqsideploy` command.

The `mqsideploy` command: For more details about the `mqsideploy` command, refer to:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp?topic=/com.ibm.etools.mft.doc/an09020_.htm

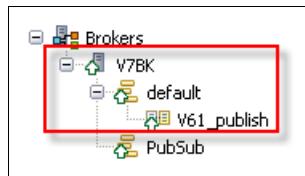


Figure 9-33 Deployed message flow

6. As shown in Example 9-5, on the input queue PUB_IN of the publication message flow V61_publish using rfhutil.exe, publish a text message, as shown in Figure 9-34 on page 378.

Example 9-5 Text message syntax

```
<?xml version="1.0" encoding="utf-8"  
?><Request><Topic>Weather</Topic><Text>This is Test Weather sample  
for Version 7</Text></Request>
```

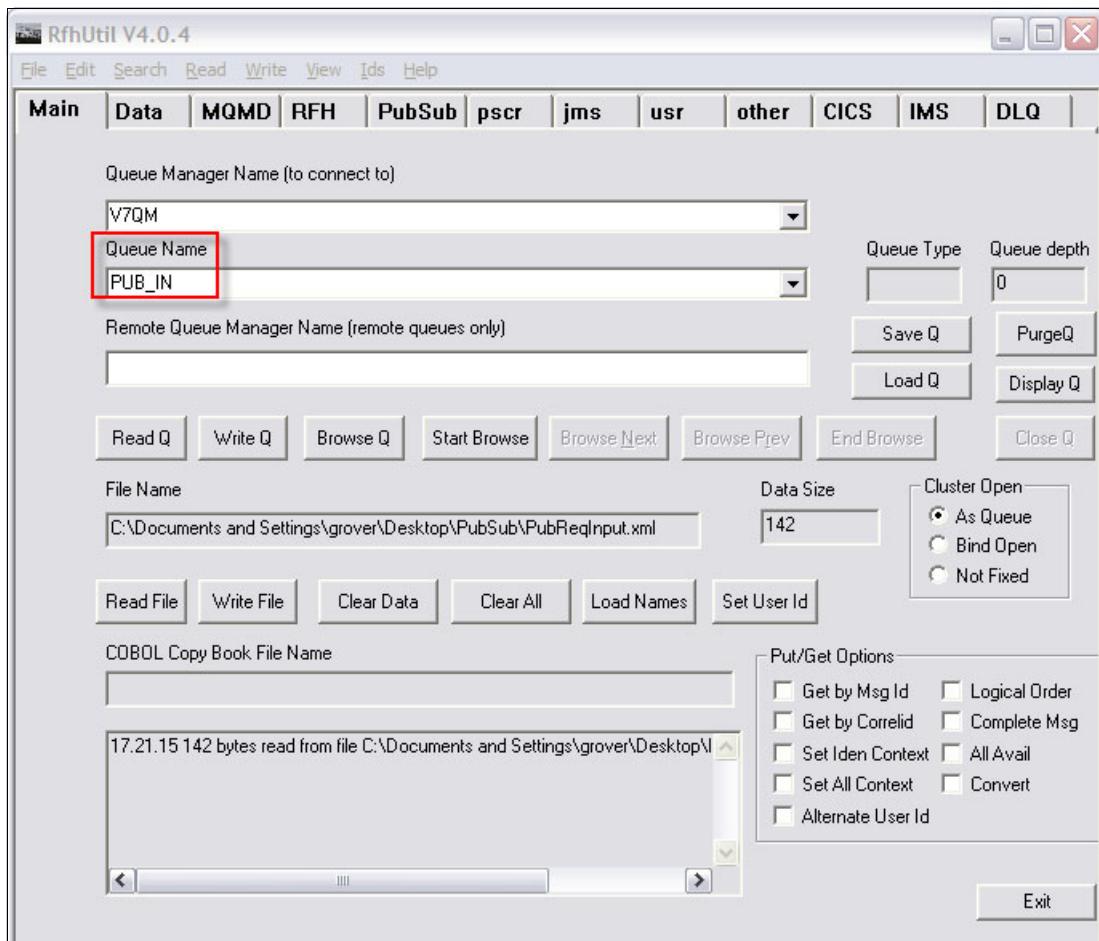


Figure 9-34 The rfhutil.exe publishing the message

7. This time the subscription queue OUT of queue manager V7QM received the published message based on the registered subscription.

As demonstrated, in WebSphere Message Broker V6.1, the messaging is carried out by the broker, whereas in Message Broker V7, the messaging is completely handled by WebSphere MQ queue manager using the queues instead of the broker database. However, it does not affect the existing Message Broker V6 users migrating to Message Broker V7 because the external behavior is still the same.

9.4 Scenario 2: Migration

WebSphere MQ supports migration of publish/subscribe configuration data from the following products:

- ▶ WebSphere Event Broker V6
- ▶ WebSphere Message Broker V6

Because the publish/subscribe messaging is now handled by WebSphere MQ, the publish/subscribe migration involves:

- ▶ Migration of subscriptions, subscription points, streams, retained publications
- ▶ Migration of Access Control Lists (ACLs)

This information exists in the broker database in Message broker V6.x. It is migrated to the SYSTEM as temporary objects in the MQ queue manager. The supported publish/subscribe migration paths include migration from WebSphere Message Broker V6.0 and V6.1 to V7. Migration is backward compatible, which means that you can migrate back from WebSphere Message Broker V7 to V6.x. Migration of publish/subscribe configuration must be performed with complete broker migration. You can migrate the publish/subscribe configuration without migrating the whole message broker, but the broker might not be usable unless it is migrated too.

This task upgrades WebSphere Message Broker Version 6.0 or 6.1 to WebSphere Message Broker Version 7.0 in your enterprise.

The migration is typically performed in three phases:

- ▶ Rehearsal phase: This phase creates a migration log that reports any errors that might be encountered, but does not modify any configurations. It is typically used to observe the potential results of the real migration.
- ▶ Initial phase: This phase creates the topic objects in the queue manager, based on the Access Control List entries that are defined in the message broker. It also produces a file containing the security commands to set up the security environment on the queue manager that is equivalent to the one that existed in the broker. These security commands must be run separately on the queue manager. This phase must be run prior to running the completion phase, and the security command file must be reviewed and changed, if needed. The migration process first checks whether the topic object (regardless of its properties) to be migrated already exists in the queue manager, and if it does, it does not create a new topic object, which prevents creating multiple Topic objects if the migration process is run multiple times.
- ▶ Completion phase: This phase retrieves the existing publish/subscribe definitions from the message broker and uses them to create equivalent definitions in the publish/subscribe component of the queue manager. When

the migration is complete, the queue manager publish/subscribe configuration is equivalent to the earlier message broker publish/subscribe configuration.

Note: The WebSphere Message Broker V6 publish/subscribe information, which is stored in the broker database tables, is not deleted by the migration process.

Migration prerequisites

The migration prerequisites are to ensure that:

- ▶ The WebSphere MQ Version 7.0.1 is installed.
- ▶ If running with an older version of WebSphere MQ, WebSphere MQ is migrated to V7.0.1 prior to migrating the Message Broker.
- ▶ On distributed platforms, the command environment is initialized such that WebSphere MQ and WebSphere Message Broker commands can be run from the command line.
- ▶ WebSphere MQ Version 7.0.1 queue manager is not currently handling any publish or subscribe messages.
- ▶ The queue manager attribute PSMODE is set to COMPAT. If not, change it to COMPAT, which you can do in MQ Explorer by right-clicking the **queue manager** → **Properties** → **Publish/Subscribe** → **Publish/Subscribe mode** or by using the **runmqsc** command: ALTER QMGR PSMODE(COMPAT).

The migration steps are:

1. Uninstall WebSphere MQ Version 6.x, and install WebSphere MQ Version 7.0.1 on the system or Upgrade WebSphere MQ V6.x to WebSphere MQ V7.0.1.
2. Install WebSphere Message Broker Version 7.0 on your system.
3. Run the migration process (**migmbbrk**) with the **-r** parameter, as demonstrated in 9.4.1, “Migrating a publish/subscribe from WebSphere Message Broker V6.1 to WebSphere Message Broker V7.0” on page 381. This option rehearses the migration of the publish/subscribe configuration data from the broker to its underlying queue manager without changing either of the configurations.
4. Review the contents of the log file to check what is to be migrated in the actual migration.
5. For ACL migration, copy the file that contains the security commands. Review and edit the commands as needed in your copy of the security commands file to ensure that they create a security environment that is like your broker security environment.

6. Run the migration process (`migmbbrk`) with the `-t` parameter to initiate migration, as shown in 9.4.1, “Migrating a publish/subscribe from WebSphere Message Broker V6.1 to WebSphere Message Broker V7.0” on page 381. The migration process migrates the publish/subscribe configuration data to the queue manager, creates a log file and a new security commands file, and shuts down the message broker.
7. Check the contents of the new security commands file against your backup copy of the file that you obtained from the rehearsal phase to make sure that nothing ACL-related changed because you rehearsed the migration. If anything has changed, you might need to edit your copy of the security commands file.
8. Run the migration process with the `-c` parameter to migrate the subscriptions and complete the migration of publish/subscribe components.
9. Run `mqsimigratecomponents` to migrate the Message broker V6.x to V7.
10. Change the queue manager `PSMODE` attribute to **Enabled** from WebSphere MQ Explorer or by using the `runmqsc` command: `ALTER QMGR PSMODE(ENABLED)`.
11. Restart the message broker.

When migrating a broker collective from WebSphere Message Broker V6 to WebSphere MQ V7.0.1 or later, publish/subscribe clusters must be created on each of the queue managers that are associated with the brokers. After that, publish/subscribe information in each of the brokers in the collectives can be migrated to queue managers in the same way as the individual brokers. However, all of the brokers in the collective should be migrated at the same time. The subscriptions are required to be resynchronized with all of the queue managers in the publish/subscribe cluster.

Migration on clusters: For details about migration on clusters, see the WebSphere MQ Information Center at:

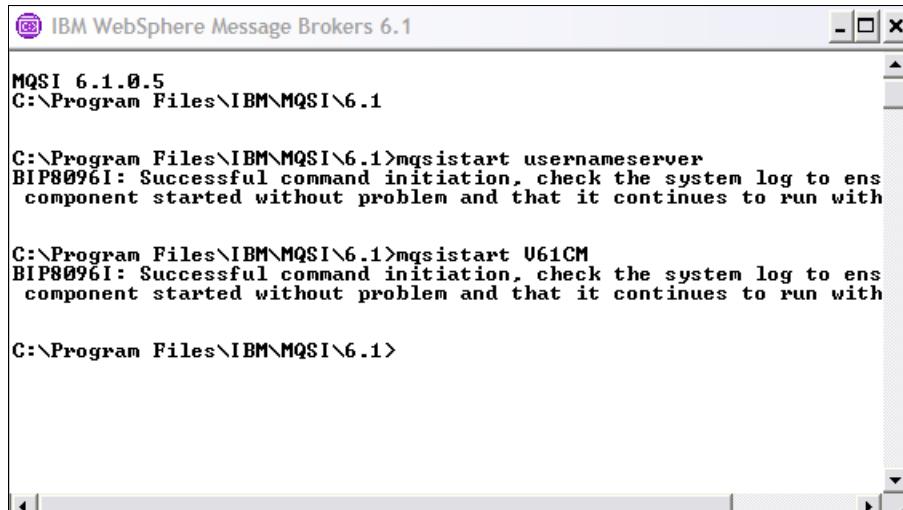
http://www publib boulder ibm com infocenter wmq7/v7r0/topic/com.ibm.mq.amqnar.doc/ps29100_.htm

9.4.1 Migrating a publish/subscribe from WebSphere Message Broker V6.1 to WebSphere Message Broker V7.0

In this scenario, we demonstrate the migration of publish/subscribe configuration (topics, subscriptions, ACLs, and flows) from WebSphere Message Broker V6.x to V7. In WebSphere Message Broker V7, the publish/subscribe component is

replaced by WebSphere MQ 7 Publish/Subscribe engine, which, as we saw previously, internally functions in a different way:

1. Start the WebSphere Message Broker V6.1 Usernameserver using the command `mqsistart usernameserver`, as shown in Figure 9-35.
2. Start the WebSphere Message Broker V6.1 configuration manager using the command `mqsistart V61CM`, as shown in Figure 9-35.



The screenshot shows a command-line interface window titled "IBM WebSphere Message Brokers 6.1". The title bar includes standard window controls (minimize, maximize, close) and a maximize/minimize button. The main area displays two sets of command outputs:

```
MQSI 6.1.0.5
C:\Program Files\IBM\MQSI\6.1>mqsistart usernameserver
BIP8096I: Successful command initiation, check the system log to ens
component started without problem and that it continues to run with

C:\Program Files\IBM\MQSI\6.1>mqsistart V61CM
BIP8096I: Successful command initiation, check the system log to ens
component started without problem and that it continues to run with

C:\Program Files\IBM\MQSI\6.1>
```

Figure 9-35 V6.1 Command console

3. Go to the Broker Administration perspective in Message Broker Toolkit B6.1, and connect to the configuration manager.
4. Double-click Topics to open the Topics view, which contains the list of topics and topics access control lists, as shown in Figure 9-36.

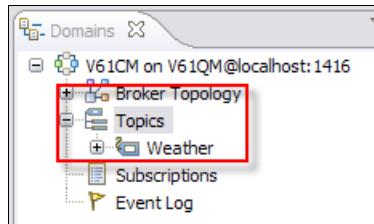


Figure 9-36 Broker Topology showing Topics1

5. The Topics view lists all of the topics in Message Broker V6.1, as shown in Figure 9-37 on page 383.

Topics				
	Principal	Publish	Subscribe	Persistent
Weather	PublicGroup	Allow	Allow	Yes
27560				
27613				

Figure 9-37 Topics view in V6.1 toolkit

6. Select the **Outline** tab, where we can find Topics, Groups, and Users that were read by the usernameserver from the operating system's local users and groups. Also by clicking the topics, you can see the Topic Access Control List window, which shows users or groups called Principal and their permissions on specific topics, as shown in Figure 9-38.

Topics				
	Principal	Publish	Subscribe	Persistent
Weather	db2admin	Inherit	Inherit	No
27560	grover	Allow	Allow	Yes
27613				

Topic/Users User/Topics	
Properties	Outline
Topics	db2admin
Groups	grover
Users	__vmware_user__ Administrator ASPNET db2admin grover Guest MUSR_MQADMIN noUserId SUPPORT_388945a0

Figure 9-38 Outline tab in V6.1 toolkit

7. The migration process of publish/subscribe takes all of the assets that are used in Message Broker V6 and tries to move them to the MQ V7.0.1 Publish/Subscribe engine. But before doing that, check WebSphere MQ Explorer V7.0.1 to see what is in MQ at the moment.

8. In the MQ Explorer V7 navigator pane, select the **Topics** tab. It is empty, as shown in Figure 9-39, prior to migration.

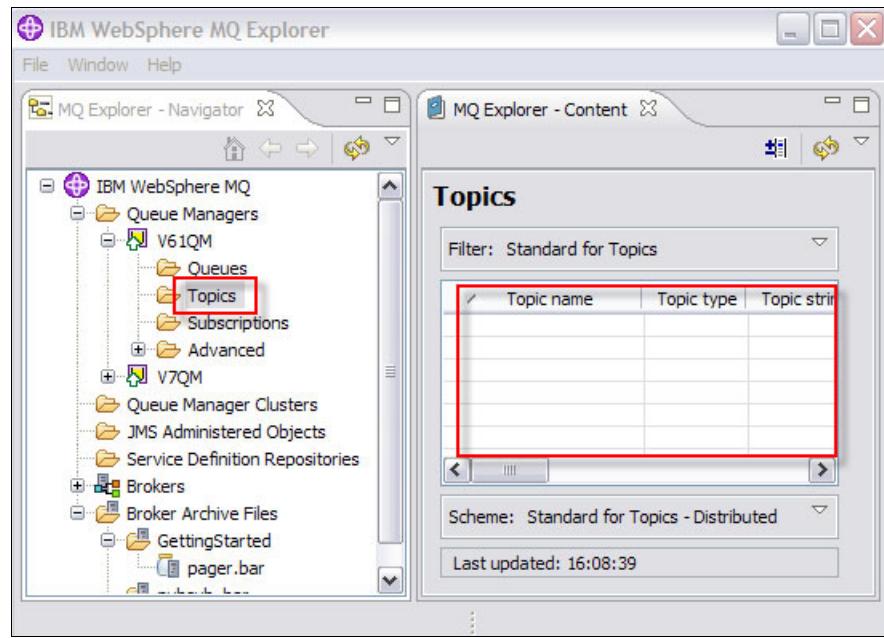


Figure 9-39 MQ Explorer V7 showing Topics

9. Also, in the MQ Explorer V7.0.1 navigator pane, select the **Subscriptions** tab. It is also empty prior to migration, as shown in Figure 9-40 on page 385.

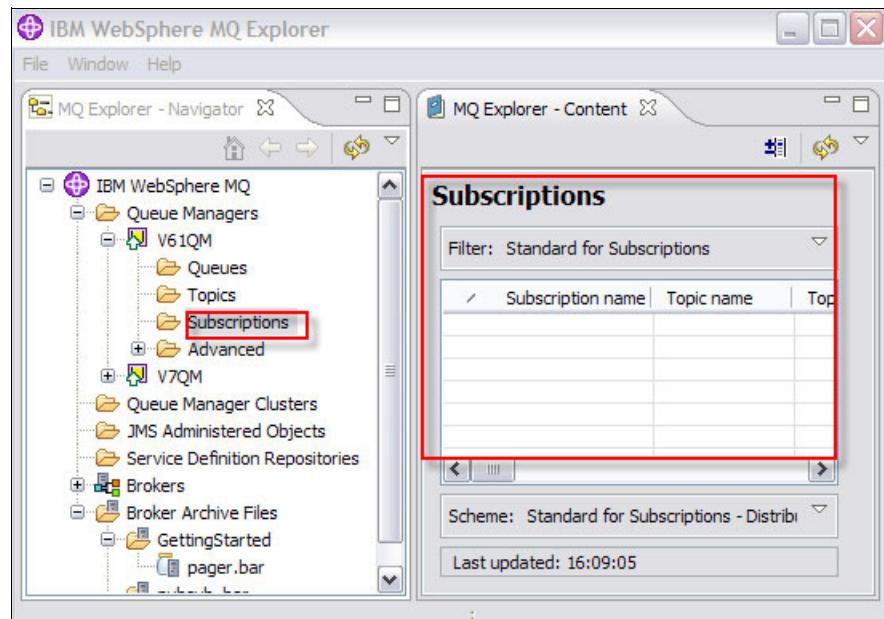


Figure 9-40 MQ Explorer V7.0.1 showing Subscriptions

10. Before starting the migration, read the overview of Access Control List (ACL) migration. This part is quite important if you are using ACLs for your publish/subscribe security.

Migrating Topic ACLs:

To migrate Topic ACLs:

1. The function that migrates publish/subscribe configuration data from WebSphere Message Broker Version 6.1 to WebSphere MQ Version 7.0.1 produces a file (amqmigrateacl.txt) that contains suggested security commands and creates topic objects, as required.
2. On WebSphere Message Broker Version 6.1, the default is that *all* user IDs have access to any topic unless the ACL explicitly restricts access. In WebSphere MQ Version 7.0.1, the default is that *no* user ID has access to any topic unless the ACL explicitly authorizes access, and it is not possible to explicitly restrict access. Because of this difference in security approaches, the migration process cannot directly migrate WebSphere Message Broker Version 6.1 ACLs to the WebSphere MQ Version 7.0.1 queue manager. Instead, the migration process writes a file (amqmigrateacl.txt) that lists the security commands that can be run separately to create equivalent ACLs in the queue manager.
3. Review and modify the commands to ensure that they set up a security environment on the queue manager that is similar to the security environment that existed in the broker. Running the migration in rehearsal mode (-r option) creates the security command file and a log file but does not perform the migration, thus allowing you to review and change the security command file before running the actual migration.
4. The migration process also creates some topic objects speculatively, based on the ACLs that are defined in the broker and in anticipation of you executing the security commands to create ACLs in the topic objects. After you resolve the security settings that you need, you can delete the topic objects that are not required.

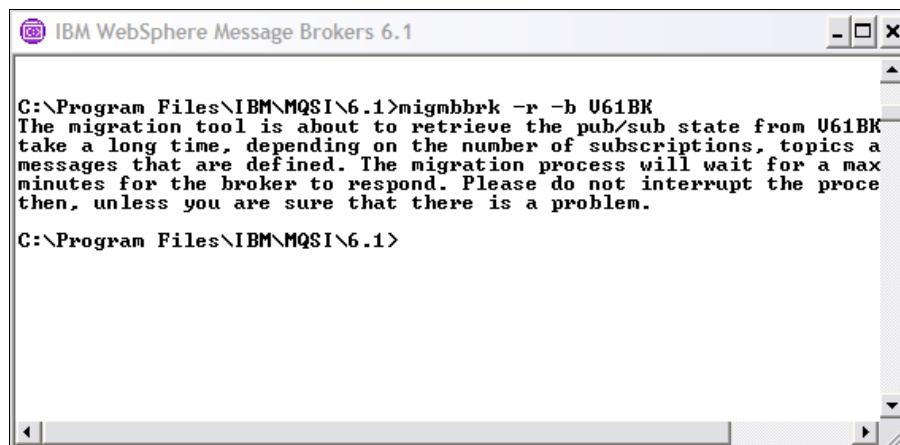
11. Run the migration process using the `migmbbrk` command to migrate the publish/subscribe information from a WebSphere Message Broker Version 6.1 broker to a WebSphere MQ Version 7.0.1 queue manager, as shown in Figure 9-41 on page 387.

Note: The migration process writes a text file, amqmigrateacl.txt, which contains `setmqaut` commands, in the current directory. These commands are best attempted by the migration process to create equivalent ACLs in the queue manager.

First use the migmbrk command with -r because this rehearses the migration of the publish/subscribe configuration data from the broker to its underlying queue manager without changing either of the configurations.

Rehearsing the migration also produces a file, amqmigrateacl.txt, that contains suggested **setmqaut** commands to set up a security environment on the queue manager that is equivalent to the security environment that existed in the broker.

Before completing the migration with the -c parameter, you must review and modify the security command file, as required, and execute the commands to set up a security environment on the queue manager that is equivalent to the one that existed in the message broker.



```
C:\Program Files\IBM\MQSI\6.1>migmbrk -r -b U61BK
The migration tool is about to retrieve the pub/sub state from U61BK
take a long time, depending on the number of subscriptions, topics and
messages that are defined. The migration process will wait for a max
minutes for the broker to respond. Please do not interrupt the process
then, unless you are sure that there is a problem.

C:\Program Files\IBM\MQSI\6.1>
```

Figure 9-41 Rehearsal phase migration

12. After it finishes, go to the directory where you executed the **migmbrk** command (in our case it is in C:\Program Files\IBM\MQSI\6.1 folder), and locate the two newly created files, amqmigmbrk.txt and amqmigrateacl.txt, as shown in Figure 9-42.

ws.ico	4 KB	Icon	10/16/2008 9:31 AM
amqmigmbrk.log	4 KB	Text Document	11/25/2009 5:12 PM
amqmigrateacl.txt	1 KB	Text Document	11/25/2009 5:12 PM

Figure 9-42 Migration log files

13. Open both files and examine their content. The amqmigmbrk.txt logs the data about topics that can be migrated. Example 9-6 on page 388 shows the snippet of these logs produced in this scenario (most of the lines here in the example are deleted).

Example 9-6 amqmigmbbrk.txt log snippet

```
2009-11-25 17:12:42.125: Migrating Pub/Sub state from WebSphere Message Broker:  
V61BK  
2009-11-25 17:12:42.125: Into queue manager: V61QM  
2009-11-25 17:12:42.125: Command switches:  
2009-11-25 17:12:42.125: -r  
2009-11-25 17:12:43.484: Starting to parse subscriptions ...  
2009-11-25 17:12:43.484: Migrating subscriptions for topic string Weather/#  
2009-11-25 17:12:43.484: [1] Migrating subscription for:  
2009-11-25 17:12:43.484: Format: mqrfh2  
2009-11-25 17:12:43.484: Queue Manager: V61QM  
2009-11-25 17:12:43.484: Queue: OUT  
2009-11-25 17:12:43.484: Migrating subscriptions for topic string  
$SYS/Broker/+/Subscription/#  
2009-11-25 17:12:43.484: [2] Migrating subscription for:  
2009-11-25 17:12:43.484: Format: mqrfh2  
2009-11-25 17:12:43.484: Queue Manager: V61QM  
2009-11-25 17:12:43.484: Queue: SYSTEM.BROKER.ADMIN.REPLY  
2009-11-25 17:12:43.484: CorrelId: 4366674d677253756234  
2009-11-25 17:12:43.484: ... finished parsing subscriptions  
2009-11-25 17:12:43.484: Starting to parse topics ...  
2009-11-25 17:12:43.484: Migrating ACLs for topic string Weather/27560  
2009-11-25 17:12:43.484: Migrating ACLs for topic string Weather  
2009-11-25 17:12:43.484: Migrating ACLs for topic string Weather/27613  
2009-11-25 17:12:43.484: Migrating ACLs for topic string  
2009-11-25 17:12:43.484: Migrating ACLs for topic string Weather/#  
2009-11-25 17:12:43.484: None found.  
2009-11-25 17:12:43.484: Starting to parse retained publications ...  
2009-11-25 17:12:43.484: Migrating retained publications for topic string  
$SYS/Broker/V61BK/Status  
2009-11-25 17:12:43.484: Migrating retained publication for default subscription point.  
2009-11-25 17:12:43.796: ... finished parsing retained publications  
2009-11-25 17:12:43.796:  
All Pub/Sub data has been retrieved from the broker.  
2009-11-25 17:12:43.796: The -r (rehearse) switch was specified.
```

2009-11-25 17:12:43.796: No changes will be made to the queue manager pub/sub state.

In amqmigrateacl.txt, there is data about ACLs that can be migrated and if ACL migration can be done automatically or needs to be altered manually. Example 9-7 shows the snippet of amqmigrateacl.txt.

Example 9-7 amqmigrateacl.txt snippet

Migrating Publish/Subscribe ACLs.

From WebSphere Message Broker: V61BK

To WebSphere MQ Queue Manager: V61QM

Timestamp: 2009-11-25 17:12:43.796

The root of the topic tree in V61BK has been changed to the same setting that is used by MQ. Furthermore, the topic tree contains only positive ACLs.

Therefore it is possible to migrate the ACLs directly from V61BK to V61QM as follows.

```
setmqaut -m V61QM -n MIGMBBRK.TOPIC.00003 -t topic -p grover +pub +sub  
setmqaut -m V61QM -n MIGMBBRK.TOPIC.00002 -t topic -p grover +pub +sub  
setmqaut -m V61QM -n MIGMBBRK.TOPIC.00001 -t topic -p grover +pub +sub
```

14. After you review the migration log files and are satisfied with the given results, you can start the real migration. First use -t parameter to create topic objects that might be needed in the queue manager, based on the ACL entries that are defined in the message broker. Use of the -t parameter also produces/updates a file, amqmigrateacl.txt, that contains suggested **setmqaut** commands to set up a security environment on the queue manager that is equivalent to the security environment that existed in the message broker.

The topic objects are created in anticipation of you executing the security commands to create ACLs for the topic objects. Before you complete the migration with the -c parameter you must review and modify the security command file, as required, and execute the commands to set up a security environment on the queue manager that is equivalent to the one that existed in the message broker.

You must run this phase before you run the completion phase with the -c parameter. Also use -z parameter to run the migration process, regardless of whether it previously ran to successful completion, as shown in Figure 9-43 on page 390.

Enter: migmbbrk -t -z -b V61BK

The screenshot shows a terminal window titled "IBM WebSphere Message Brokers 6.1". The command entered is "C:\Program Files\IBM\MQSI\6.1>migmbbrk -t -z -b U61BK". The output message reads: "The migration tool is about to retrieve the pub/sub state from U61BK take a long time, depending on the number of subscriptions, topics and messages that are defined. The migration process will wait for a max minutes for the broker to respond. Please do not interrupt the process then, unless you are sure that there is a problem." Below the command prompt, it says "C:\Program Files\IBM\MQSI\6.1>".

Figure 9-43 Initial phase migration

15. Check the migrated topics (note that ACLs are not implemented yet!). Go to WebSphere MQ Explorer, and click **Topics**, as shown in Figure 9-44.

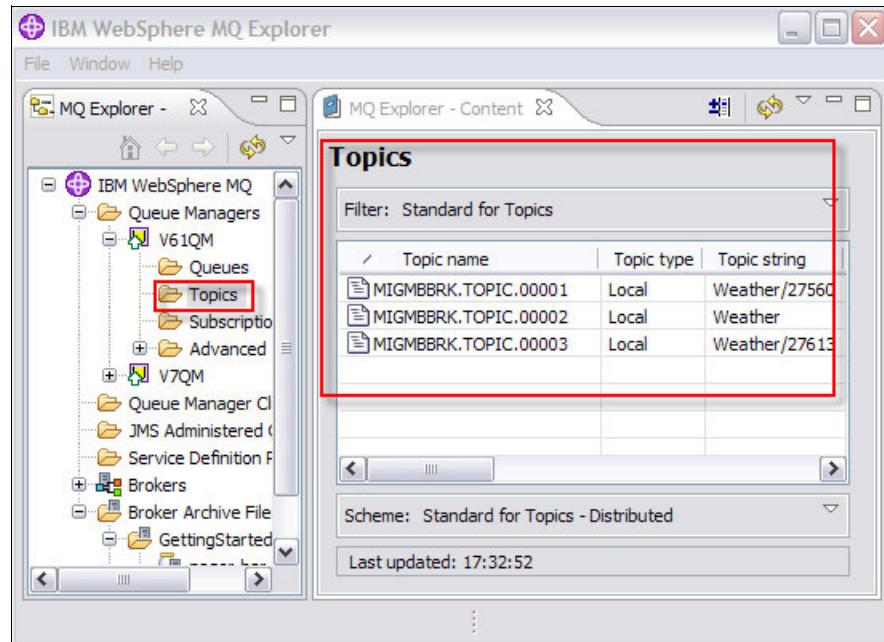
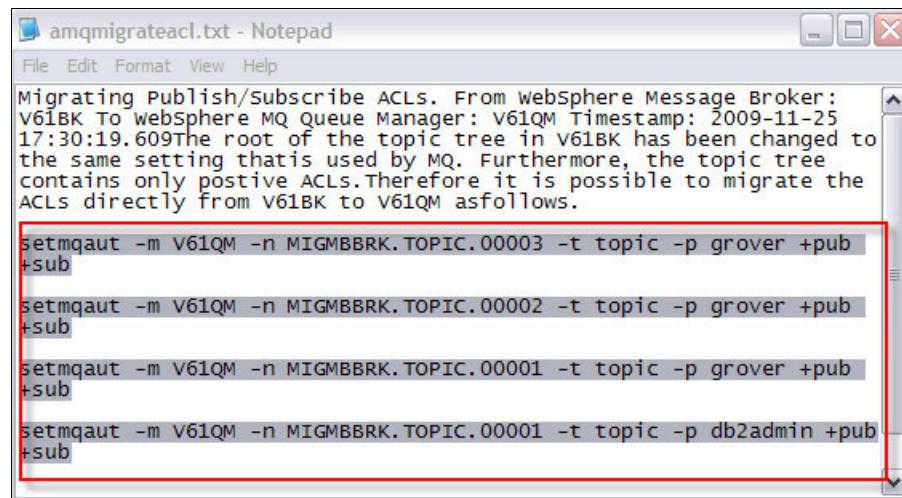


Figure 9-44 Migrated Topics list

16. Now set the appropriate ACLs on topics. In this simple example, all we need to do is to open the amqmigrateacl.txt file and copy **setmqaut** commands, as shown in Figure 9-45.

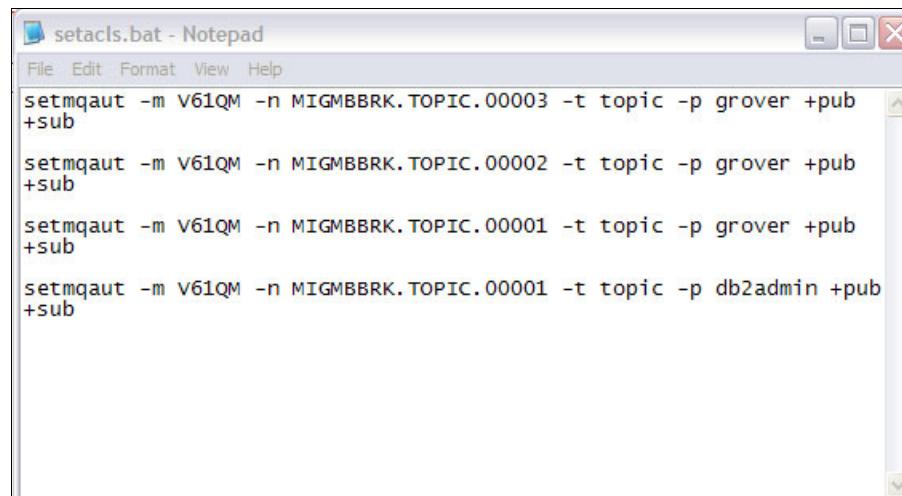


The screenshot shows a Windows Notepad window titled "amqmigrateacl.txt - Notepad". The window contains text explaining the migration of Publish/Subscribe ACLs from WebSphere Message Broker V61BK to WebSphere MQ Queue Manager V61QM. It states that the root of the topic tree in V61BK has been changed to the same setting used by MQ, and that the topic tree contains only positive ACLs, making direct migration possible. Below this text, four "setmqaut" commands are listed, each enclosed in a red rectangular box:

```
setmqaut -m V61QM -n MIGMBBRK.TOPIC.00003 -t topic -p grover +pub+sub
setmqaut -m V61QM -n MIGMBBRK.TOPIC.00002 -t topic -p grover +pub+sub
setmqaut -m V61QM -n MIGMBBRK.TOPIC.00001 -t topic -p grover +pub+sub
setmqaut -m V61QM -n MIGMBBRK.TOPIC.00001 -t topic -p db2admin +pub+sub
```

Figure 9-45 The amqmigrateacl.txt file

17. Paste the commands in a new .txt file, and save the file as a .bat file (setacls.bat), as shown in Figure 9-46.

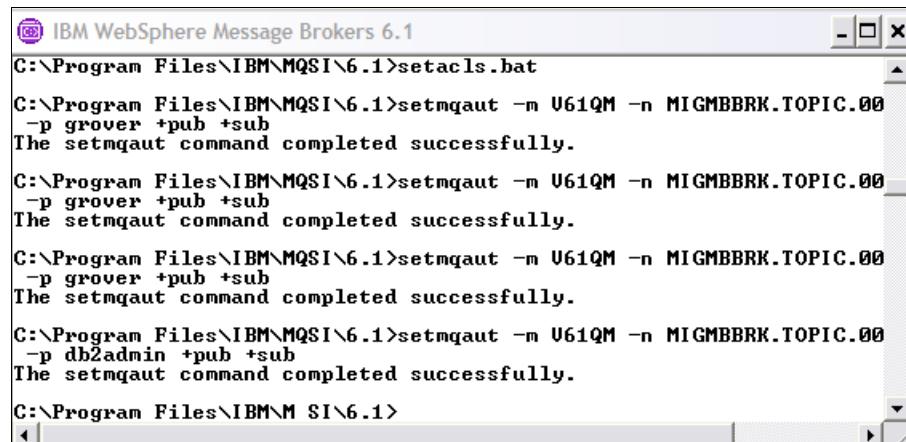


The screenshot shows a Windows Notepad window titled "setacls.bat - Notepad". The window contains the same four "setmqaut" commands as the previous file, pasted into a new batch file. The commands are identical to those in Figure 9-45:

```
setmqaut -m V61QM -n MIGMBBRK.TOPIC.00003 -t topic -p grover +pub+sub
setmqaut -m V61QM -n MIGMBBRK.TOPIC.00002 -t topic -p grover +pub+sub
setmqaut -m V61QM -n MIGMBBRK.TOPIC.00001 -t topic -p grover +pub+sub
setmqaut -m V61QM -n MIGMBBRK.TOPIC.00001 -t topic -p db2admin +pub+sub
```

Figure 9-46 Paste the setmqaut into a new file

18. Execute the **setacl.bat** command, as shown in Figure 9-47. At this point all of the authorizations regarding topics are set.



```
C:\Program Files\IBM\MQSI\6.1>setacl.bat  
C:\Program Files\IBM\MQSI\6.1>setmqaut -m V61QM -n MIGMBBRK.TOPIC.00  
-p grover +pub +sub  
The setmqaut command completed successfully.  
C:\Program Files\IBM\MQSI\6.1>setmqaut -m V61QM -n MIGMBBRK.TOPIC.00  
-p grover +pub +sub  
The setmqaut command completed successfully.  
C:\Program Files\IBM\MQSI\6.1>setmqaut -m V61QM -n MIGMBBRK.TOPIC.00  
-p db2admin +pub +sub  
The setmqaut command completed successfully.  
C:\Program Files\IBM\MQSI\6.1>
```

Figure 9-47 Results of executing .bat

19. Now check the authorization settings in MQ Explorer by navigating to **Queue Managers** → **V61QM** → **Topics**.
20. Right-click one of the topics, and choose **Object Authorities** → **Manage Authority Records**, as shown in Figure 9-48 on page 393.

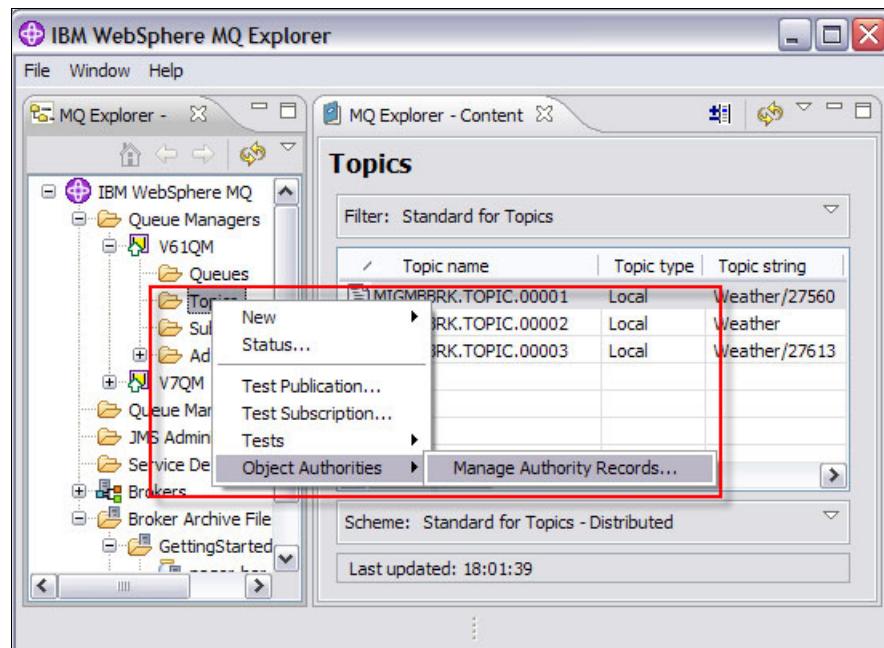


Figure 9-48 Manage Authority Records

21. Expand **Specific Profiles** to select the broker topic, and click the **Users** tab. The migrated user rights are displayed. Check the applied permissions, as shown in Figure 9-49 on page 394.

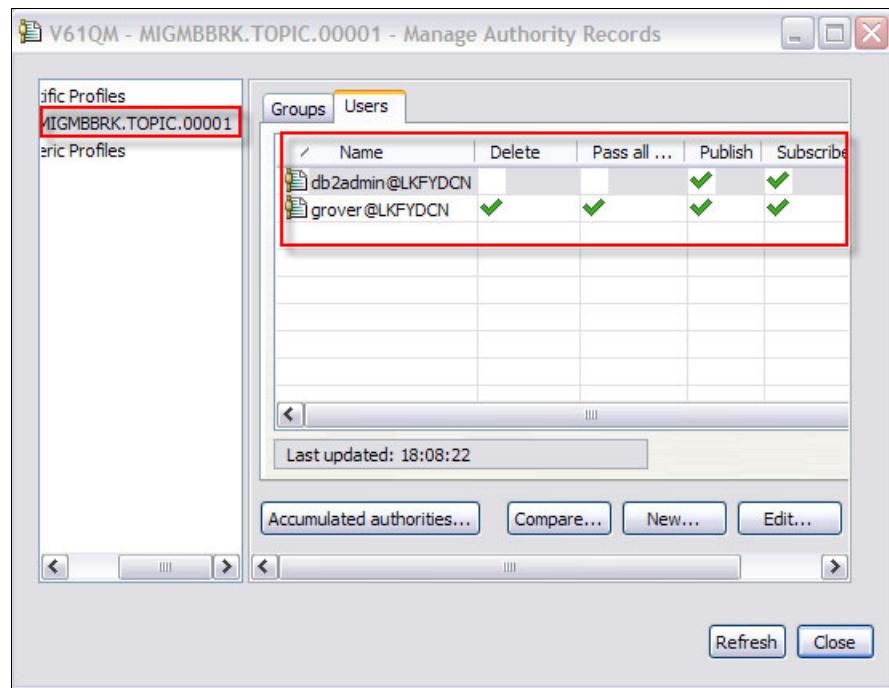


Figure 9-49 Users view showing migrated user rights

22. Complete the migration of the publish/subscribe configuration data using the **-c** parameter. The completion phase of the migration uses the topic objects that are created in the initial **-t** phase. It is possible that the broker state changed since the initial phase was run and that new additional topic objects are now required. If this is the case, the completion phase creates these new topic objects as necessary.

The completion phase does not delete any topic objects that are not needed; instead, you must delete any unrequired topic objects.

Before completing the migration, you must review and modify the security command file that was produced in the **-r** or **-t** phase, as required, and execute the commands to set up a security environment on the queue manager that is equivalent to the one that existed in the message broker.

As shown in Figure 9-50 on page 395, type the command: **migmbbrk -c -z -b V61BK**.

The screenshot shows a terminal window titled "IBM WebSphere Message Brokers 6.1". The output of the command "migmbbrk -c -z -b V61BK" is displayed. The message indicates that the migration tool is about to retrieve pub/sub state from V61BK, noting it may take a long time depending on the number of subscriptions and topics. It also states that the migration process will wait for a maximum of 10 minutes for the broker to respond. A success message "BIP8071I: Successful command completion." is shown at the end. The command prompt "C:\Program Files\IBM\MQSI\6.1>" is visible at the bottom.

Figure 9-50 Completion phase migration

23. In MQ Explorer check the migrated subscriptions, as shown in Figure 9-51.

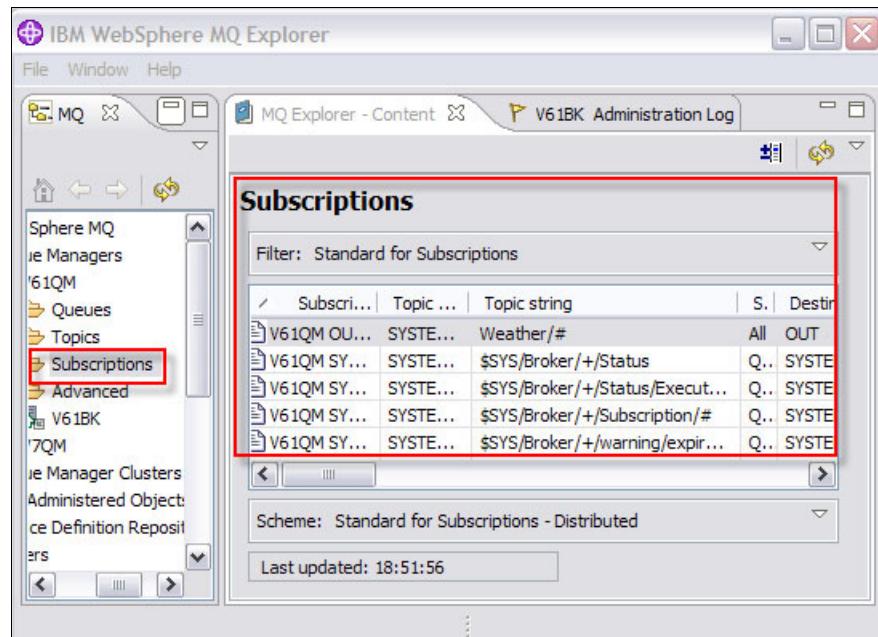


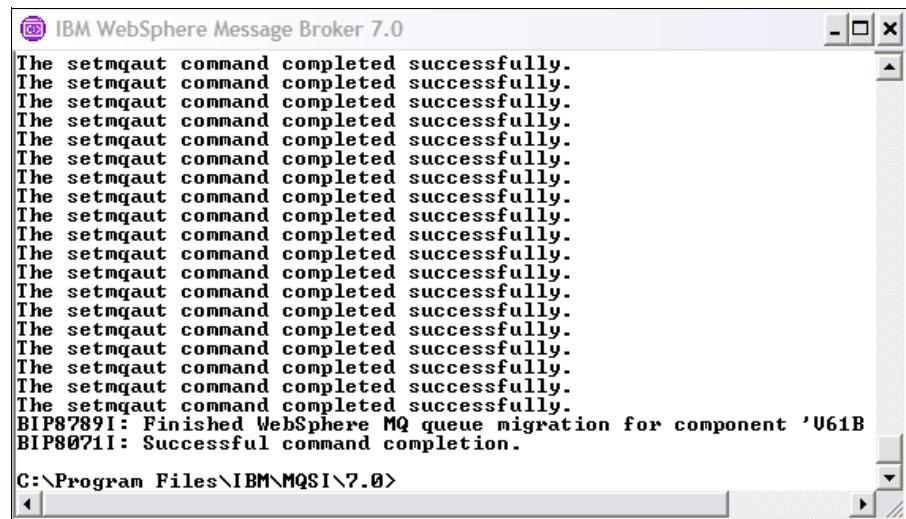
Figure 9-51 Migrated subscriptions

24. Use the **mqsimigratecomponents** command to migrate the whole broker now from a V6.1 to V7.

When migrating the message broker to Version 7.0, its configuration is read from the broker database and is migrated to xml files that are maintained by the migrated broker. The broker database used in V6.1 or V6.0 is no longer required by the V7.0 message broker. When you successfully complete migration, you can drop the database or uninstall the database product, if you have no further use for it, in which case, you might not be able to migrate back to V6.0 or V6.1.

You must run this command from whichever version of the installed product is the later, regardless of whether it is the source version or the target version.

25. Open the WebSphere Message Broker 7.0 Command Console, and run the command: **mqsimigratecomponents V61BK**, as shown in Figure 9-52.



The screenshot shows a Windows command-line interface window titled "IBM WebSphere Message Broker 7.0". The window contains the following text output:

```
The setmqaut command completed successfully.  
BIP8789I: Finished WebSphere MQ queue migration for component 'V61B'  
BIP8071I: Successful command completion.
```

The command prompt at the bottom of the window shows "C:\Program Files\IBM\MQSI\7.0>".

Figure 9-52 The **mqsimigratecomponents** command

26. Start the broker from the Message Broker Explorer, as shown in Figure 9-53 on page 397 or using the command: **mqsistart V61BK**.

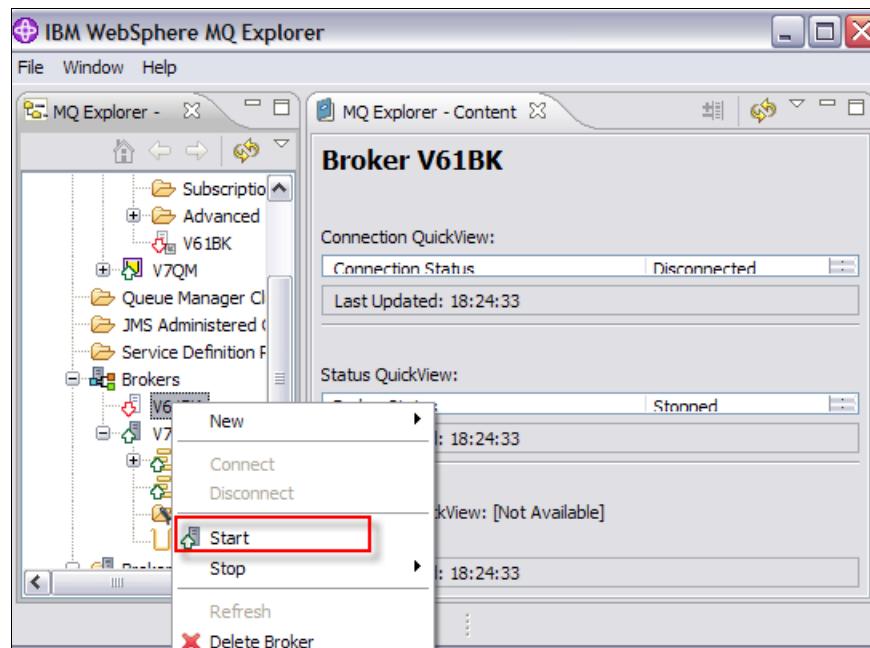


Figure 9-53 Start the broker

27. Choose the migrated broker to check its version, as shown in Figure 9-54 on page 398.

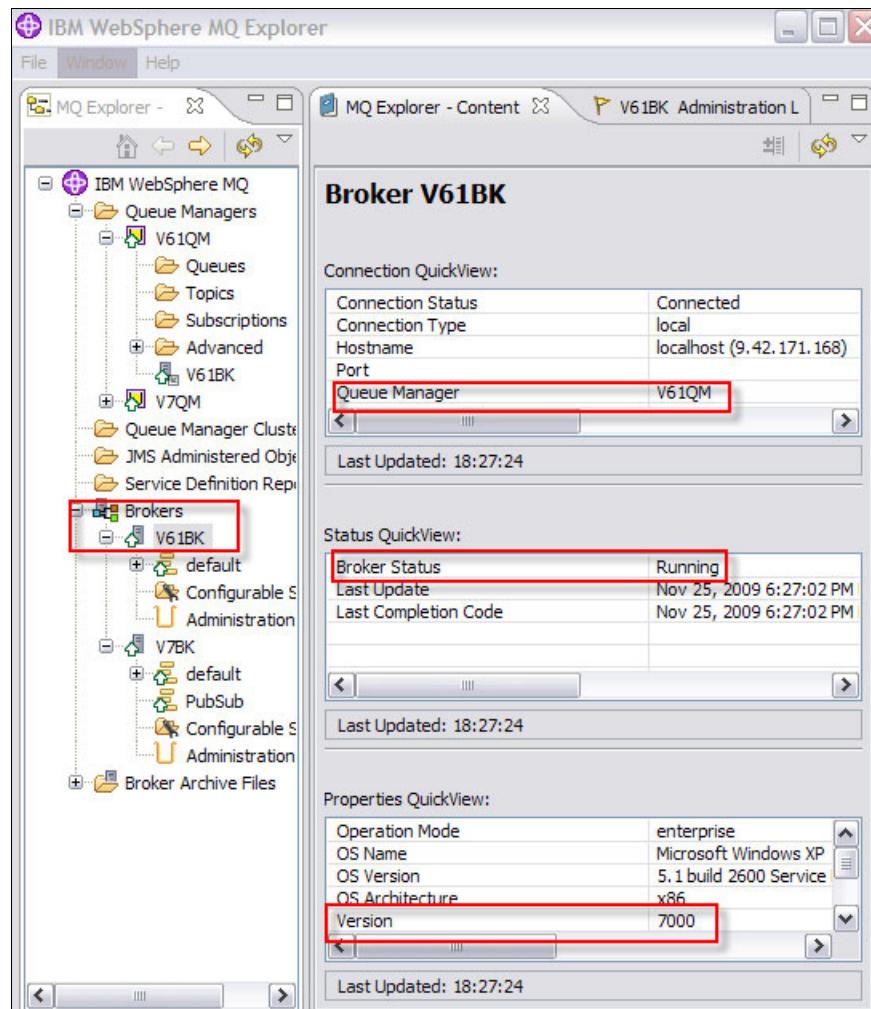


Figure 9-54 Broker version

Notice the version is: 7000, which indicates that the broker V61BK is now migrated to V7.

28. After the broker is migrated, set up the queue manager way of handling publish/subscribe. Right-click **V61QM**, and choose **Properties**, as shown in Figure 9-55 on page 399.

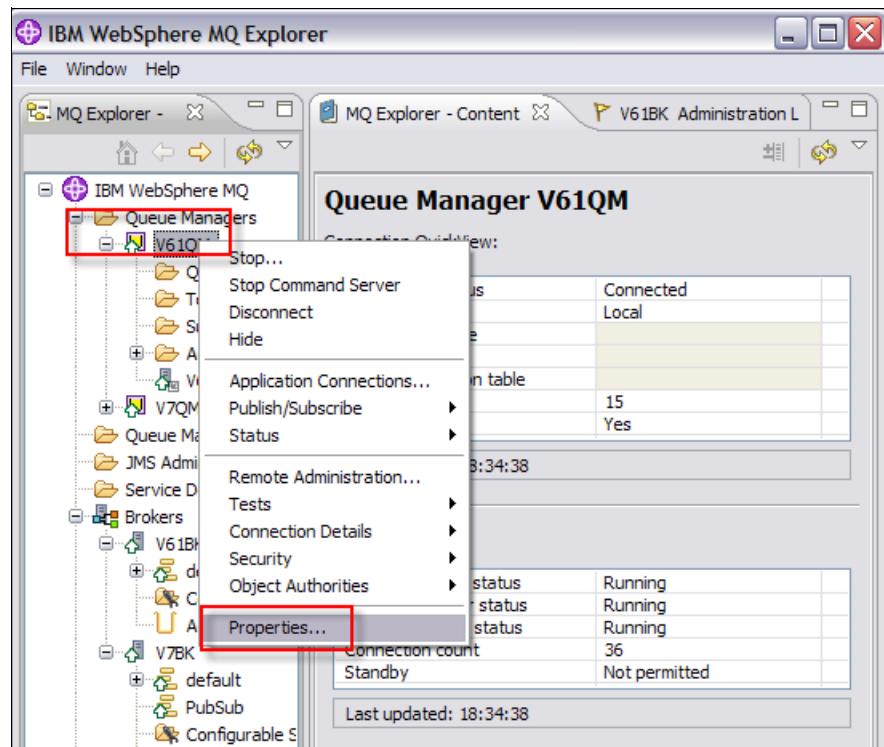


Figure 9-55 Changing queue manager properties

29. Choose **Publish/Subscribe properties**, and set the Publish/Subscribe mode as **Enabled**, as shown in Figure 9-56 on page 400.

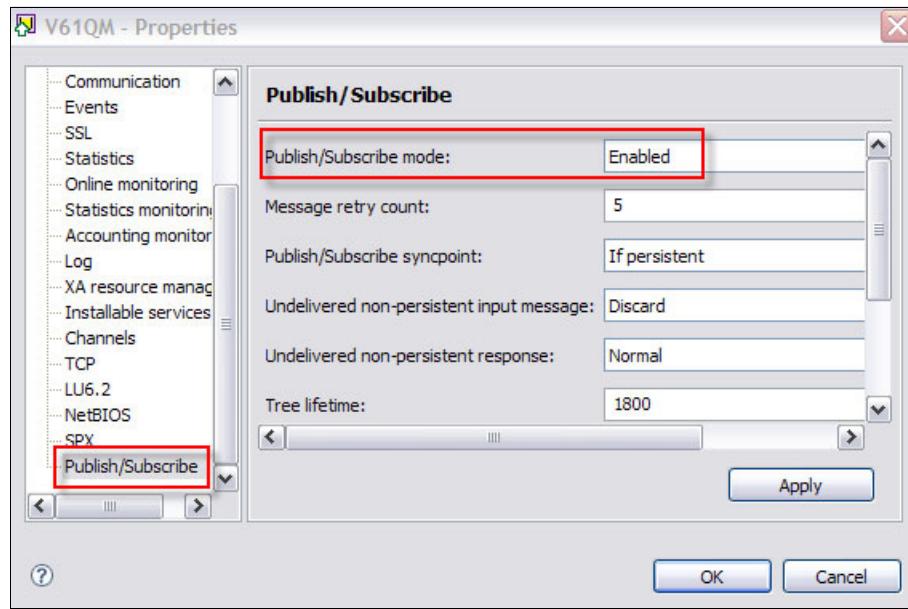


Figure 9-56 Publish/Subscribe mode

30. At this point in time, you should have all of the components that make up your Publish/Subscribe example. You successfully migrated topics, subscriptions, ACLs, and message flows.

You can now test the scenario.

Testing the scenario

To test the scenario:

1. Start rfhutil.exe, and browse to go to the sample publication message, as shown in Figure 9-57 on page 401.

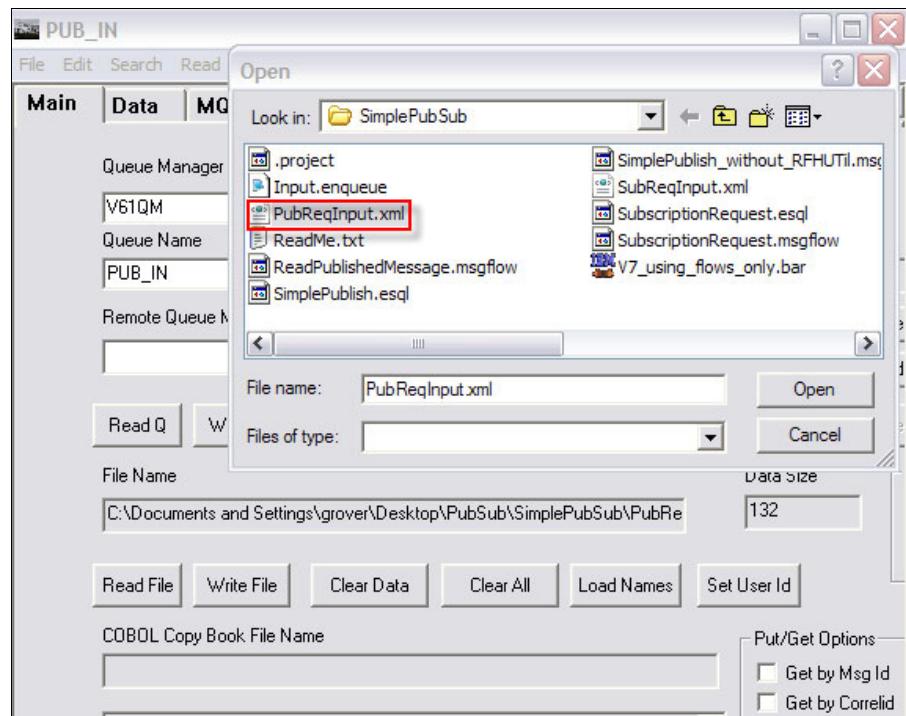


Figure 9-57 Using mqsitl browse to the message

2. Check to see if the messages went to the appropriate queue in WebSphere MQ Explorer, as shown in Figure 9-58 on page 402.

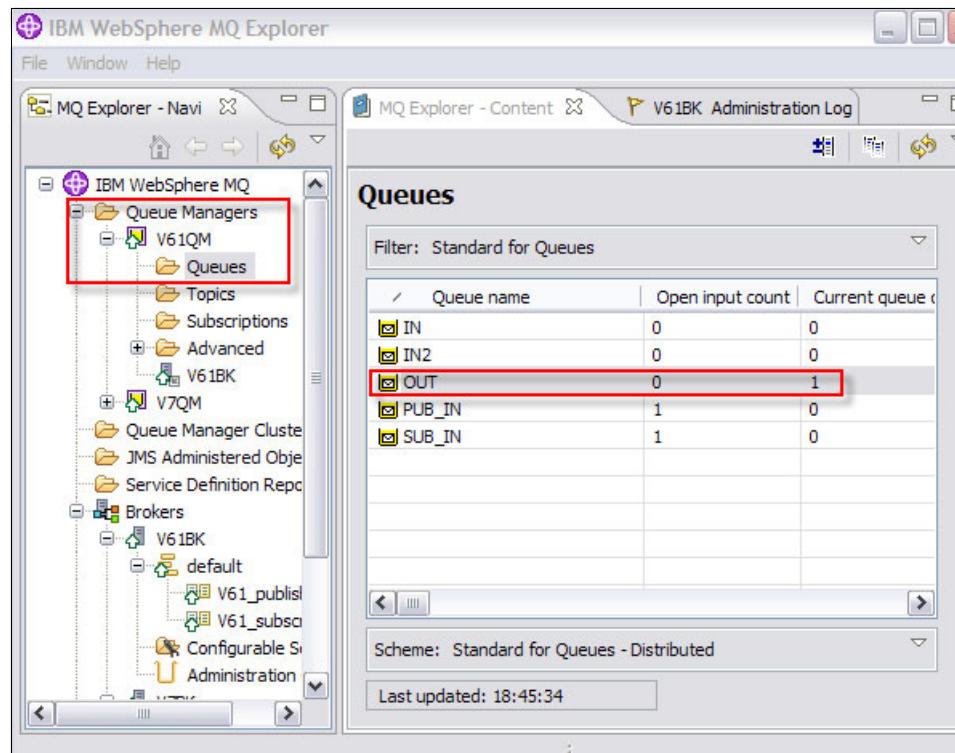


Figure 9-58 Output message gone to destination queue

The message went to the local queue OUT on the migrated queue manager V61QM, which indicates that the migration task was performed correctly and for all subsequent use, WebSphere Message Broker V7 can be used.

9.5 Scenario 3: Content-based filtering

When creating subscriptions, you can choose to subscribe to a specific topic, such as Weather, as in our example application of a Weather reporting system. You can also subscribe to specific content, for example, Weather, for certain zipcodes only. The difference here is that the content-based subscription is filtering out information from the topic, giving the user a much more refined result. This filtering takes place on the message broker.

Content-based filtering in Message Broker allows a subscriber to restrict the messages that it wants to receive based on the content of the message. The filters are specified as ESQL expressions. WebSphere MQ V7.0.1 supports a limited filtering capability on the message header and message properties but

not the message body. If the filter string specified on the subscription refers to the content of the message, WebSphere MQ requests the extended message selection provider, for example, Message Broker to process the messages, as shown in Example 9-8.

Example 9-8 Processing the message

Processing of the filter *Root.MQMD.Format* like '*MQHRF%*' that references a part of *MQMD*, can be performed by MQ alone and might not need a broker.

For filtering based on the message body or the content, use Message broker. Message Broker supports filtering on the whole message, including the body of the message, as shown in Example 9-9.

Example 9-9 Filtering on the body of the message

Processing of the filter *Body.Name.Number='XK5000'* that makes a reference to an element within the body of the message requires a message broker, Figure 9-59 on page 404.

When a client registers a subscription, it can specify a filter to be applied to the content of fields within each publication message. The message broker then validates the filter against the ESQL that the subscriber used. When the publisher sends the message on that topic, broker evaluates the filter against the message and returns it to the queue manager, which then routes the message appropriately based on the success or failure of the matching filter.

The filter processing is performed in the message broker by executing the ESQL in the message flow that is deployed to the Execution Group. The content-based filtering service that Message Broker provides runs within the nominated execution groups. The execution groups are nominated by enabling the content-based filtering user preference, which can be done in one of the following ways:

- ▶ Enable content-based filtering by going to the Brokers and selecting **Execution Group Properties** → **ContentBasedFiltering** → **Content Based Filtering in MQ Explorer**.
- ▶ From the Message Broker V7 command console, run the command
`mqsicchangeproperties <Broker> -e <Execution group> -o ContentBasedFiltering -n cbfEnabled -v true`

Note: To view the PubSub attribute values, you can run the following command:
`mqsireportproperties V7BK -e PubSub -o ContentBasedFiltering -r`

Figure 9-59 shows the Execution Group, PubSub's, properties from where content-based filtering can be enabled in the Message Broker Explorer.

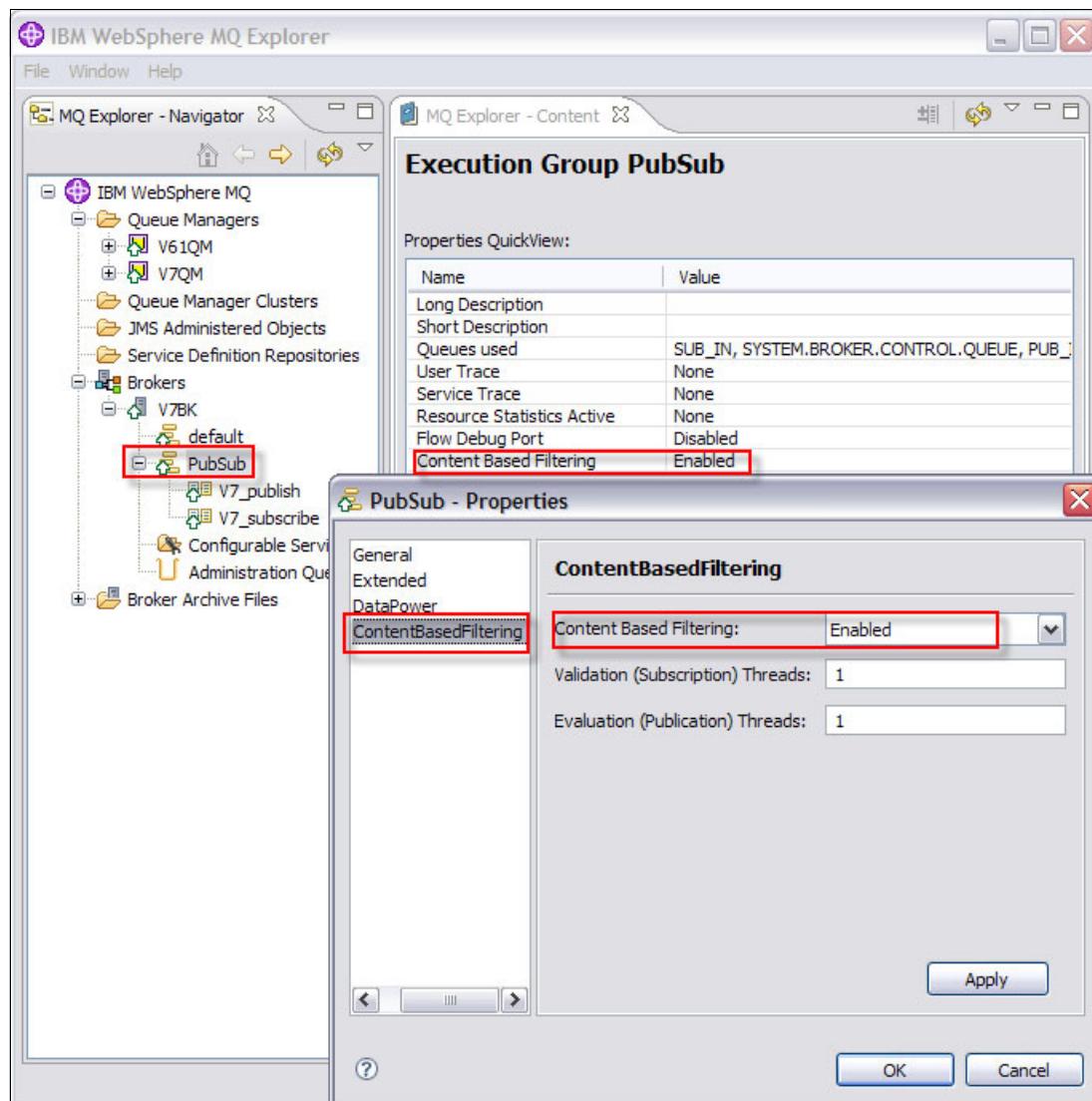


Figure 9-59 Enable ContentBasedFiltering on the Execution Group

If a filter or selector is specified on the subscription and WebSphere MQ can evaluate an expression without any message broker assistance, broker will not be used. There is a configuration option that is available on queue manager

properties. This parameter can be set in MQ Explorer by right-clicking **QueueManager** → **Properties** → **Publish/Subscribe** → **Publish/Subscribe mode**. The possible values for Publish/Subscribe mode are:

- ▶ Enabled = Pass to message broker if required
- ▶ Compat = Never pass selection to message broker
- ▶ Disabled = Always pass selection to message broker

9.5.1 Demonstrating publish/subscribe with content-based filtering using Message Broker

This example demonstrates how publish/subscribe works when content-based filtering is enabled on the Execution Group. For simplicity, we can use the same message flow project that we demonstrated in the previous examples. The subscriber application registers a subscription on topic Weather with a filter Root.XMLNSC.Forecast.Temperature>30 that indicates that it only wants to see a weather report for areas (zip) that have temperature more than 30 degrees. So, when a publication message arrives on the message broker for a zip having a temperature of more than 30 degrees, the filter is evaluated to true by the message broker, and the subscriber queue gets the message. However, if a publication arrives on the broker for a zip that has a temperature that is less than 30 degrees, the filter is evaluated to false by the broker, and the subscriber queue does not receive any messages. See Appendix A, “Additional material” on page 533 or a sample Project Interchange V7_flows.zip, which contains the message flows V7_publish.msgflow and V7_subscribe.msgflow.

Although, it is not required to have message flows when publishing or subscribing with content-based filtering, in this scenario, we use the message flow V7_publish.msgflow and V7_subscribe.msgflow:

1. Create a message flow called V7_publish.msgflow in the Message Broker Toolkit. Figure 9-60 shows the message flow with the nodes.



Figure 9-60 Publication message flow

The MQInput node is configured to use the Queue Name PUB_IN and enter Message Parsing > Message Domain as XMLNSC. All of the other parameters are set to default.

The compute node, Publish, references the ESQL shown in Figure 9-61 on page 406.

```

CREATE COMPUTE MODULE V7_publish_Compute
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN

    CALL CopyMessageHeaders();
    -- CALL CopyEntireMessage();
    -- Set up the Publication
    SET OutputRoot.MQRFH2.psc.Command = 'Publish';
    SET OutputRoot.MQRFH2.psc.PubOpt = 'Local';
    SET OutputRoot.MQRFH2.psc.Topic = InputRoot.XMLNSC.Forecast.Topic;
    SET OutputRoot.MQRFH2.psc.zip = InputRoot.XMLNSC.Forecast.zip;
    SET OutputRoot.XMLNSC.Forecast.Temperature = InputRoot.XMLNSC.Forecast.Text;

    RETURN TRUE;
END;

CREATE PROCEDURE CopyMessageHeaders() BEGIN
    DECLARE I INTEGER 1;
    DECLARE J INTEGER;
    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;

```

Figure 9-61 ESQL of Publication message flow

2. The compute node, Publish, references the ESQL, as shown in Example 9-10.

Example 9-10 Publish references the ESQL

```

CREATE COMPUTE MODULE V7_publish_MQ_Compute
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
    CALL CopyMessageHeaders();
    -- CALL CopyEntireMessage();
    -- Set up the Publication
    SET OutputRoot.MQRFH2.psc.Command = 'Publish';
    SET OutputRoot.MQRFH2.psc.PubOpt = 'Local';
    SET OutputRoot.MQRFH2.psc.Topic = InputRoot.XMLNSC.Forecast.Topic;
    SET OutputRoot.MQRFH2.psc.zip = InputRoot.XMLNSC.Forecast.zip;
    RETURN TRUE;
END;
CREATE PROCEDURE CopyMessageHeaders() BEGIN
    DECLARE I INTEGER 1;

```

```

DECLARE J INTEGER;
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;

```

3. In the Message Broker Toolkit, create a message flow called V7_subscribe.msgflow. Figure 9-62 shows the message flow with the nodes. You can also use rfhutil.exe, as shown in previous examples, to register the subscription with a filter in which case you might not need this message flow.



Figure 9-62 Subscriber message flow

The MQInput node is configured to use the Queue Name SUB_IN and Input Message Parsing > Message Domain as XML. All of the other parameters are set to default.

4. The compute node Subscribe references the ESQL that is shown in Figure 9-63 on page 408. Review the ESQL code that sets the filter:
SET OutputRoot.MQRFH2.psc.Filter = 'Root.XMLNSC.Forecast.Temperature > 30';

```

CREATE COMPUTE MODULE V7_subscribe_Compute
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN

    CALL CopyMessageHeaders();
    -- CALL CopyEntireMessage();

    -- Set up the subscription request
    SET OutputRoot.MQRFH2.psc.Command = 'RegSub';
    SET OutputRoot.MQRFH2.psc.Topic = 'Weather';
    SET OutputRoot.MQRFH2.psc.QMgrName = 'V7QM';
    SET OutputRoot.MQRFH2.psc.Filter = 'Root.XMLNSC.Forecast.Temperature > 30';
    SET OutputRoot.MQRFH2.psc.QName = 'OUT';

    RETURN TRUE;
END;

CREATE PROCEDURE CopyMessageHeaders() BEGIN
DECLARE I INTEGER 1;
DECLARE J INTEGER;
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;

```

Figure 9-63 Subscriber flow ESQL

5. The compute node Subscribe references the ESQL shown in Example 9-11. Review the ESQL code that sets the filter:

```
SET OutputRoot.MQRFH2.psc.Filter = 'Root.XMLNSC.Forecast.Temperature
> 30';
```

Example 9-11 ESQL code

```

CREATE COMPUTE MODULE V7_subscribe_Compute
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
    CALL CopyMessageHeaders();
    -- CALL CopyEntireMessage();
    -- Set up the subscription request
    SET OutputRoot.MQRFH2.psc.Command = 'RegSub';
    SET OutputRoot.MQRFH2.psc.Topic = 'Weather';
    SET OutputRoot.MQRFH2.psc.QMgrName = 'V7QM';

```

```

SET OutputRoot.MQRFH2.psc.Filter = 'Root.XMLNSC.Forecast.Temperature
> 30';
SET OutputRoot.MQRFH2.psc.QName = 'OUT';
RETURN TRUE;
END;
CREATE PROCEDURE CopyMessageHeaders() BEGIN
DECLARE I INTEGER 1;
DECLARE J INTEGER;
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 MQOutput node is configured with the queue name
SYSTEM.BROKER.CONTROL.QUEUE.

6. Deploy the message flows V7_publish.msgflow and V7_subscribe.msgflow to the execution group PubSub in the message broker V7BK. Figure 9-64 shows the deployed message flow to the execution group PubSub.

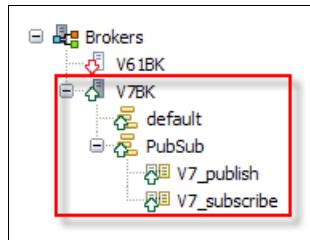


Figure 9-64 Message Broker V7 Broker Topology

7. Ensure that the broker queue manager V7QM is publish/subscribe enabled. Figure 9-65 on page 410 shows MQ Explorer showing the queue manager **V7QM** → **Properties** → **Publish/subscribe**. The publish/subscribe mode is set to Enabled.

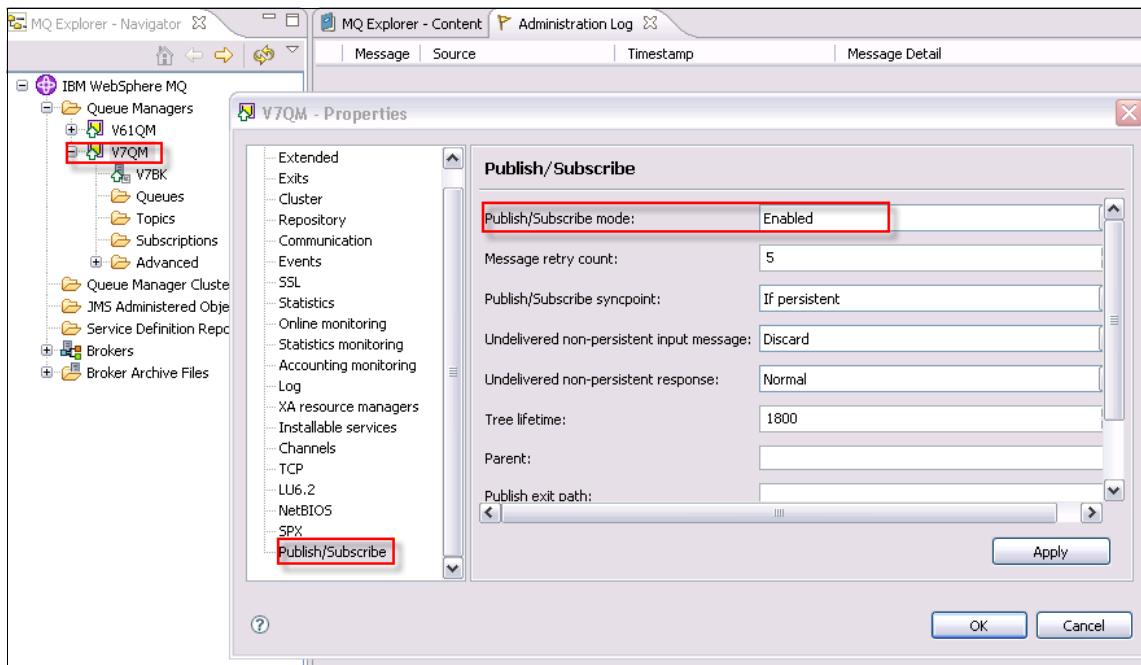


Figure 9-65 QueueManager PubSub mode

8. Ensure that the execution group PubSub in the broker V7BK has content-based filtering set to Enabled. Figure 9-66 on page 411 shows the Content Based Filtering property as Enabled.

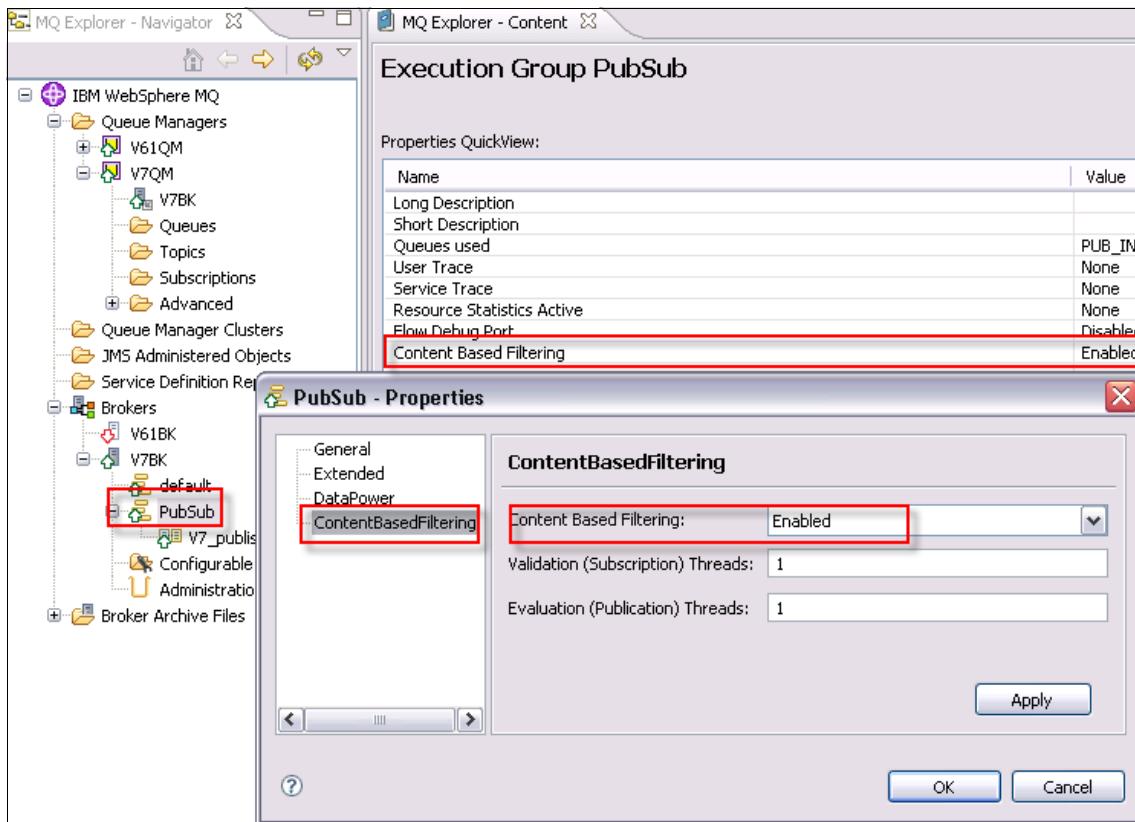


Figure 9-66 ContentBasedFiltering on the Execution Group properties

9. Stop the broker V7BK using the command **mqsistop V7BK**, and stop the queue manager V7QM using the command **endmqm V7QM**.
10. Restart the queue manager V7QM using the command **strmqm V7QM**, and start the broker V7BK using the command **mqsistart V7BK** so that the configuration changes can take effect.
11. To register a subscription, use MQ Explorer to put a message on the queue SUB_IN, as shown in Figure 9-67 on page 412.

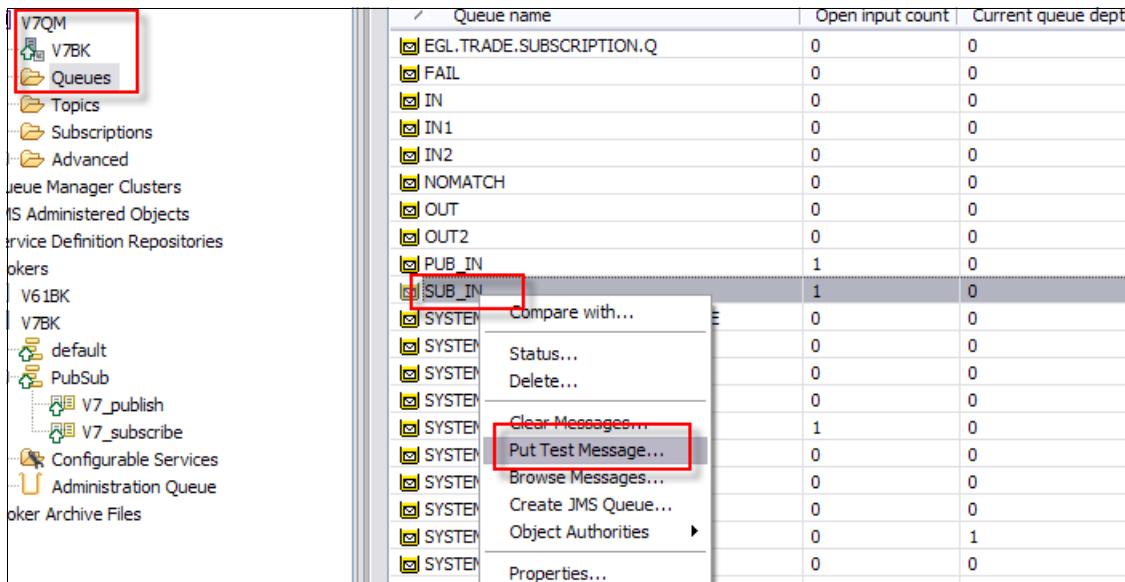


Figure 9-67 Put a message on Subscriber flow Input queue

A test subscription xml message is put to the SUB_IN queue, as shown in Figure 9-68, to register the subscription through the message flow V7_subscribe.msgflow.

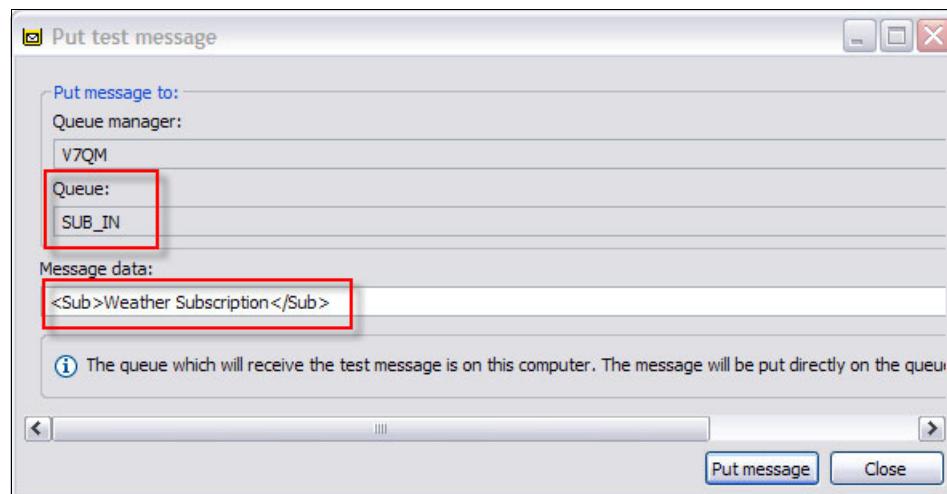


Figure 9-68 Put test message

A test subscription xml message is put to the *SUB_IN* queue, as shown in Example 9-12, to register the subscription through the message flow V7_subscribe.msgflow.

Example 9-12 SUB_IN queue

```
<Sub>Weather Subscription</Sub>
```

The subscription is now registered and can be viewed in the **WebSphere MQ Explorer** → **Queue Managers** → **V7QM** → **Subscriptions** pane, as shown in Figure 9-69.

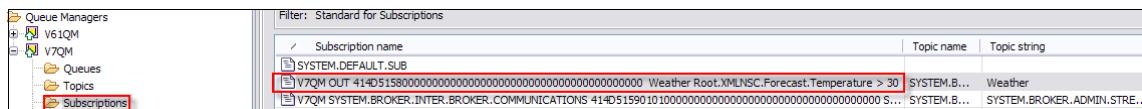


Figure 9-69 Subscription registration

12. Use rfhutil.exe to publish a message on the queue PUB_IN, as shown in Figure 9-70 on page 414.

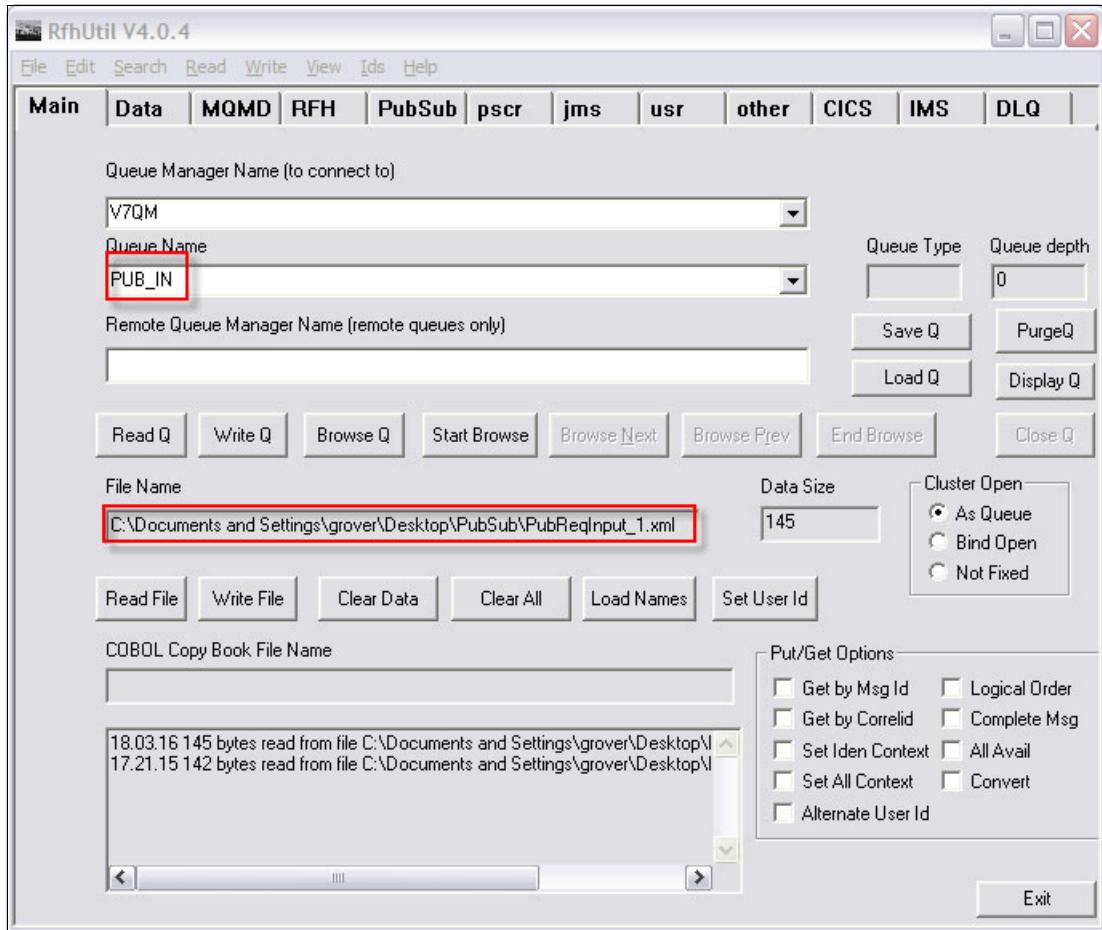


Figure 9-70 Send publication to the Publisher message flow

The Data tab of rfutil.exe shows the actual publication xml message in Figure 9-71 on page 415. Notice that the Temperature is 25, which is less than 30.

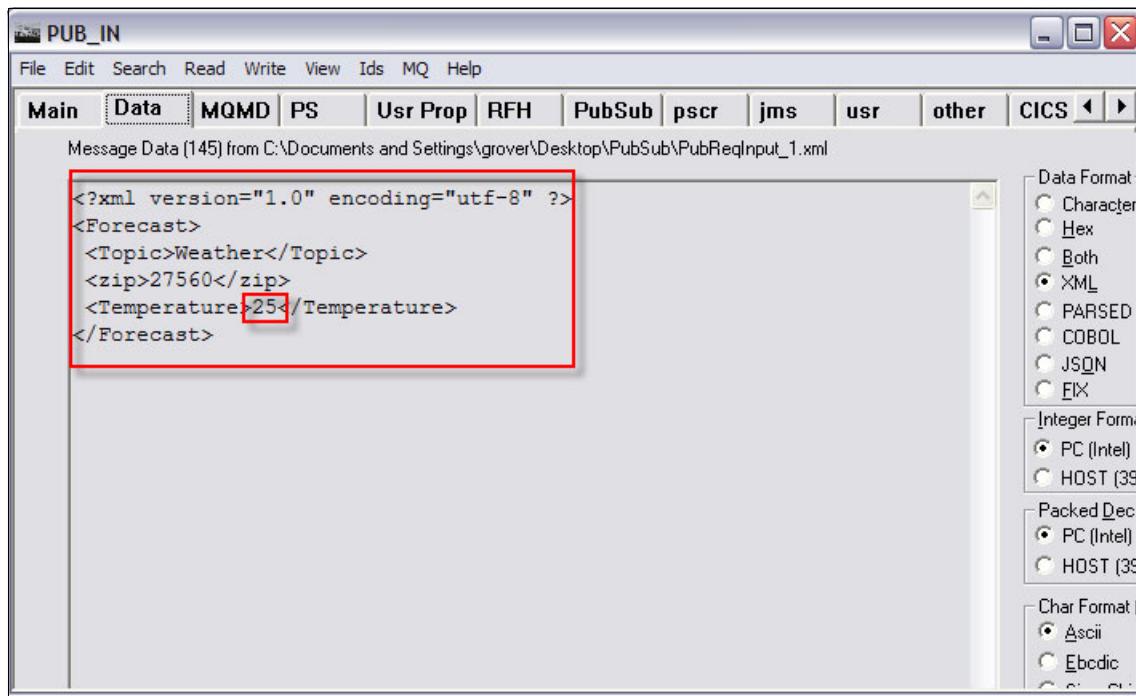


Figure 9-71 Publication data

Example 9-13 shows the actual publication xml message that is used for this test. Notice that the Temperature is 25, which is less than 30.

Example 9-13 Publication message

```
<?xml version="1.0" encoding="utf-8" ?>
<Forecast>
  <Topic>Weather</Topic>
  <zip>27560</zip>
  <Temperature>25</Temperature>
</Forecast>
```

13. As we put the message on the queue PUB_IN, the filter is evaluated to False by the broker, and the message is not received by the subscriber queue OUT, as shown in Figure 9-72 on page 416.

The screenshot shows the WebSphere MQ V7.0 interface. On the left, the 'Queue Managers' section lists 'V61QM' and 'V7QM'. Under 'V7QM', there are 'V7BK' and 'Queues' (which is highlighted with a red box). Other sections include 'Topics', 'Subscriptions', 'Advanced', 'Queue Manager Clusters', 'JMS Administered Objects', and 'Service Definition Repositories'. On the right, a table titled 'Filter: Standard for Queues' lists the following queues:

Queue name	Open input count	Current queue depth
EGL.TRADE.SUBSCRIPTION.Q	0	0
FAIL	0	0
IN	0	0
IN1	0	0
IN2	0	0
NOMATCH	0	0
OUT	0	0
OUT2	0	0

Figure 9-72 Subscriber queue did not get the message

14. Now, put another message, PubReqInput.xml, as shown in Figure 9-73.
Notice the Temperature in this message is 55, which is greater than 30.

The screenshot shows the PUB_IN tool window. The title bar says 'PUB_IN'. The menu bar includes File, Edit, Search, Read, Write, View, Ids, MQ, Help. The tab bar shows Main, Data, MQMD, PS, Usr Prop, RFH, PubSub, pscr, jms, usr, other, CICS. The main pane displays 'Message Data (142) from C:\Documents and Settings\grover\Desktop\PubSub\PubReqInput.xml'. The XML content is:

```

<?xml version="1.0" encoding="utf-8" ?>
<Forecast>
  <Topic>Weather</Topic>
  <zip>27613</zip>
  <Temperature>55</Temperature>
</Forecast>

```

A red box highlights the 'Temperature' value '55'. To the right, there are several configuration panels:

- Data Format: Radio buttons for Character, Hex, Both, XML (selected), PARSED, COBOL, JSON, FIX.
- Integer Format: Radio buttons for PC (Intel) (selected), HOST (39).
- Packed Dec: Radio buttons for PC (Intel) (selected), HOST (39).
- Char Format I: Radio buttons for Ascii (selected), Ebcdic.

Figure 9-73 New Publication data

15. Now, put another message, PubReqInput.xml, as shown in Example 9-14 on page 417. Notice the Temperature in this message is 55, which is greater than 30.

Example 9-14 PubReqInput.xml message

```
<?xml version="1.0" encoding="utf-8" ?>
<Forecast>
<Topic>Weather</Topic>
<zip>27613</zip>
<Temperature>55</Temperature>
</Forecast>
```

16. The broker evaluates the filter against the message to True and successfully processes this message and delivers it to the destination subscriber queue, as shown in Figure 9-74.

The screenshot shows the WebSphere MQ V7.0 Administration Console. On the left, there is a tree view of the queue manager objects. A red box highlights the 'Queues' node under 'V7QM'. On the right, there is a table titled 'Queue name' with columns for 'Open input count' and 'Current queue depth'. The table lists several queues: EGL.TRADE.SUBSCRIPTION.Q, FAIL, IN, IN1, IN2, NOMATCH, and OUT. The 'OUT' queue has a current queue depth of 1, indicated by a red box around the row.

Queue name	Open input count	Current queue depth
EGL.TRADE.SUBSCRIPTION.Q	0	0
FAIL	0	0
IN	0	0
IN1	0	0
IN2	0	0
NOMATCH	0	0
OUT	0	1

Figure 9-74

The message broker evaluated the filter to false for the first message and did not deliver it to the destination subscriber queue. Whereas, the filter was evaluated to true for the second message, and the message was delivered successfully.

9.5.2 Publish/subscribe using Selector in WebSphere MQ only

In this example, we demonstrate how publish/subscribe works when filtering is performed by WebSphere MQ alone. WebSphere MQ V7, itself, supports limited filtering ability on the message header and message properties. MQ checks for the subscriptions with the filter on message header or properties. After MQ makes sure that it can handle the filtering by itself, it does not send the publication messages to the message broker for any filter evaluation. Therefore, it does not make any difference if the Execution Groups are configured for content based filtering or not.

For simplicity, we can use the same message flow project that we demonstrated in the previous examples. The subscriber application registers a subscription on topic Weather with a filter of Root.MQMD.Priority > 1, which indicates that it only wants to see high priority weather report messages with the message priority that is greater than 1; therefore, when a publication arrives on the broker with the message priority that is greater than 1, the filter is evaluated to true by the queue manager, and the subscriber queue gets the message. However, if a publication

arrives on the broker and the message priority is 0 (which is less than 1), the filter is evaluated to false by the queue manager and the subscriber queue does not receive any messages. See Appendix A, “Additional material” on page 533 for a sample Project Interchange V7_flows_MQ.zip that contains the message flow V7_publish_MQ.msgflow.

Although, it is not required to have message flows when publishing or subscribing with content-based filtering, in this scenario, we use the message flow V7_publish.msgflow:

1. In the Message Broker Toolkit, create a message flow V7_publish.msgflow. Figure 9-75 shows the message flow with the nodes.

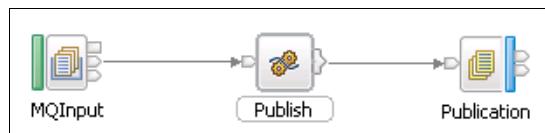


Figure 9-75 Message flow with nodes

The MQInput node is configured to use the Queue Name as PUB_IN and Input Message Parsing → Message Domain as XMLNSC. All of the other parameters are set to default.

Figure 9-76 on page 419 shows the compute node Publish references the ESQL.

```

CREATE COMPUTE MODULE V7_publish_Compute
  CREATE FUNCTION Main() RETURNS BOOLEAN
  BEGIN
    CALL CopyMessageHeaders();
    -- CALL CopyEntireMessage();
    -- Set up the Publication
    SET OutputRoot.MQRFH2.psc.Command = 'Publish';
    SET OutputRoot.MQRFH2.psc.PubOpt = 'Local';
    SET OutputRoot.MQRFH2.psc.Topic = InputRoot.XMLNSC.Forecast.Topic;
    SET OutputRoot.XMLNSC.psc.zip = InputRoot.XMLNSC.Forecast.zip;
    RETURN TRUE;
  END;

  CREATE PROCEDURE CopyMessageHeaders() BEGIN
    DECLARE I INTEGER 1;
    DECLARE J INTEGER;
    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;

```

Figure 9-76 Publisher flow ESQL

The compute node Publish references the ESQL shown in Example 9-15.

Example 9-15 ESQL code that compute node Publish references

```

CREATE COMPUTE MODULE V7_publish_MQ_Compute
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
  CALL CopyMessageHeaders();
  -- CALL CopyEntireMessage();
  -- Set up the Publication
  SET OutputRoot.MQRFH2.psc.Command = 'Publish';
  SET OutputRoot.MQRFH2.psc.PubOpt = 'Local';
  SET OutputRoot.MQRFH2.psc.Topic = InputRoot.XMLNSC.Forecast.Topic;
  SET OutputRoot.MQRFH2.psc.zip = InputRoot.XMLNSC.Forecast.zip;
  RETURN TRUE;
END;

CREATE PROCEDURE CopyMessageHeaders() BEGIN
  DECLARE I INTEGER 1;
  DECLARE J INTEGER;
  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;

```

2. For demonstration purposes only, we disable the Content Based Filtering on the Execution group PubSub from **WebSphere MQ Explorer** → **Brokers** → **V7BK** → **PubSub** → **Properties** → **ContentBasedFiltering**, as shown in Figure 9-77.

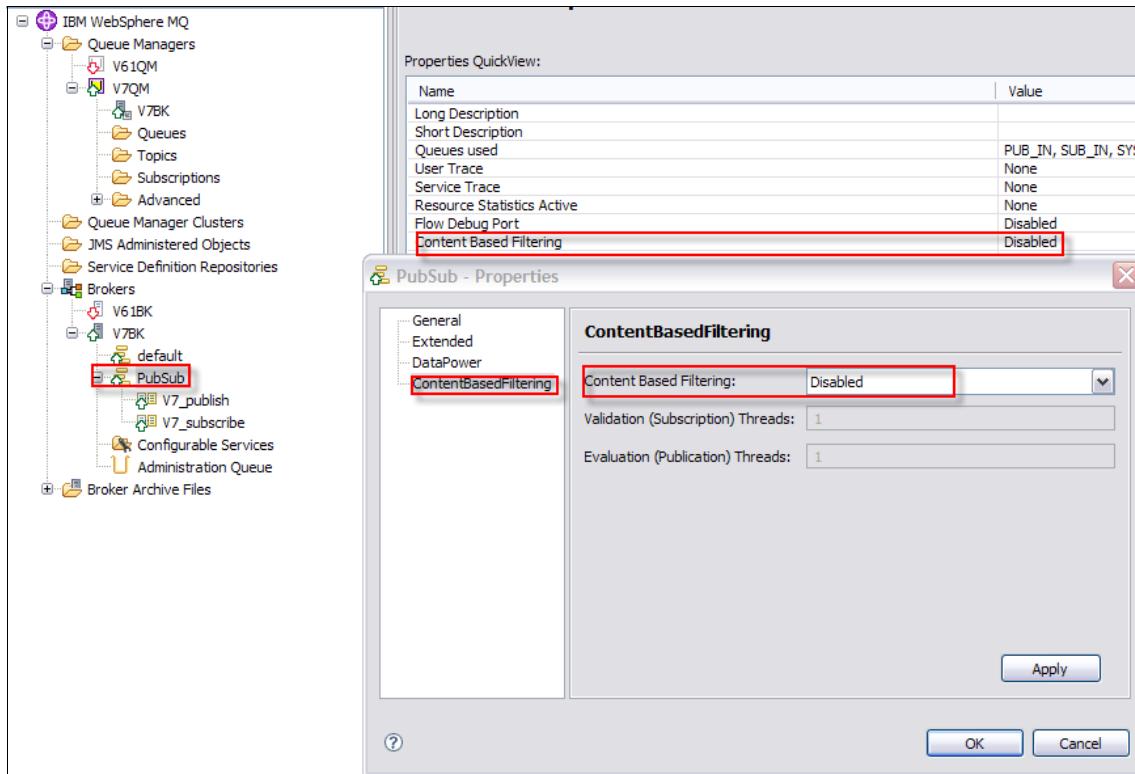


Figure 9-77 Disable the ContentBasedFiltering on Execution Group

3. Stop the broker using the command **mqsistop V7BK**, and restart it back with the command **mqsistart V7BK**.

4. Register a subscription on topic Weather with the filter Root.MQMD.Priority>1 using rfhutil.exe, as shown in Figure 9-78.

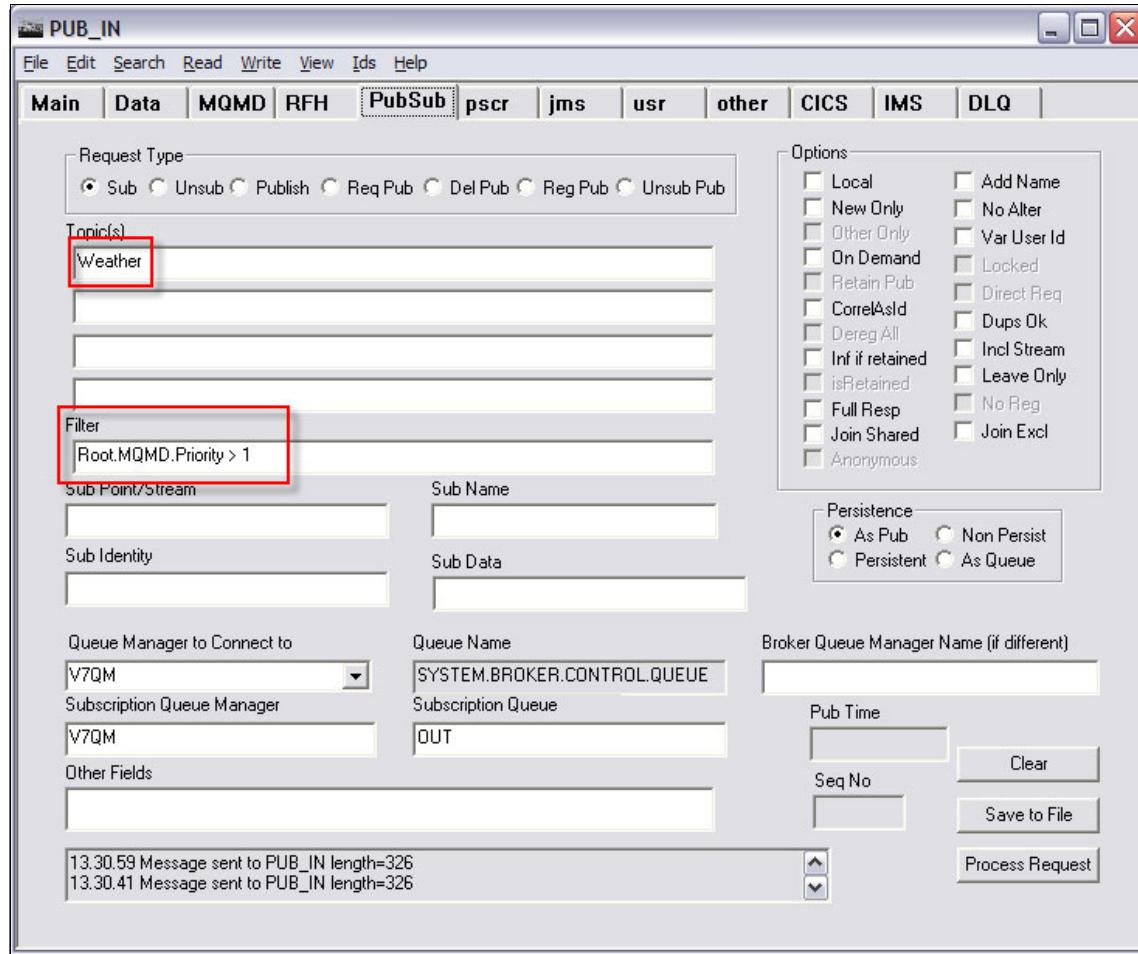


Figure 9-78 Subscription Registration with filter

5. The subscription is now registered and can be viewed in the **WebSphere MQ Explorer** → **Queue Managers** → **V7QM** → **Subscriptions** pane, as shown in Figure 9-79.

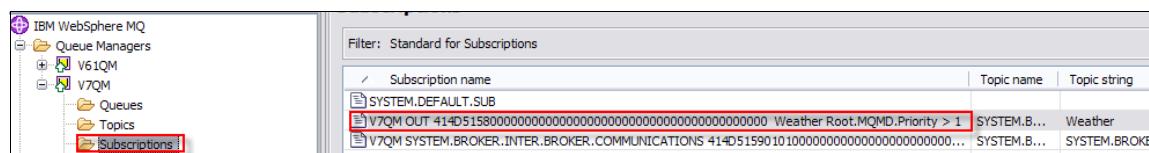


Figure 9-79 Subscription with filter

- To publish the message to the queue PUB_IN, we again start rfhutil.exe, and adjust the Priority of the message to 0 from the **MQMD** tab, as shown in Figure 9-80.

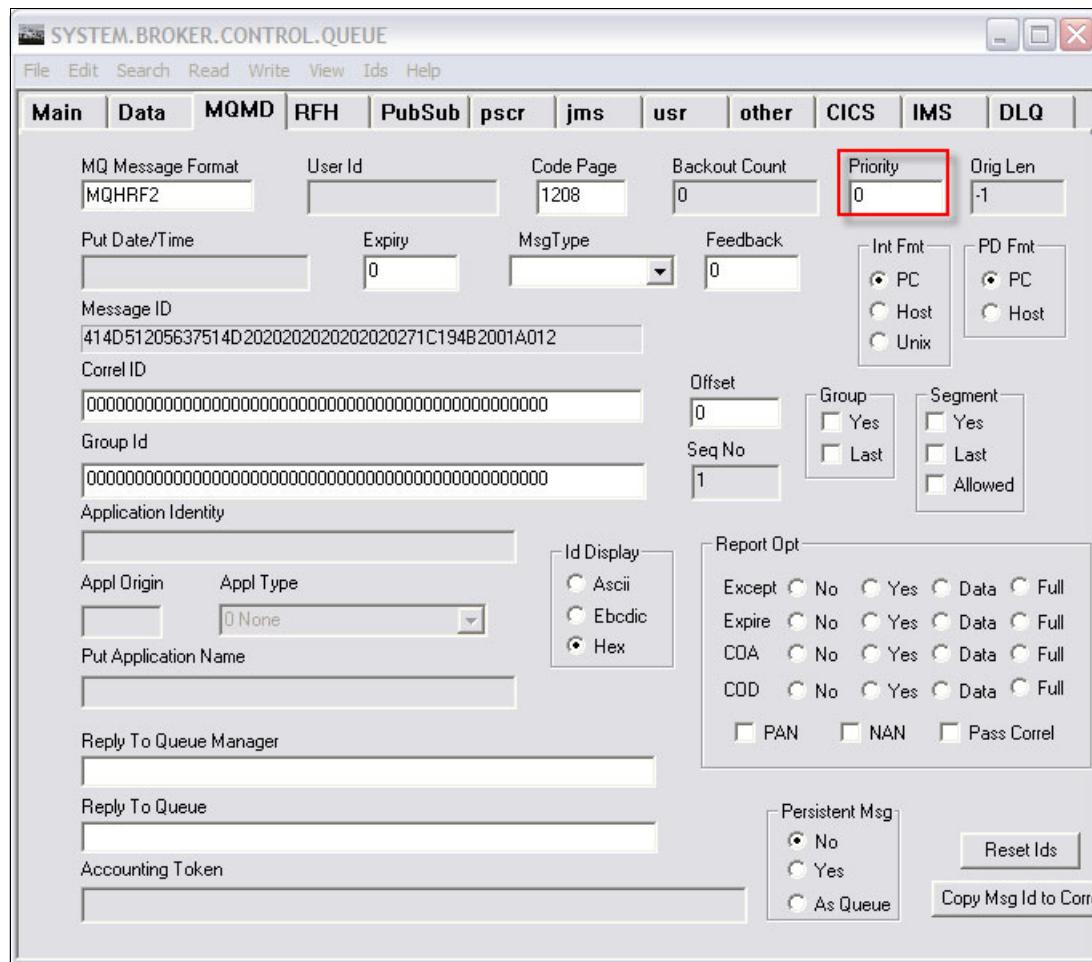


Figure 9-80 Message priority

- Submit the message to the queue PUB_IN, as shown in Figure 9-81 on page 423.

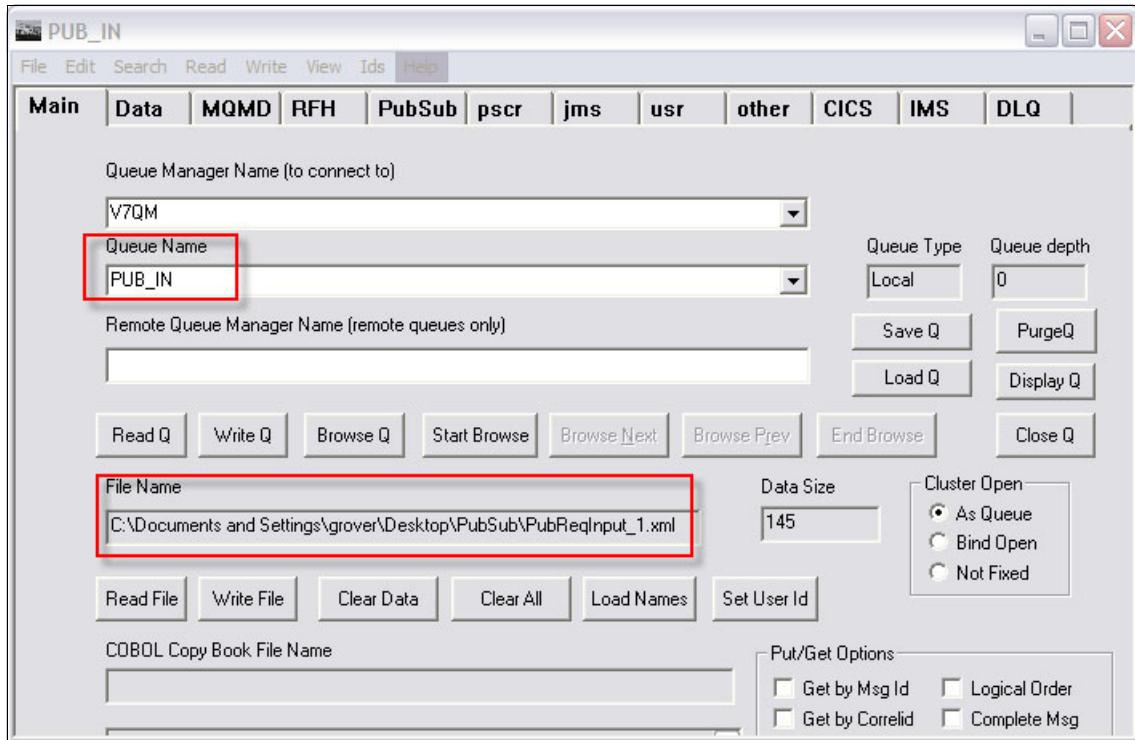


Figure 9-81 Publish the message

8. The filter evaluates to False because the MQMD.Priority of the output message is less than 1, so the message is not delivered to the destination subscriber queue OUT, as shown in Figure 9-82.

Queue Managers		Filter: Standard for Queues				
		Queue name	Queue type	Open input count	Open output count	Current queue depth
	V7QM	OUT	Local	0	0	0
		OUT2	Local	0	0	0

Figure 9-82 Current depth on the subscriber queue

9. Next, start the rfhutil.exe, and publish a message to the queue PUB_IN with message Priority 2 (which is greater than 1), as shown in Figure 9-83 on page 424.

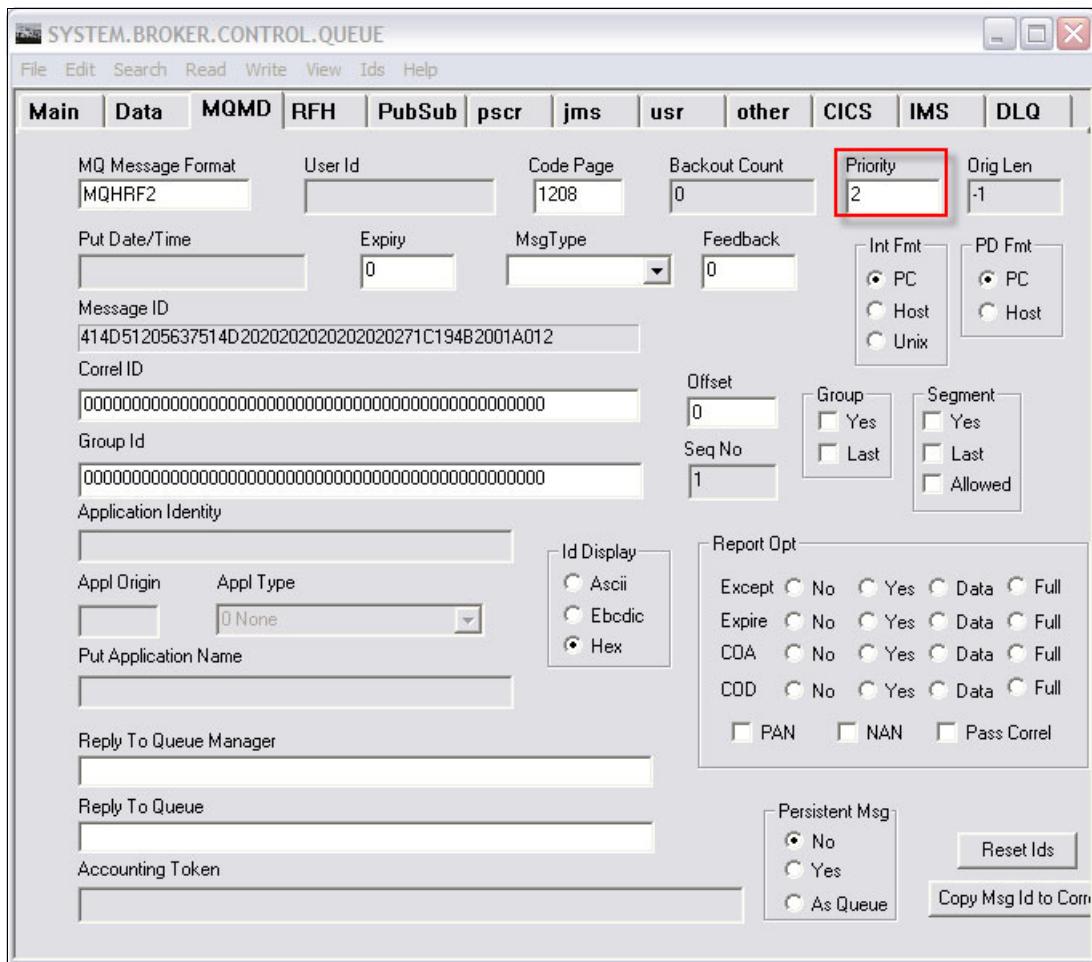


Figure 9-83 Message Priority

10. This time the filter evaluates to True because the message Priority is greater than 1 now, as shown in Figure 9-84.

Queue Managers		Filter: Standard for Queues				
		Queue name	Queue type	Open input count	Open output count	Current queue dep
	V61QM	/ OUT	Local	0	1	1
	V7QM	/ OUT2	Local	0	0	0

Figure 9-84 Current depth on the subscriber queue

9.6 Conclusion

WebSphere Message Broker V7 and WebSphere MQ V7.0.1 together provide a comprehensive publish/subscribe facility. They enable you to build a topic-based publish/subscribe infrastructure.

WebSphere MQ adds the new Publish Subscribe capabilities because it hosts the Publish/Subscribe engine. The queue managers use this publish/subscribe engine to receive messages from publishers and subscription requests from the subscribers.

WebSphere Message Broker incrementally extends publish/subscribe to the advanced level with the enhanced publication node. The publication node uses the underlying publish/subscribe engine in WebSphere MQ. The Publication node was enhanced and contains the new Out and NoMatch (no matching subscriptions) terminals for further propagation of messages.

Message Broker enhances publish/subscribe further by enabling the subscribers to perform content-based filtering on the messages.

Message Broker supports a wide range of ESQL expressions in the subscription filter. There is a straight forward publish/subscribe migration process from the previous versions of Message Broker and Event Broker. The migration occurs in three phases:

- ▶ Rehearsal phase: This phase creates a migration log, reporting any errors but does not modify any configurations. It is typically used to observe the results of the real migration.
- ▶ Initial phase: This phase migrates the topic objects to the queue manager, based on the Access Control List entries that are defined in the message broker. It also produces a file that contains the security commands to set up the security environment on the queue manager that is equivalent to the one that existed in the broker.
- ▶ Completion phase: This phase retrieves the existing publish/subscribe definitions from the message broker and uses them to create equivalent definitions in the publish/subscribe component of the queue manager.

The SYSTEM.BROKER.* queues are still used for administration purposes in Message Broker V7, as in the previous versions. The security model is now implemented by WebSphere MQ. It can either be queue-based and topic-based, so the administrators can create ACLs for each restricted topic.

The WebSphere Message Broker and WebSphere MQ publish/subscribe implementation is highly scalable. Multiple queue managers that are associated

with their brokers can be connected in tightly-coupled clusters to enhance the performance.

Publish/subscribe message: For more information about publish/subscribe messaging in WebSphere MQ V7.0.1 and WebSphere Message Broker V7, refer to:

<http://www.publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp>

<http://www.publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/index.jsp>



Scenario: Monitoring and WebSphere Business Monitor support

In the scenario, we demonstrate:

- ▶ “Business Activity Monitoring” on page 428
- ▶ “Connectivity” on page 432
- ▶ “Monitoring events” on page 459
- ▶ “Command-line configuration” on page 466
- ▶ “Monitor model” on page 469

10.1 Introduction to the products

As an advanced ESB and inter connectivity solution, WebSphere Message Broker sits in a unique position to support monitoring requirements. It typically operates in the middle of transactional paths. It has visibility of data in any format and can handle high-performance messaging. WebSphere Message Broker allows the generation of events throughout the execution instance of a message flow. The events can be used to monitor and analyze aspects of the process and enable capture and audit capabilities in an ESB.

WebSphere Business Monitor is a business activity monitoring (BAM) solution that can monitor and measure processes and applications across a heterogeneous infrastructure. Based on events that are generated from numerous disparate sources, the WebSphere Business Monitor solution enables complex event correlation, key performance indicator (KPIs) measurements, and predictive analytics in a user-friendly, consumable manner. You can use this real-time, end-to-end view of the business process to track, manage, and operate your business based on quantitative measures.

The combination of these two products enhances the ability to deliver an enterprise level ESB supporting a Business Activity Monitoring (BAM) solution.

10.1.1 Business Activity Monitoring

Business Activity Monitoring (BAM) is a enterprise solution that gives you the capability to see real time, end-to-end information that is associated with one or more business processes. Its purpose is to monitor the business events that occur throughout the enterprise and provide quantifiable evidence that can be used to improve and optimize the business processes.

Fundamental concepts that apply to BAM solutions include:

- ▶ *Business Event* indicates the occurrence of an event that is pertinent to the operation of a business activity.
- ▶ *Business Activity* is a specific function that is required to fulfill a specific business need.
- ▶ *Metric* is a measurement of a property that can be used to monitor business operations.
- ▶ *Key Performance Indicators* are measurements that are used to quantify business objectives.

An important distinction between a BAM and other IT-related monitoring solutions is that BAM solutions monitor business-focused events. IT monitoring provides information that describes the availability and performance of a specific IT resource. In a business event, the pertinent data is usually embedded in the event payload (for example, the order quantity, price, duration, and so on); whereas, in an IT event, often times, the event itself is enough to describe the situation (for example, system unavailable).

10.1.2 System architecture of the scenario

In this scenario we use WebSphere Business Monitor as the Business Activity Monitoring solution. WebSphere Message Broker and WebSphere Process Server act as the event producers hosting the business process components in the event-based monitoring solution. Architecturally these components are loosely-coupled using concepts, such as asynchronous messaging, pub/sub topics, subscriptions, and a Common Event Infrastructure (CEI) to support a high-available, extensible solution. Figure 10-1 demonstrates the different architectural touch points that encompass the overall solution.

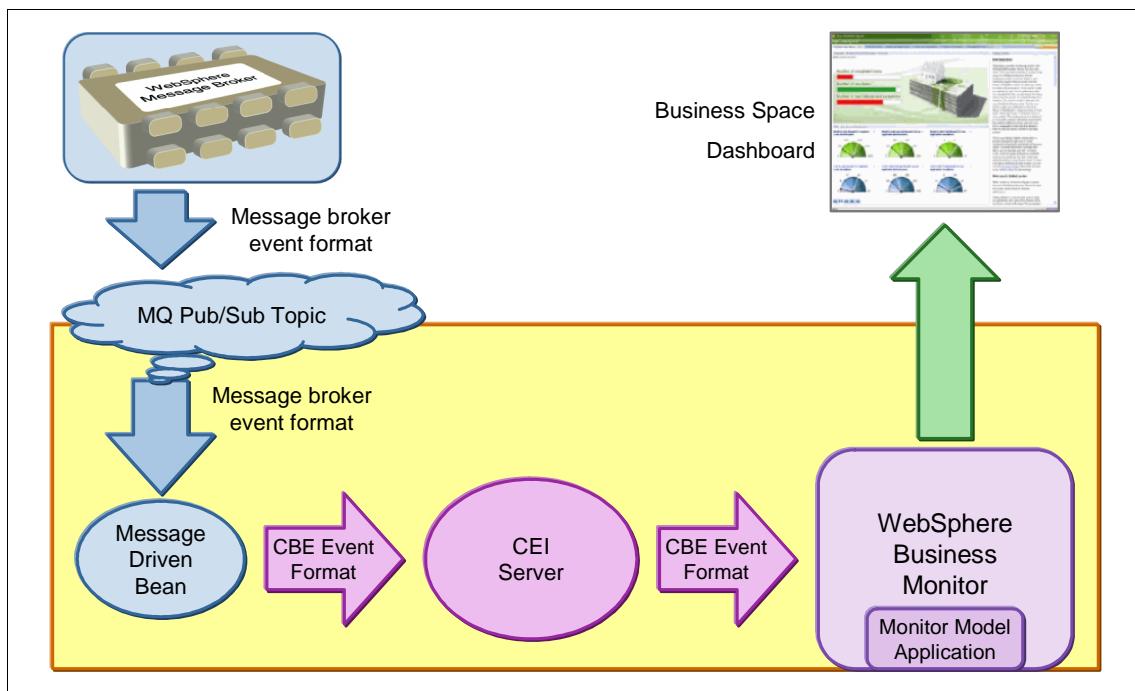


Figure 10-1 System architecture

Message Broker is configured to emit events. The event is emitted in an XML format that is compatible with the Common Base Event (CBE) specification and allows integration with other monitoring applications. The format also enables capture and logging capabilities of the entire message for auditability. The event contains information that is used for identification, sequencing, correlation, and application data. The events are published to a topic that multiple consumers can subscribe to.

The Message-Driven bean (MDB) is provided as a part of the WebSphere Business Monitor sample that is available in the WebSphere Message Broker Samples Gallery. This bean is hosted on a WebSphere Application Server and wraps the message broker event with a Common Base Event wrapper and then submits it to the CEI server.

Common Event Infrastructure is the IBM implementation of an embeddable solution for providing a unified event infrastructure. It enables a common integration point for storage and publication of raw events between multiple disparate systems. The events are represented by using the Common Base Event specification.

WebSphere Business Monitor runs within a WebSphere Application Server instance. It executes the monitor models, which extract information from business events, stores the information for analysis, and responds to business situations. The monitor models give context and correlation to multiple events and enable the capabilities surrounding the metrics, keys, counters, stopwatches, triggers, events, and Key Performance Indicators (KPI).

Dashboards are the primary user interface for access to the WebSphere Business Monitor information. The dashboard is made up of Web 2.0-based components that use a set of REST services to dynamically update the monitored events and information.

10.2 Scenario environment

The scenario was developed in an environment with the following components:

- ▶ WebSphere Message Broker 7.0.0
- ▶ WebSphere Message Broker Toolkit 7.0.0
- ▶ WebSphere Business Monitor 7.0.0
- ▶ WebSphere Process Server 7.0.0
- ▶ WebSphere Integration Developer 7.0.0
- ▶ WebSphere MQ 7.0.1

The broker that we used during this scenario was created using the *Create Default Broker* configuration that is available in the WebSphere Message Broker Toolkit.

The WebSphere Business Monitor and WebSphere Process Server that we use throughout this scenario is the unit test environment (UTE) that was created by the WebSphere Integration Developer installation.

10.3 Scenario

The business scenario that we discuss in this chapter relates to a garden supply company that is improving its order handling process. The new process should include shipping capabilities and the ability to monitor key aspects of its new process. Figure 10-2 shows the basic flow of the scenario.

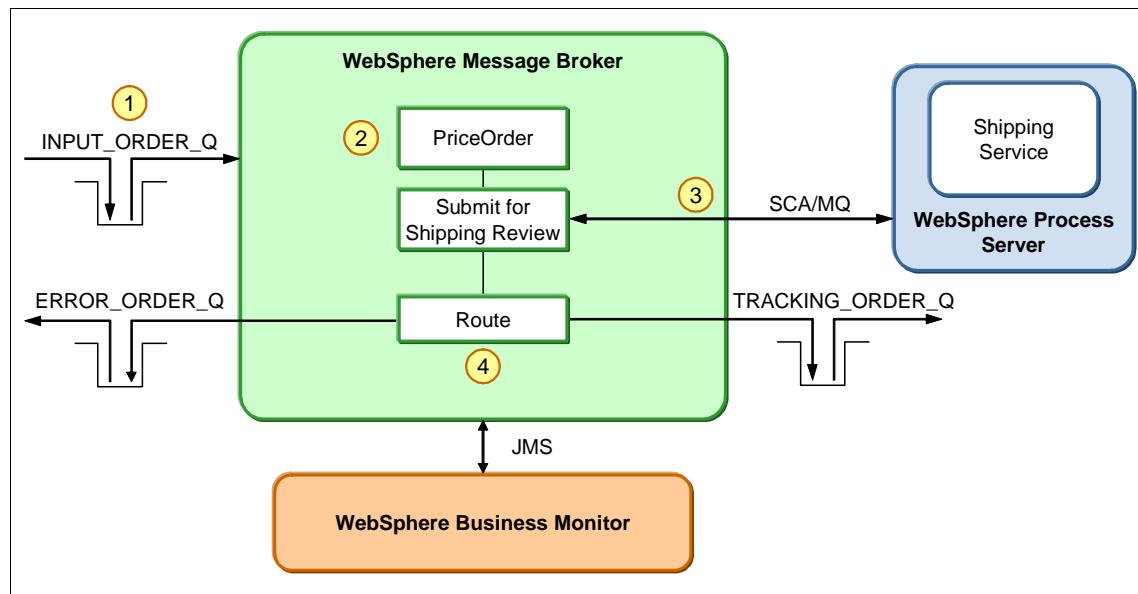


Figure 10-2 Scenario Overview

The highlights of the scenario, numbered in Figure 10-2, are:

1. Orders from two different kinds of customers, Gold and Regular, are received on a WebSphere MQ queue (INPUT_ORDER_Q).
2. The orders are priced based on customer type. GOLD customers are given a 15% discount on their order.

If the invoice total of the order is greater than \$1000.00, it should be considered high priority.

3. The priced order is then submitted to a shipment review process that a WebSphere Process Server business process hosts. This business process reviews the order to confirm that it is not too large (< 200 items) to ship and to assign a shipping tracking ID.

Processing time for the review depends on the customer type. The standard processing time is from 2-60 seconds. However, for GOLD customers processing time is less than 30 seconds.

4. If the shipping review is successful, the completed order is forwarded to a tracking system using a queue (TRACKING_ORDER_Q); otherwise, the priced order is returned to the submitter with an appropriate error using the ReplyToQ that is provided in the MQ message header of the original message (ERROR_ORDER_Q).

To monitor the new business process, WebSphere Business Monitor will monitor the following aspects:

- ▶ Average order processing time
- ▶ Average high priority order processing time
- ▶ Average shipping time
- ▶ Maximum order shipping time for GOLD customers.
- ▶ Ability to see failed order details.

10.4 Connectivity

In this section, we demonstrate the connectivity steps that are necessary to enable communication between a WebSphere Message Broker broker and WebSphere Business Monitor instance.

10.4.1 WebSphere MQ/JMS

WebSphere Message Broker leverages a pub/sub model to disseminate events to consumers. Events are published to a topic, which multiple subscribers can read. Example 10-1 shows the topic format. The hierarchical structure of the topic format and the ability to use wildcards in the topic subscription allow subscribers to filter the events they want.

Example 10-1 Topic format

\$SYS/Broker/<brokerName>/Monitoring/<executionGroupName>/<flowName>

In this scenario, WebSphere Business Monitor consumes the events using a JMS connection. Some preparation is required to configure MQ to be the JMS provider. The JMS topic that is used for event publication is created on the WebSphere Message Broker queue manager, MB7QMGR.

Using the WebSphere MQ Explorer:

1. Create the TCP listener called MB7QMGR in WebSphere MQ Explorer by selecting **Queue Managers** → **MB7QMGR** → **Advanced** → **Listeners**. Right-click **Listeners**, and select **New** → **TCP Listener**.

Figure 10-3 shows the Create TCP Listener wizard.

/	Listener name	Control	Listener status	Xmit protocol	Port	IP address	Backlog	TP name	Adapter	Local name
	MB7QMGR	Queue Manager	Running	TCP	2414		0			

Figure 10-3 TCP Listener

2. Start the MB7QMGR TCP listener.
3. Create a Server Connection channel called MONITOR.CHL.

You can define new channels by selecting **Queue Managers** → **MB7QMGR** → **Advanced** → **Channels**. Right-click **Channels**, and select **New** → **Server-connection Channel**.

Figure 10-4 shows the Create Server-Connection Channel wizard.

/	Channel ...	Channel type	Overall channel status	Conn name	Xmit protocol	Transmission queue	Queue
	MONITOR.CHL	Server-connection	Running		TCP		

Figure 10-4 Server-connection Channel

4. Run the WebSphere MQ-supplied MQJMS_PSQ.mqsc file to create the required JMS queues on the WebSphere Message Broker queue manager, MB7QMGR. This MQJMS_PSQ.mqsc file is contained in the WebSphere MQ product install directory in the java/bin subdirectory. Run the **runmqsc** command, as shown in Example 10-2.

Example 10-2 runmqsc command

```
runmqsc MB7QMGR < MQJMS_PSQ.mqsc
```

Validate that the message No commands have a syntax error. is provided, as shown in Example 10-3 on page 434.

Example 10-3 runmqsc output

No commands have a syntax error.
All valid MQSC commands were processed.

10.4.2 Message-Driven Bean

WebSphere Message Broker monitoring events are published in XML format. A message driven bean (MDB) is provided to transform the event in to a Common Base Event for submission to the Common Event Infrastructure (CSI).

In this scenario, it is assumed that the MDB is installed on the same WebSphere Application Server instance that the WebSphere Business Monitor server is installed on.

The following steps describe the installation of the message driven bean into the WebSphere Application Server instance:

1. Using a Web browser, access the WebSphere Business Monitor Administration Console:
<https://localhost:9043/ibm/console/login.do>
2. Create a new topic connection factory, as shown in Figure 10-5 on page 435. The connection factory defines the connection properties that are used to access the queue manager where the topic resides. The JNDI name for the new connection factory is jms/topicConn.
Topic connection factories can be created or managed in **Resources → JMS → Topic connection factories**.

General Properties

Administration

Scope
Node=qnode,Server=server1

Provider
WebSphere MQ messaging provider

* **Name**
topicConn

* **JNDI name**
jms/topicConn

Description

Connection

Queue manager
MB7QMGR

Transport
Client

* **Hostname**
localhost

Port
2414

Server connection channel
MONITOR.CHL

Use SSL to secure communication with WebSphere MQ

Centrally managed

Specific configuration

SSL configuration
NodeDefaultSSLSettings

Figure 10-5 Topic connection factory

3. Create a new topic, as shown in Figure 10-6. The topic provides a definition for the MDB to use to access the published events. The JNDI name for the new topic is `jms/topicName`.

Topic name: The Topic name is dependent on the deployment details for your system. In this scenario, the Topic Name is `$SYS/Broker/MB7BROKER/Monitoring/OrderServiceExecGroup/OrderServiceFlow`. However you can modify it based on your broker name, execution group, message flow, and subscription requirements. Wild cards can also be used to filter the events that are received.

You can create or arrange Topics in **Resources → JMS → Topics**.

The screenshot shows the configuration interface for a JMS topic. It consists of two main sections: 'General Properties' at the top and 'WebSphere MQ topic' below it.

General Properties:

- Administration:**
 - Scope:** Node=qnode,Server=server1
 - Provider:** WebSphere MQ messaging provider
- * Name:** topicName
- * JNDI name:** jms/topicName
- Description:** (A large text area with scroll bars.)

WebSphere MQ topic:

- * Topic name:** \$SYS/Broker/MB7BROKER/Monitoring/OrderProcessingEG
- Broker durable subscription queue:** (Empty input field)
- Broker durable subscriber connection consumer queue:** (Empty input field)
- Broker publication queue:** As connection (A dropdown menu showing 'As connection' and 'From connection manager...')
- Broker publication queue manager:** (Empty input field)

Figure 10-6 Topic

4. Install the Event Emitter MDB:

The WebSphere Message Broker Event Emitter MDB: The WebSphere Message Broker Event Emitter MDB is provided as a part of the WebSphere Business Monitor sample that is available in the WebSphere Message Broker Toolkit Samples Gallery.

The sample, WebSphere Business Monitor sample, is in the WebSphere Message Broker Toolkit through **Help → Samples and Tutorials → WebSphere Message Broker Toolkit - Message Broker**. Locate the WebSphere Business Monitor sample and Import the sample. In the WebSphere Message Broker Toolkit, projects that are created by that sample there should be a WMBEventEmitterEAR.zip file that contains the Event Emitter MDB EAR file. In preparation to installing the MDB EAR, the file can be exported and unzipped on to a local filesystem.

- a. Navigate to **Applications → New Application → New Enterprise Application**, and select the WMBEventEmitterEAR.ear, as seen in Figure 10-7.

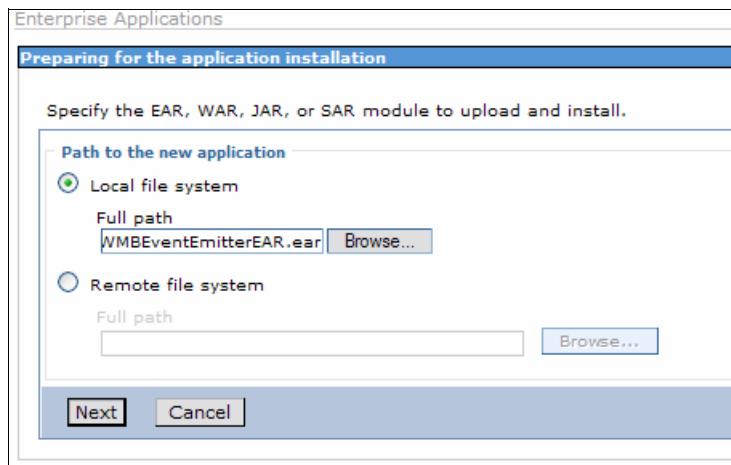


Figure 10-7 New Enterprise Application

- b. Click **Next** to continue.
- c. Select **Detailed - show all installation options and parameters**.
- d. Proceed until you get to the Bind listeners for message driven bean section. Write down the name of the listener port, as shown in Figure 10-8 on page 438.

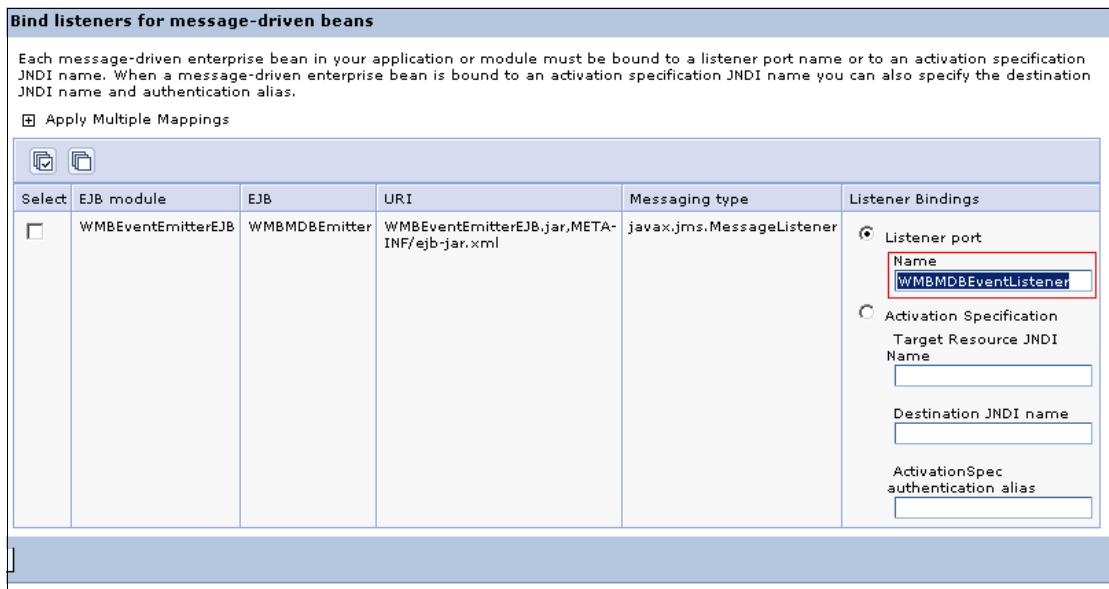


Figure 10-8 Listener port bindings

Optional: If security is enabled on the Monitor server then complete the following steps; otherwise, skip to step g.

- e. Continue to the Map security roles to users or groups section. Map the eventEmitter role to the proper administrative user by selecting the eventEmitter role option, and click **Map Users**. Search for the administrative user, and move it to the Selected pane using the \Rightarrow icon. Click **Ok**.
- f. Continue to the Map RunAs roles to users section. Set the username and password on the **eventEmitter** role to the proper administrative. Click **Ok**.
- g. Continue throughout the rest of the installation menus accepting the default settings to confirm and save the successful installation.

5. Create a listening port for the Message-driven bean to listen for incoming events:
 - a. Navigate to **Servers** → **Server Types** → **WebSphere Application Servers** → **server1** → **Messaging** → **Message listener service** → **Listener Ports**.
 - b. Create a new Listener port, as shown in Figure 10-9. Set the Name to the value that you wrote down in step d on page 437 for the Listener Port. The Connection factory JNDI name and Destination JNDI name must correspond to those configured in steps 2 on page 434 and step 3 on page 436, respectively.

[Application servers](#) > [server1](#) > [Message listener service](#) > [Listener Ports](#) > WMBMDBEventListener

Use this page to configure listener ports upon which message-driven beans listen for messages. Each JMS connection factory and JMS destination that a message-driven bean, deployed against that port,

Configuration	Runtime
<p>General Properties</p> <p>* Name <input type="text" value="WMBMDBEventListener"/></p> <p>* Initial State <input type="button" value="Started"/></p> <p>Description <input type="text"/></p> <p>* Connection factory JNDI name <input type="text" value="jms/topicConn"/></p> <p>* Destination JNDI name <input type="text" value="jms/topicName"/></p> <p>Maximum sessions <input type="text" value="1"/></p> <p>Maximum retries <input type="text" value="0"/></p> <p>Maximum messages <input type="text" value="1"/></p> <p><input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/></p>	

Figure 10-9 Listener port creation

6. Save and restart the server.

7. Validate that the Listener Port is running, as shown in Figure 10-10.

The screenshot shows the 'Listener Ports' configuration page. At the top, there's a header with the path: Application servers > server1 > Message listener service > Listener Ports. Below the header, a message says: 'Use this page to configure listener ports upon which message-driven beans listen for messages. Each port specifies the JMS connection factory and JMS destination that a message-driven bean, deployed against that port, listens upon.' There are buttons for Preferences, Convert to activation specification, New, Delete, Start, and Stop. A toolbar with icons for copy, paste, delete, and search follows. A table lists resources: Name (WMBMDBEventListener), Description (empty), Connection factory JNDI name (jms/topicConn), Destination JNDI name (jms/topicName), and Status (green circle with plus). A note below the table says: 'You can administer the following resources:' and shows a total of 1 resource.

Figure 10-10 Listener port validation

10.5 WebSphere MQ

In this section, we create the MQ queues that the WebSphere Process Server business process and WebSphere Message Broker message flow used as a part of the scenario.

Using the WebSphere MQ Explorer:

1. On the WebSphere Message Broker queue manager, MB7QMGR, create the Local Queues that are listed in Table 10-1. To create a new Local Queue, navigate to **Queue Managers** → **MB7QMGR** → **Queues** → **New** → **Local Queue**. Enter the Name listed in Table 10-1, and click **Finish**.

Table 10-1 Queue names

Name:
INPUT_ORDER_Q
TRACKING_ORDER_Q
ERROR_ORDER_Q
SHIP.REQ
SHIP.RESP

10.6 WebSphere Process Server

In this section, we demonstrate the implementation of the shipping service described in 10.3, “Scenario” on page 431, as a business integration module deployed on WebSphere Process Server.

10.6.1 Business integration module

The business process is implemented in a business integration module called `ShippingServiceModule`. Figure 10-11 shows the assembly diagram for this module.

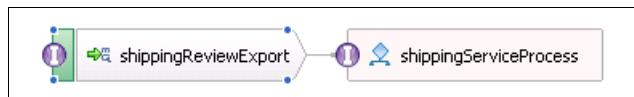


Figure 10-11 *ShippingServiceModule* assembly diagram

The assembly diagram consists of the following components:

- ▶ *ShippingReviewExport* is the export component that the `OrderService` message flow uses to access the business process. It has a `ShippingServiceInterface`.
- ▶ *ShippingServiceProcess* contains the business process implementation described in step 3 on page 432 of the 10.3, “Scenario” on page 431 section.

10.6.2 Business objects

A shared set of business objects are used in the `ShippingServiceModule` and the `OrderService` message flow that we discussed in 10.7.1, “`OrderService` message flow” on page 452. Figure 10-12 on page 442 shows the five Data Types.

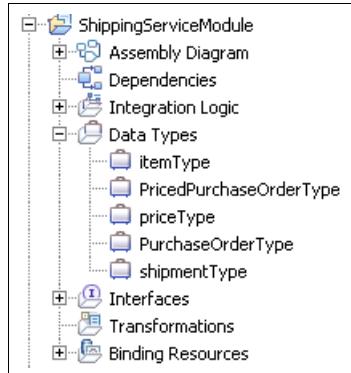


Figure 10-12 Data Types

Example 10-4 shows the XML schema that can be imported to generate the business objects that this module uses:

1. Save the XSD to a local file (ExtendedPOInstance.xsd) and import it in to WID module, ShippingServiceModule.
2. Navigate to **File** → **Import** → **Business Integration** → **WSDL and XSD**, and follow the prompts to import the local file.

Example 10-4 ExtendedPOInstance.xsd

```
<?xml version="1.0" encoding="UTF-8"?><xsd:schema
elementFormDefault="qualified"
targetNamespace="http://www.ibm.com/ProcessOrder"
xmlns:PO="http://www.ibm.com/ProcessOrder"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="purchaseOrder" type="PO:PurchaseOrderType"/>
<xsd:element name="purchaseOrderPriced"
type="PO:PricedPurchaseOrderType"/>

<xsd:complexType name="PurchaseOrderType">
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="purchaseOrderID"
type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="customerID"
type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="1" name="customerType"
type="xsd:string"/>
        <xsd:element name="items" type="PO:itemType"/>
    </xsd:sequence>
</xsd:complexType>
```

```

<xsd:complexType name="PricedPurchaseOrderType">
  <xsd:complexContent>
    <xsd:extension base="P0:PurchaseOrderType">
      <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="0" name="transactionId"
type="xsd:string"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="priceSummary"
type="P0:priceType"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="shippingSummary"
type="P0:shipmentType"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="itemType">
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="1" name="item">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="partNum" type="xsd:string"/>
          <xsd:element name="productName" type="xsd:string"/>
            <xsd:element name="quantity" type="xsd:integer"/>
            <xsd:element name="price" type="xsd:decimal"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="priceType">
  <xsd:sequence>
    <xsd:element name="itemTotal" type="xsd:decimal"/>
    <xsd:element name="itemCount" type="xsd:integer"/>
    <xsd:element name="deliveryCharges" type="xsd:decimal"/>
    <xsd:element name="discount" type="xsd:decimal"/>
    <xsd:element name="invoice" type="xsd:decimal"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="shipmentType">
  <xsd:choice>
    <xsd:element name="trackingId" type="xsd:string"/>
    <xsd:element name="shippingStatus" type="xsd:string"/>
  </xsd:choice>
</xsd:complexType>
</xsd:schema>

```

10.6.3 Interfaces

The module has one interface, `shippingServiceInterface`, shown in Figure 10-13. It is the interface to the business process. The interface has one request-response operation that sends a `PricedPurchaseOrderType` message and receives a modified `PricedPurchaseOrderType` message that contains shipping information.

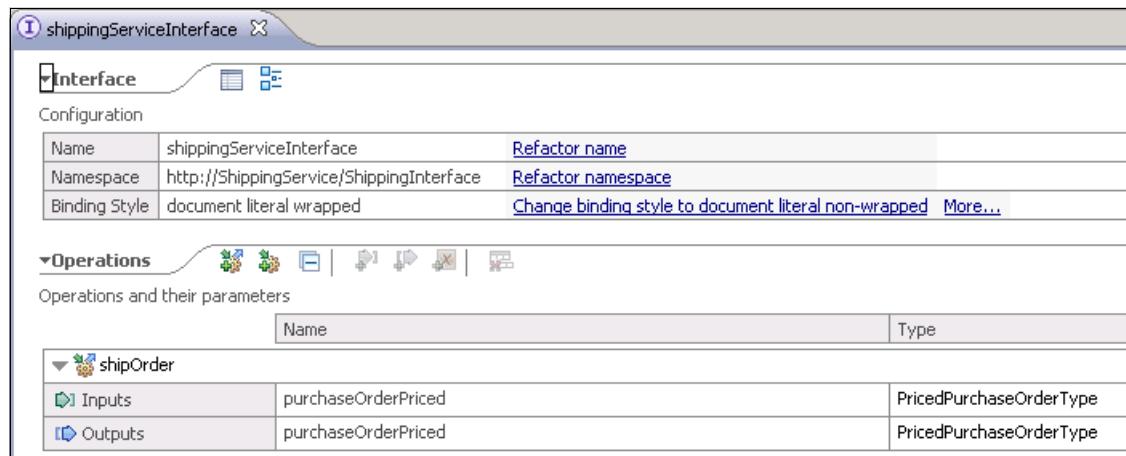


Figure 10-13 *ShippingServiceInterface interface specification*

10.6.4 Business process

The business process, `shippingServiceProcess`, performs the shipment review process that we described in the 10.3, “Scenario” on page 431. In the following sections, we describe the process settings for the steps that are shown in Figure 10-14.

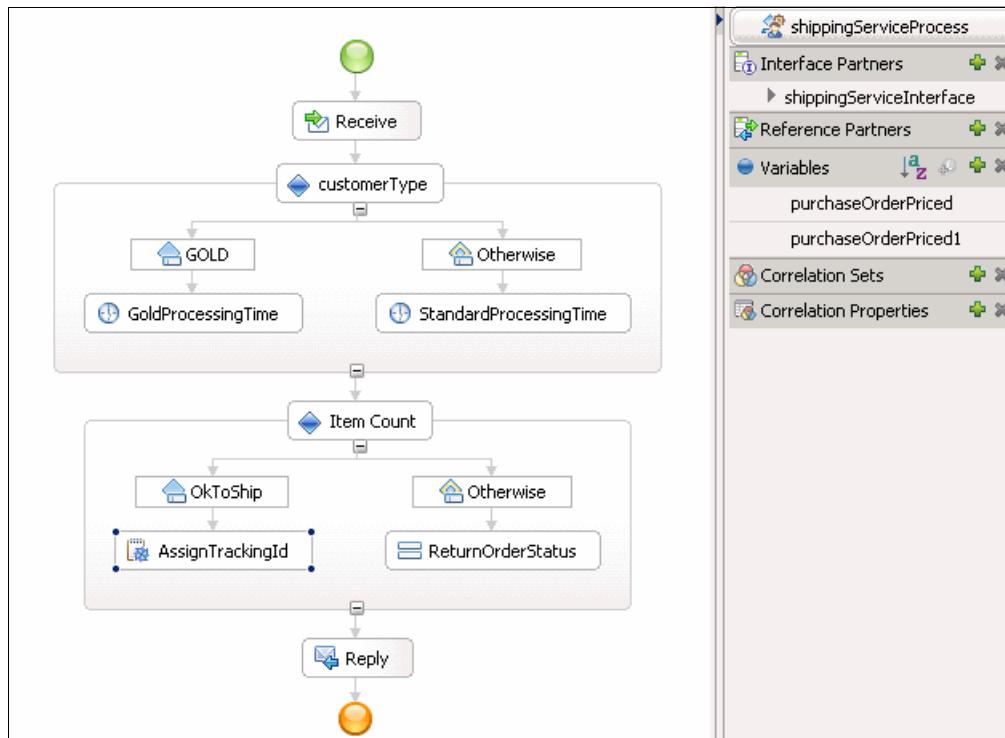


Figure 10-14 `shippingServiceProcess` Business Process

Business process properties

The business process properties are set to make it a long-running process due to the Wait step that is necessary to simulate processing time. See Figure 10-15.

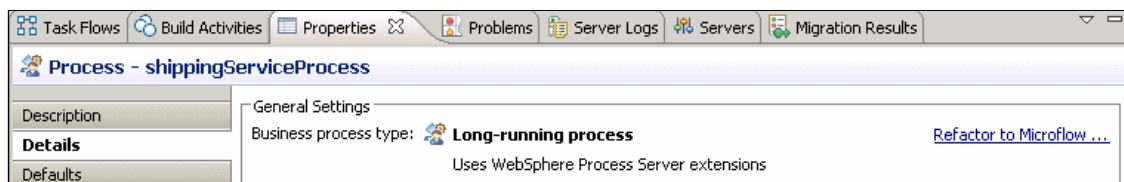


Figure 10-15 The `shippingServiceProcess` Business Process properties

Receive activity

The receive activity accepts the input to the business process. It is associated with the shippingServiceInterface, as shown in Figure 10-16.

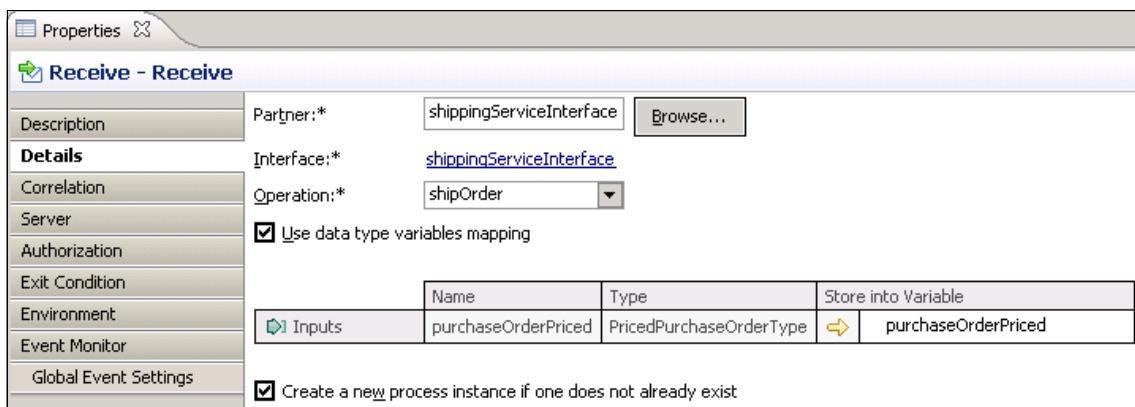


Figure 10-16 Receive activity properties

CustomerType: GOLD case activity

The customerType Choice structure determines the processing delay that is required based on the customer type. It has two cases: GOLD and Otherwise.

Processing time: As we discussed in 10.3, “Scenario” on page 431, standard processing time is from 2-60 seconds. GOLD customer processing time should be less than 30 seconds.

Figure 10-17 shows the GOLD case configuration.

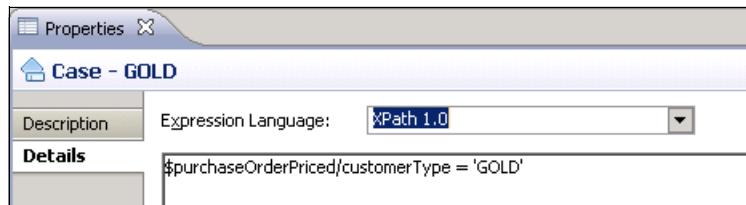


Figure 10-17 GOLD Case activity properties

GoldProcessTime Wait activity

The Wait step provides the gold customer delay associated with shipment review process. As shown in the configuration in Figure 10-18 on page 447, processing time for GOLD customers is between two and 30 seconds.

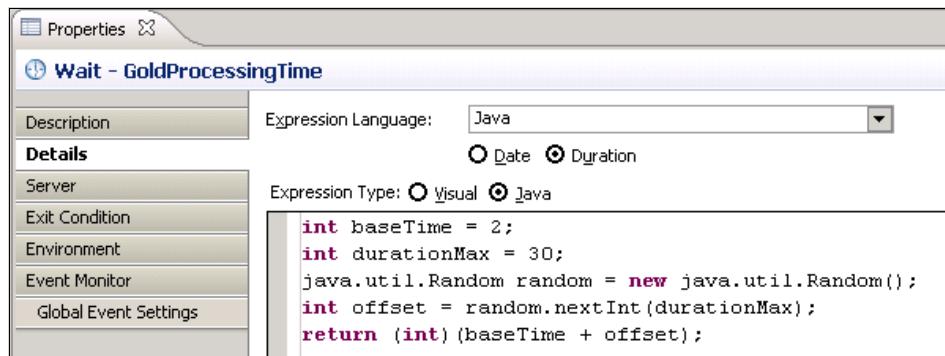


Figure 10-18 GOLD Processing Wait activity properties

Example 10-5 provides the code snippet that you can use to generate the delay.

Example 10-5 GOLD Processing Wait activity Java Snippet

```
// The Standard shipment processing time is from 2 to 60 seconds.
// The process generates a random duration to add to the base 2 seconds
// to simulate shipment processing time.
int baseTime = 2;
int durationMax = 30;
java.util.Random random = new java.util.Random();
int offset = random.nextInt(durationMax);
return (int)(baseTime + offset);
```

StandardProcessingTime Wait activity

The Wait step provides the standard delay that is associated with the shipment review process. As shown in the configuration in Figure 10-19, processing time for non-GOLD customers is between two and 60 seconds.

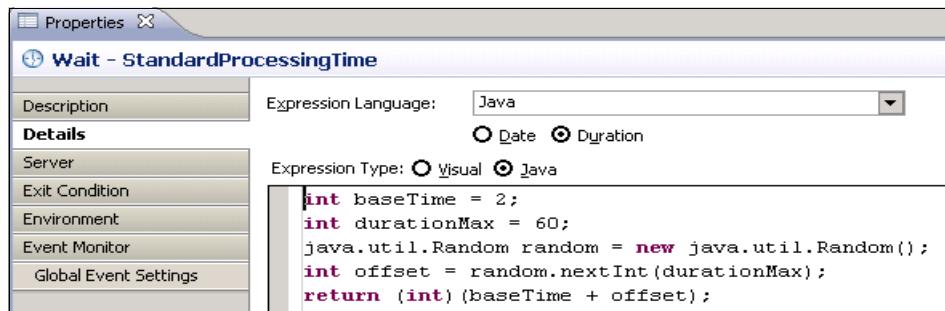


Figure 10-19 StandardProcessingWait activity properties

Example 10-6 provides the code snippet that you can use to generate the delay.

Example 10-6 StandardProcessingWait activity Java Snippet

```
// The Standard shipment processing time is from 2 to 60 seconds.  
// The process generates a random duration to add to the base 2 seconds  
// to simulate shipment processing time.  
  
int baseTime = 2;  
int durationMax = 60;  
java.util.Random random = new java.util.Random();  
int offset = random.nextInt(durationMax);  
return (int)(baseTime + offset);
```

ItemCount: OkToShip case activity

The itemCount Choice structure determines whether the order is too big to ship. It has two cases: OkToShip and Otherwise.

As we discussed in the 10.3, “Scenario” on page 431, the order is too big to ship if it exceeds a total of 200 items.

Figure 10-20 shows the OkToShip case configuration.

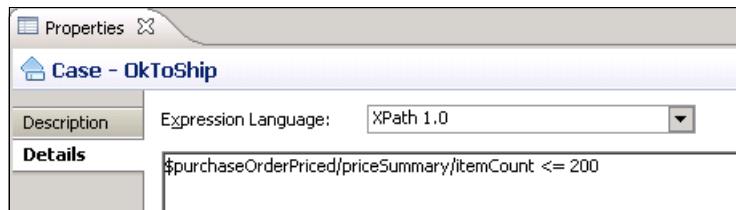


Figure 10-20 OkToShip Case activity properties

AssignTrackingId snippet activity

The AssignTrackingId snippet activity generates a unique tracking ID that is assigned to the shippingSummary/trackingId element of the response pricedPurchaseOrder message. Figure 10-21 on page 449 shows the java snippet details.

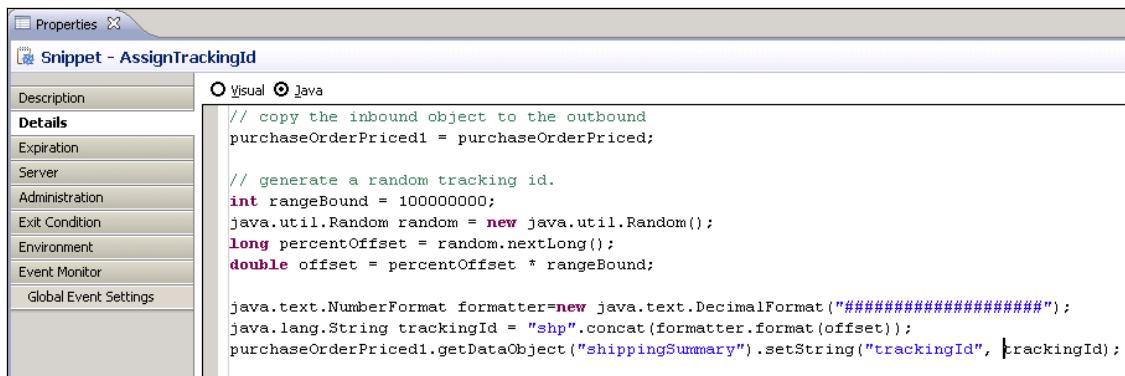


Figure 10-21 *AssignTrackingId* snippet properties

Example 10-7 provides the code snippet that you can use to generate the tracking ID.

Example 10-7 AssignTrackingId java snippet

```
// copy the inbound object to the outbound
purchaseOrderPriced1 = purchaseOrderPriced;

// generate a random tracking id.
int rangeBound = 100000000;
java.util.Random random = new java.util.Random();
long percentOffset = random.nextLong();
double offset = percentOffset * rangeBound;

java.text.NumberFormat formatter=new
java.text.DecimalFormat("#####");
java.lang.String trackingId = "shp".concat(formatter.format(offset));
purchaseOrderPriced1.getDataObject("shippingSummary").setString("tracki
ngId", trackingId);
```

ReturnOrderStatus Assign activity

The ReturnOrderStatus Assign activity is called in the event that the order has too many items. In this case, an error message, ERROR: Order to big to ship, is assigned to the shippingSummary/shippingStatus element of the response pricedPurchaseOrder message. Figure 10-22 shows the configuration.

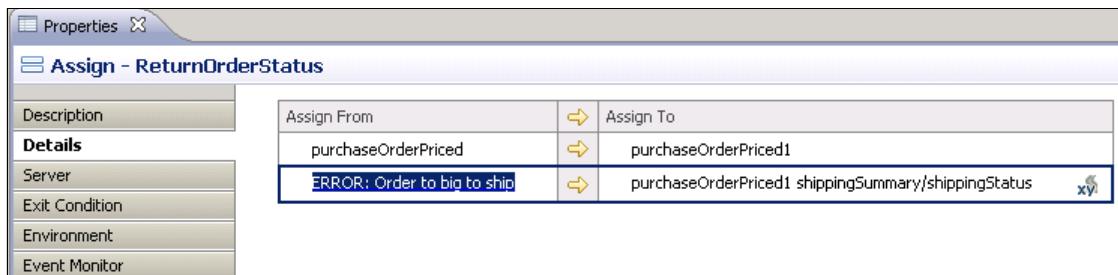


Figure 10-22 ReturnOrderStatus properties

Reply activity

The Reply activity returns the response to the business process client, as shown in Figure 10-23.

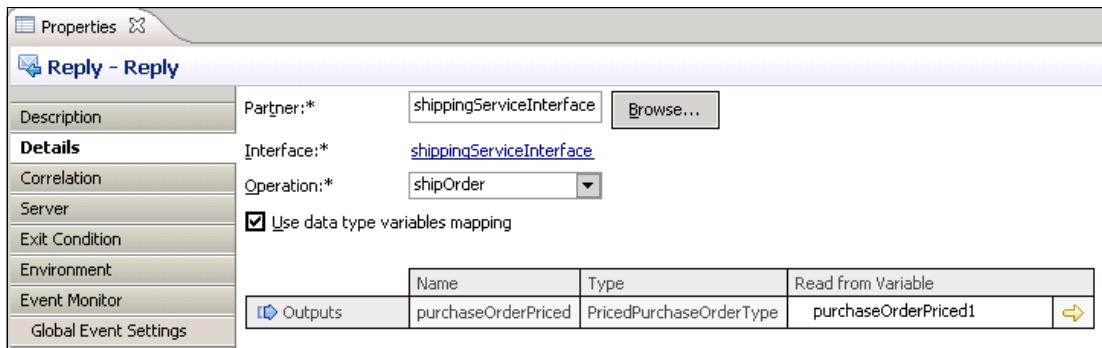


Figure 10-23 Reply properties

10.6.5 Export component and binding

The shippingReviewExport export component is used by the OrderService message flow to call the business process, shippingServiceProcess. The interface that is associated with the export is shippingServiceInterface. The export is used to receive request messages from the OrderService message flow placed on the queue (SHIP.REQ) and send responses back on the response queue (SHIP.RESP). Figure 10-24 on page 451 shows the MQ bindings configuration for this export.

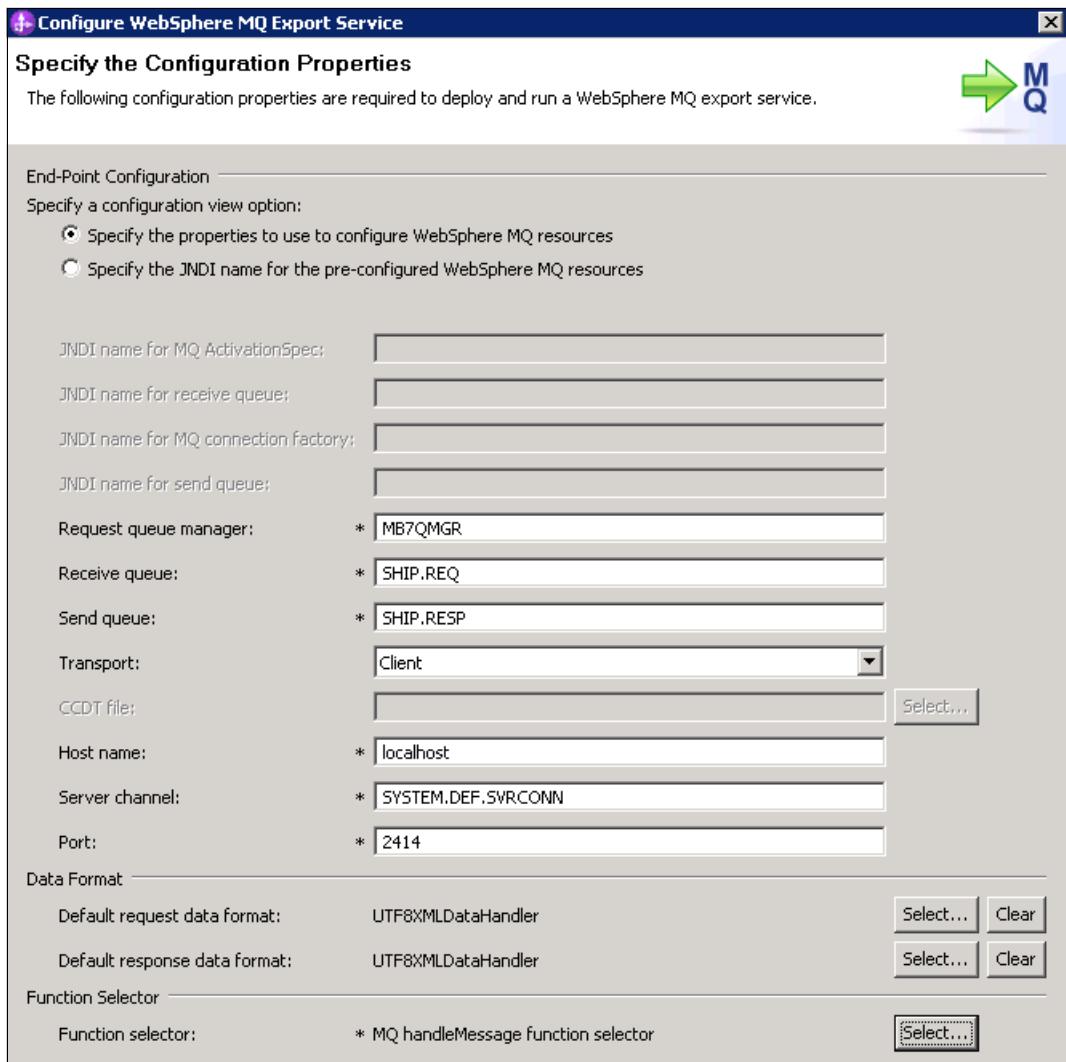


Figure 10-24 The *shippingReviewExport* binding properties

10.6.6 ShippingServiceModule Project Interchange

The OrderService message flow that you create in section 10.7.1, “OrderService message flow” on page 452 uses the SCAResponse node to interact with the shippingService business process. To configure the SCAResponse node, some project artifacts from the ShippingServiceModule are required.

Export the ShippingServiceModule project as a Project Interchange file:

1. Navigate to **File** → **Export** → **Other** → **Project Interchange**.
2. Follow the prompts to export the ShippingServiceModule project to a zip file called, `ShippingServiceModule.zip`.

10.7 WebSphere Message Broker

In this section, we demonstrate the steps in creating the OrderService message flow, shown in Figure 10-25, that will be deployed on WebSphere Message Broker. The service performs the Order handling process that is described in 10.3, “Scenario” on page 431. As a part of the message flow, we configure the monitoring events that the WebSphere Business Monitor uses in providing the key process metrics required.

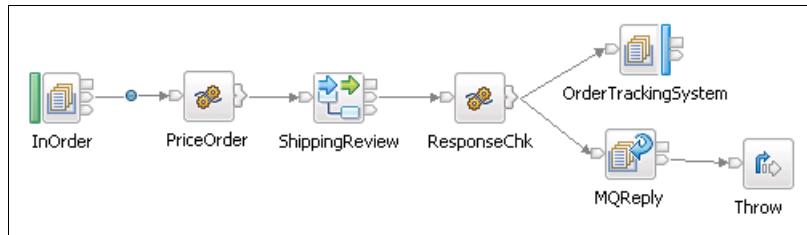


Figure 10-25 OrderServiceFlow message flow

In Figure 10-25:

- ▶ *PriceOrder* compute node calculates the priceSummary for the Order that includes applying the discount of 15% for GOLD customer types.
- ▶ *ShippingReview* SCAResponse node sends the pricedPurchaseOrder to the shippingReviewExport for processing by the shippingServiceProcess business process.
- ▶ *ReponseChk* compute node reviews the updated pricedPurchaseOrder to determine if there were errors in the shipping review.
- ▶ *MQReply* node sends the pricePurchaseOrder that contains errors back to the client through the MQMD ReplyToQ.
- ▶ *OrderTrackingSystem* MQOutput node sends a successfully priced and shipped pricePurchaseOrder on to the Order tracking system.

10.7.1 OrderService message flow

In this section, we discuss the OrderService message flow.

Message set

The OrderServiceMessageSet message set contains the message definitions for the same set of business objects that are defined in the XML Schema in Example 10-4 on page 442. It also contains the .outsca file that is necessary for the SCAResponse node to interact with the WebSphere Process Server shippingServiceProcess business process:

1. Navigate to **File → New → Message Set** to create a new message set called OrderServiceMessageSet.
2. Import the message definitions and .outsca file from the project interchange file ShippingServiceModule.zip that was created in 10.6.6, “ShippingServiceModule Project Interchange” on page 451:
 - a. Navigate to **File → New → Message Definition From**.
 - b. Select **SCA Import or Export**.
 - c. Select the project interchange archive, as shown in Figure 10-26.

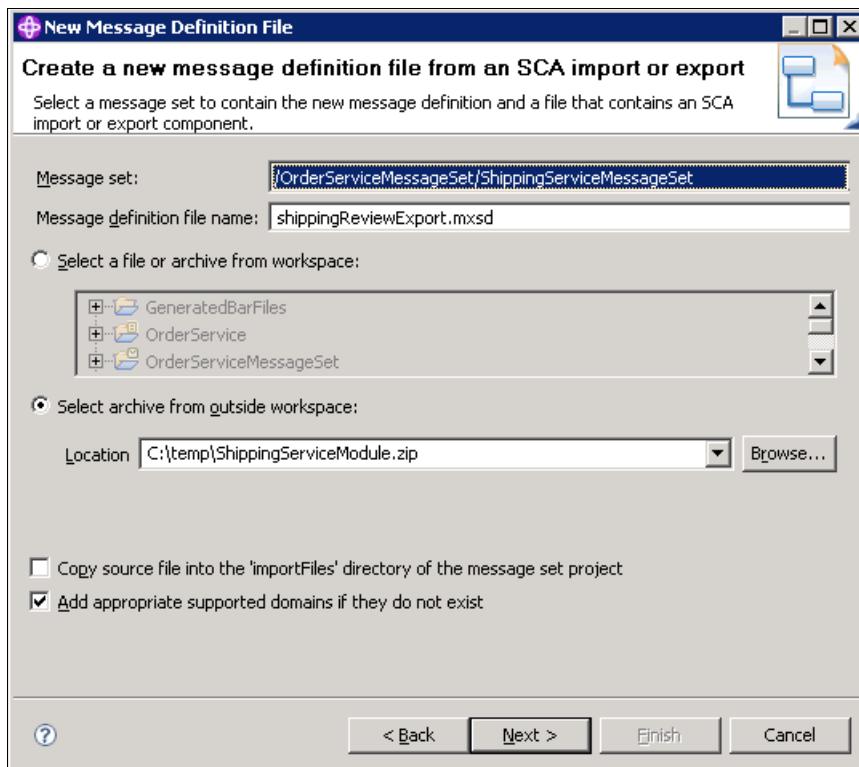


Figure 10-26 Importing the message definition

- d. Click **Next**. Click **Finish**.

InOrder MQInput node

Set the Queue name, INPUT_ORDER_Q, in the Basic settings, Figure 10-27, for the MQInput node.



Figure 10-27 MQInput Basic properties

In the Input Message Parsing settings, Figure 10-28, the Message Domain is set to XMLNSC, and the Message Set is OrderServiceMessageSet.



Figure 10-28 MQInput Input Message Parsing properties

In the Advanced settings, Figure 10-29, the Transaction mode is set to No.

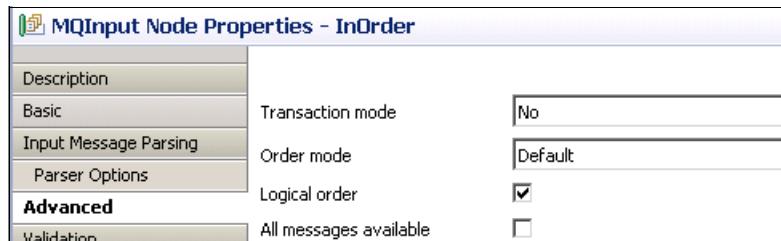


Figure 10-29 MQInput Advanced properties

PriceOrder Compute node

The PriceOrder Compute node calculates the pricingSummary details of the pricedPurchaseOrder. Example 10-8 on page 455 provides the ESQL that is necessary to satisfy the business requirements.

Example 10-8 PriceOrder ESQL

```
CREATE COMPUTE MODULE OrderServiceFlow_PriceOrder
  CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
      DECLARE PO NAMESPACE 'http://www.ibm.com/ProcessOrder';
      DECLARE RUNNING_TOTAL DECIMAL 0.00;
      DECLARE QUANTITY DECIMAL 0.00;
      DECLARE PRICE DECIMAL 0.00;
      DECLARE DELIVERY_CHARGES DECIMAL 19.99 ;
      DECLARE DISCOUNT DECIMAL 0.00;
      DECLARE GOLD_DISCOUNT DECIMAL 0.00;
      DECLARE ITEM_COUNT INTEGER 0;

      CALL CopyMessageHeaders();
      SET Environment.MQMD = InputRoot.MQMD;
      SET OutputRoot.XMLNSC.PO:purchaseOrderPriced =
InputRoot.XMLNSC.PO:purchaseOrder;

      --calculate the total cost of each item ordered
      FOR price_source AS
        InputRoot.XMLNSC.PO:purchaseOrder.PO:items.PO:item[] DO
          SET QUANTITY = CAST (price_source.PO:quantity AS DECIMAL);
          SET PRICE = CAST (price_source.PO:price AS DECIMAL);
          SET RUNNING_TOTAL = RUNNING_TOTAL + (PRICE * QUANTITY);
          SET ITEM_COUNT = ITEM_COUNT + CAST (price_source.PO:quantity
AS INTEGER);
        END FOR;

      --add the purchaseOrderPriced fields
      CREATE LASTCHILD OF OutputRoot.XMLNSC.PO:purchaseOrderPriced
NAMESPACE PO NAME 'transactionId' VALUE
Environment.Monitoring.EventCorrelation.localTransactionId;
      CREATE LASTCHILD OF OutputRoot.XMLNSC.PO:purchaseOrderPriced
NAMESPACE PO NAME 'priceSummary';
      CREATE LASTCHILD OF OutputRoot.XMLNSC.PO:purchaseOrderPriced
NAMESPACE PO NAME 'shippingSummary';
      CREATE LASTCHILD OF
OutputRoot.XMLNSC.PO:purchaseOrderPriced.PO:priceSummary NAMESPACE PO
NAME 'itemTotal' VALUE RUNNING_TOTAL;
      CREATE LASTCHILD OF
OutputRoot.XMLNSC.PO:purchaseOrderPriced.PO:priceSummary NAMESPACE PO
NAME 'itemCount' VALUE ITEM_COUNT;
```

```

CREATE LASTCHILD OF
OutputRoot.XMLNSC.P0:purchaseOrderPriced.P0:priceSummary NAMESPACE PO
NAME 'deliveryCharges' VALUE DELIVERY_CHARGES;

IF InputRoot.XMLNSC.P0:purchaseOrder.P0:customerType = 'GOLD'
THEN
    --for GOLD customers give a 15% discount
    SET GOLD_DISCOUNT =
(OutputRoot.XMLNSC.P0:purchaseOrderPriced.P0:priceSummary.PO:itemTotal
+ DELIVERY_CHARGES) * 85/100;
    SET GOLD_DISCOUNT = ROUND(GOLD_DISCOUNT, 2 MODE ROUND_UP);
    SET DISCOUNT =
(OutputRoot.XMLNSC.P0:purchaseOrderPriced.P0:priceSummary.PO:itemTotal
+ DELIVERY_CHARGES) - GOLD_DISCOUNT;
    CREATE LASTCHILD OF
OutputRoot.XMLNSC.P0:purchaseOrderPriced.P0:priceSummary NAMESPACE PO
NAME 'discount' VALUE DISCOUNT;
    CREATE LASTCHILD OF
OutputRoot.XMLNSC.P0:purchaseOrderPriced.P0:priceSummary NAMESPACE PO
NAME 'invoice' VALUE GOLD_DISCOUNT;
ELSE
    -- priced like GUEST
    CREATE LASTCHILD OF
OutputRoot.XMLNSC.P0:purchaseOrderPriced.P0:priceSummary NAMESPACE PO
NAME 'discount' VALUE DISCOUNT;
    CREATE LASTCHILD OF
OutputRoot.XMLNSC.P0:purchaseOrderPriced.P0:priceSummary NAMESPACE PO
NAME 'invoice' VALUE (DELIVERY_CHARGES + RUNNING_TOTAL);
END IF;

RETURN TRUE;
END;

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

```

ShippingReview SCAResponse node

The ShippingReview SCAResponse invokes the shippingServiceProcess business process using the imported .outsca file:

1. Drag-and-drop the shippingReviewExport.outsca file from the OrderServiceMessageSet onto the OrderService message flow, as shown in Figure 10-30.

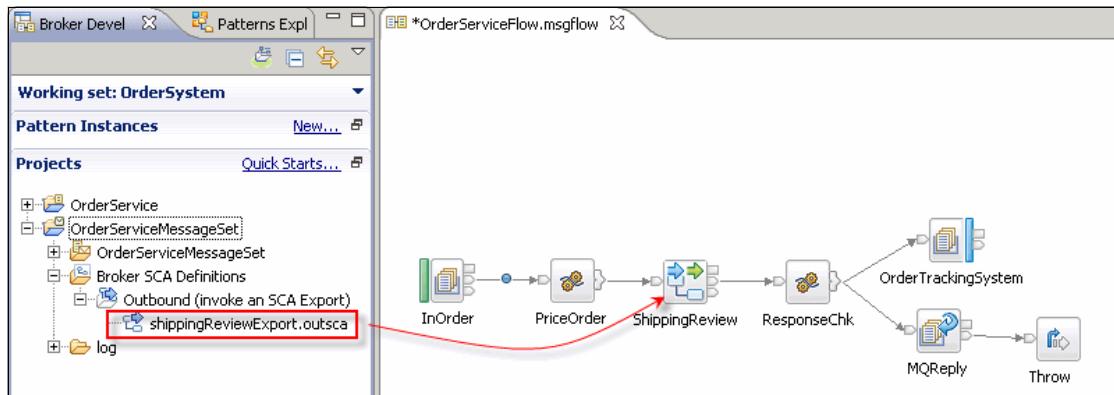


Figure 10-30 Importing the SCAResponse node

2. A pop-up message is displayed that gives you the option to choose the operation to call and to invocation mode. Press **Ok**, accepting the defaults, as shown in Figure 10-31.



Figure 10-31 SCA Response node properties

ResponseChk Compute node

The PriceOrder Compute node routes the message back to the client if the shippingStatus returns an error in the shippingStatus element. Example 10-9 shows the ESQL for ResponseChk.

Example 10-9 ResponseChk ESQL

```
CREATE COMPUTE MODULE OrderServiceFlow_ResponseChk
    CREATE FUNCTION Main() RETURNS BOOLEAN
        BEGIN
            CALL CopyMessageHeaders();
            DECLARE PO NSPACE 'http://www.ibm.com/ProcessOrder';
            SET OutputRoot.XMLNSC.PO:purchaseOrderPriced =
InputRoot.XMLNSC.*:shipOrderResponse.PO:purchaseOrderPriced;
            IF
                OutputRoot.XMLNSC.PO:purchaseOrderPriced.PO:shippingSummary.PO:shipping
                Status IS NULL THEN
                    PROPAGATE TO TERMINAL 'out';
                ELSE
                    SET OutputRoot.MQMD = Environment.MQMD;
                    PROPAGATE TO TERMINAL 'out1';
                END IF;
                RETURN FALSE;
            END;

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

OrderTrackingSystem MQOutput node

In the Basic settings, Figure 10-32 on page 459, set the Queue name, TRACKING_ORDER_Q, for the MQOutput node.



Figure 10-32 MQOutput properties

10.7.2 Monitoring events

Monitoring events in WebSphere Message Broker are configured using the nodes Monitoring tab in the Properties view, shown in Figure 10-33. There are no events that are configured by default. Each event can be enabled or disabled in this tab or using the command-line interface that we discussed in 10.7.3, “Command-line configuration” on page 466.

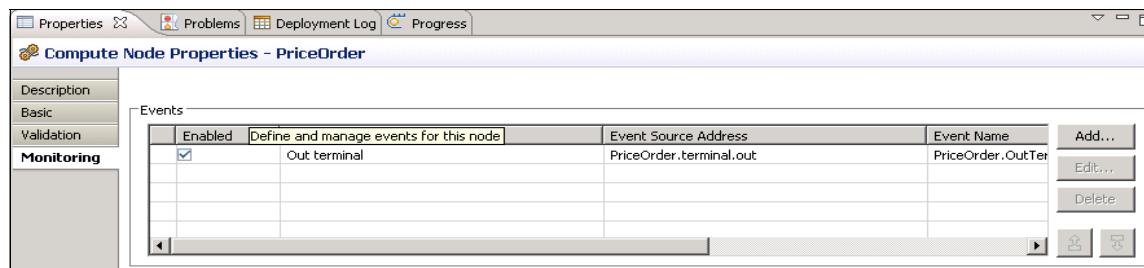


Figure 10-33 Monitoring tab

The event configuration has three tabs that deal with the different aspects of the event:

- ▶ Under the *Basic* tab, you define the general properties that relate to the event, as shown in Figure 10-34 on page 460. The Event Source, Event Source Address, and Event Name settings define the identity of the event. The Event Filter enables an XPATH expression to control whether or not an event is emitted. The Event Payload section allows additional data, including the entire encoded message, to be added in the event.

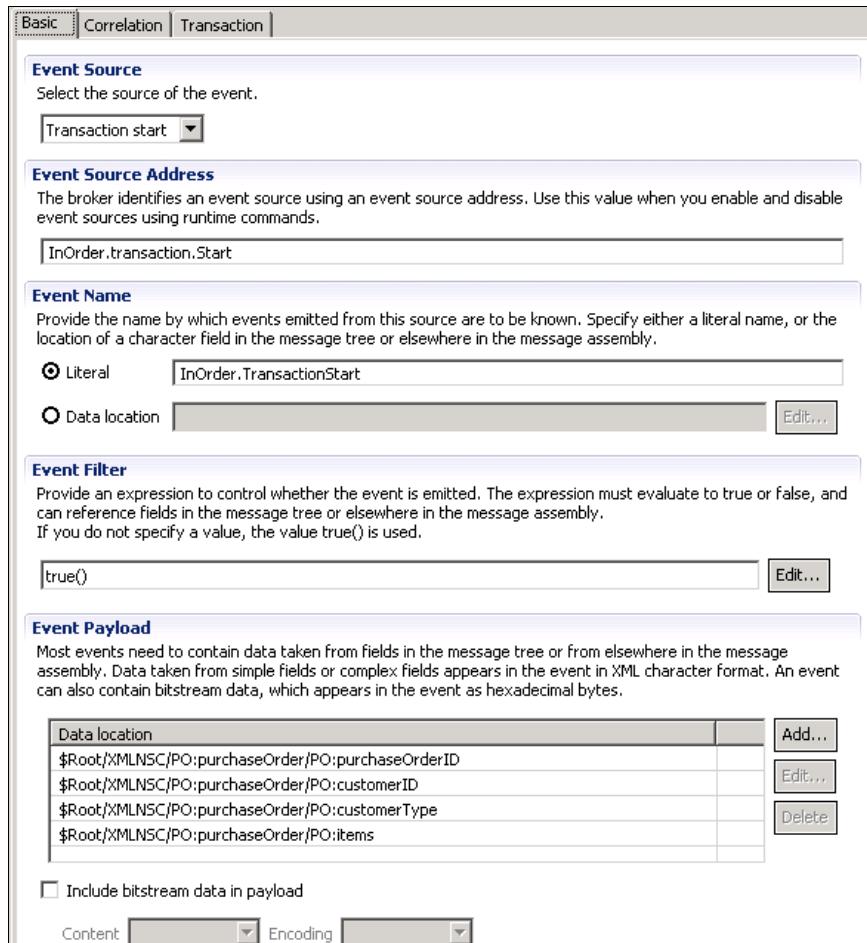


Figure 10-34 Monitoring event Basic properties tab

- ▶ Under the *Correlation* tab, you define three different correlators that can be used by a monitor or event consumer application to relate multiple events together, as shown in Figure 10-35 on page 461:
 - The *Local transaction correlator* field defaults to be automatically populated with a unique identifier, which is common across all events that are emitted during a single invocation of the message flow. It can be overridden to use a field in the message, which is often times from a header element.
 - The *Parent transaction correlator* and *Global transaction correlator* fields are empty by default. They enable two additional levels of correlation that can be used to relate external events.

A more detailed discussion about correlation is in the WebSphere Message Broker Information Center:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac60389_.htm

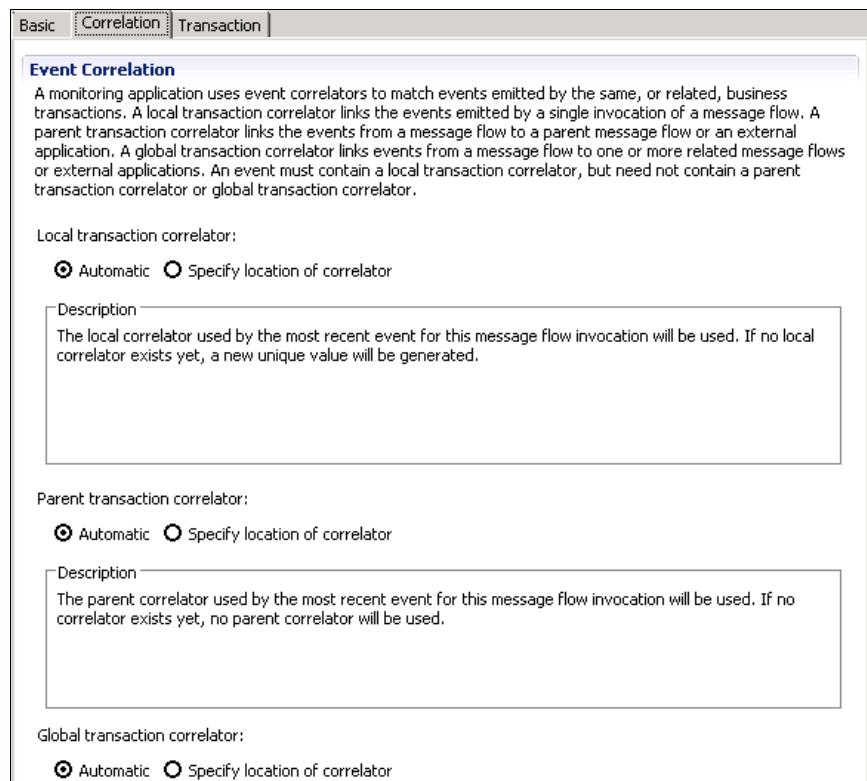


Figure 10-35 Monitoring event Correlation properties

- ▶ Under the *Transaction* tab, you define the context for how the event is coordinated with the message flow transaction, as shown in Figure 10-36 on page 462:
 - The *Message Flow* option coordinates the event to be emitted if the message flow commit its unit of work successfully. If the message fails this event is rolled back.
 - The value in the *Independent* option coordinates the event to be emitted in a second unit of work. An event with this setting will emit the event regardless of if the message flow finishes successfully. This setting can be used in the case of an event generated in the error path of a message flow.

- The *None* option coordinates the event immediately. It is available as soon the message passes through the event source.

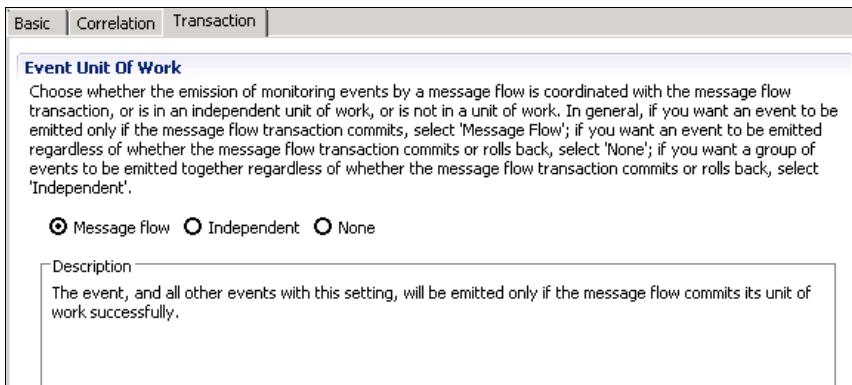


Figure 10-36 Monitoring event Transaction properties

In the next sections, we configure the monitoring events that the OrderService message flow emits.

InOrder

There are three transaction-related events that are configured for the InOrder MQInput node: Transaction Start, Transaction End, and Transaction Rollback. Transaction events are only available on Input nodes:

1. Add an event to the InOrder node. Set the Event Source to be Transaction start.
2. Add the Event Payload data fields, which we show in Table 10-2.

Table 10-2 Event payload data fields

Data Location
\$Root/XMLNSC/PO:purchaseOrder/PO:purchaseOrderID
\$Root/XMLNSC/PO:purchaseOrder/PO:customerID
\$Root/XMLNSC/PO:purchaseOrder/PO:customerType
\$Root/XMLNSC/PO:purchaseOrder/PO:items

3. Add another event to the InOrder node. Set the Event Source to be Transaction end.
4. Add a third event to the InOrder node. Set the Event Source to be Transaction rollback.

PriceOrder

The PriceOrder Compute node emits a single event to designate a high priority order. Configure the event on the Out terminal of the node:

1. Set the Event Filter XPATH to be

`$Root/XMLNSC/P0:purchaseOrderPriced/P0:priceSummary/P0:invoice >= 1000.00.`

Figure 10-37 shows the Basic settings.

The screenshot shows the 'Event Source' configuration dialog. It includes sections for 'Event Source', 'Event Source Address', 'Event Name', and 'Event Filter'. The 'Event Filter' section contains an XPATH expression: `$Root/XMLNSC/P0:purchaseOrderPriced/P0:priceSummary/P0:invoice >= 1000.00`.

Event Source	Select the source of the event.
Out terminal	<input type="button" value="▼"/>
Event Source Address	The broker identifies an event source using an event source address. Use this value when you enable and disable event sources using runtime commands.
PriceOrder.terminal.out	
Event Name	Provide the name by which events emitted from this source are to be known. Specify either a literal name, or the location of a character field in the message tree or elsewhere in the message assembly.
<input checked="" type="radio"/> Literal	PriceOrder.OutTerminal
<input type="radio"/> Data location	<input type="button" value="Edit..."/>
Event Filter	Provide an expression to control whether the event is emitted. The expression must evaluate to true or false, and can reference fields in the message tree or elsewhere in the message assembly. If you do not specify a value, the value true() is used.
<input checked="" type="radio"/> \$Root/XMLNSC/P0:purchaseOrderPriced/P0:priceSummary/P0:invoice >= 1000.00 <input type="button" value="Edit..."/>	

Figure 10-37 PriceOrder event Basic properties

2. Set the Event Unit Of Work in the Transaction tab to be Independent, which is shown in Figure 10-38. The Independent setting demonstrates the ability to monitor high priority transactions, even when the message flow fails.

The screenshot shows the 'Event Unit Of Work' configuration dialog under the 'Transaction' tab. It includes sections for 'Event Unit Of Work' and a radio button selection for 'Independent'.

Basic	Correlation	Transaction
Event Unit Of Work	Choose whether the emission of monitoring events by a message flow is coordinated with the message flow transaction, or is in an independent unit of work, or is not in a unit of work. In general, if you want an event to be emitted only if the message flow transaction commits, select 'Message Flow'; if you want an event to be emitted regardless of whether the message flow transaction commits or rolls back, select 'None'; if you want a group of events to be emitted together regardless of whether the message flow transaction commits or rolls back, select 'Independent'.	
<input type="radio"/> Message flow	<input checked="" type="radio"/> Independent	<input type="radio"/> None

Figure 10-38 PriceOrder event Transaction properties

ShippingReview

The ShippingReview SCAResult node emits two events: one when the message passes through the In terminal and another when it passes through the Out terminal. The combination of these events allows the monitor application to determine the duration of the call to the shippingServiceProcess business process:

1. Configure a new event. Set the Event Source to In Terminal, as shown in Figure 10-39.

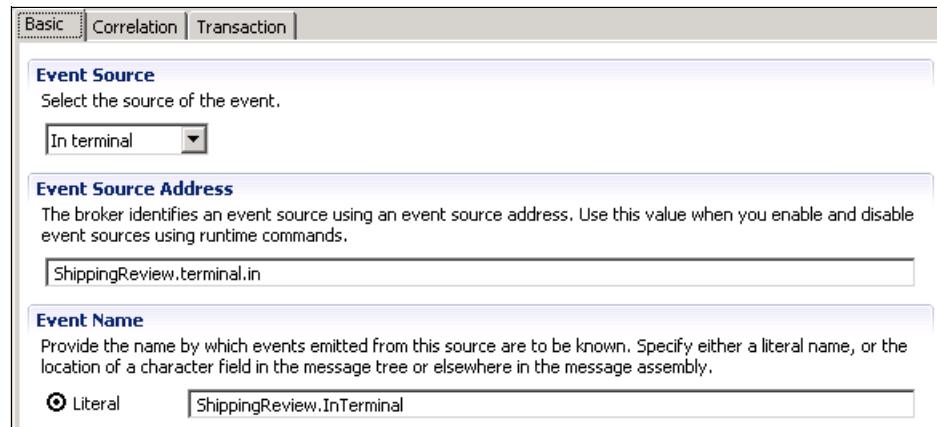


Figure 10-39 ShippingReview In terminal event properties

2. Configure a second event, setting the Event Source to Out terminal, as shown in Figure 10-40.

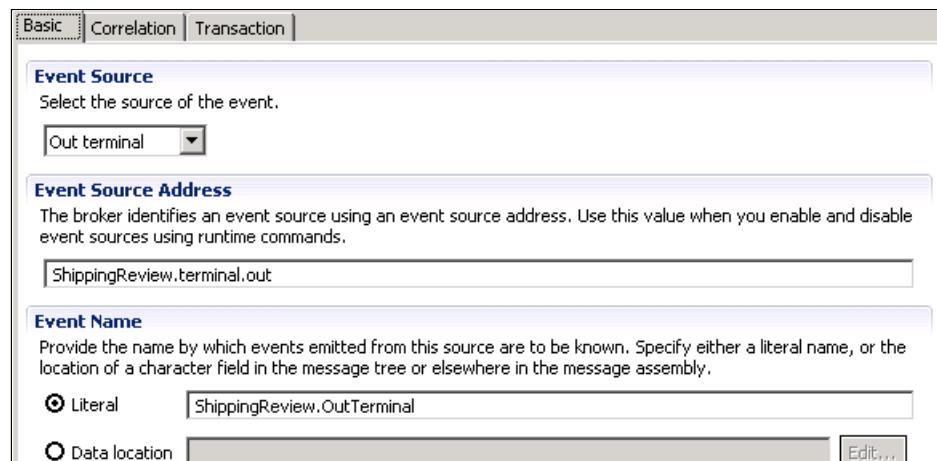


Figure 10-40 ShippingReview Out terminal event properties

MQReply

The MQReply node sends orders that were not able to be shipped back to the client using the MQMD ReplyToQ. The event captures the status information from the updated purchase order:

1. Configure an event. Set the Event Source to In Terminal, as shown in Figure 10-41.

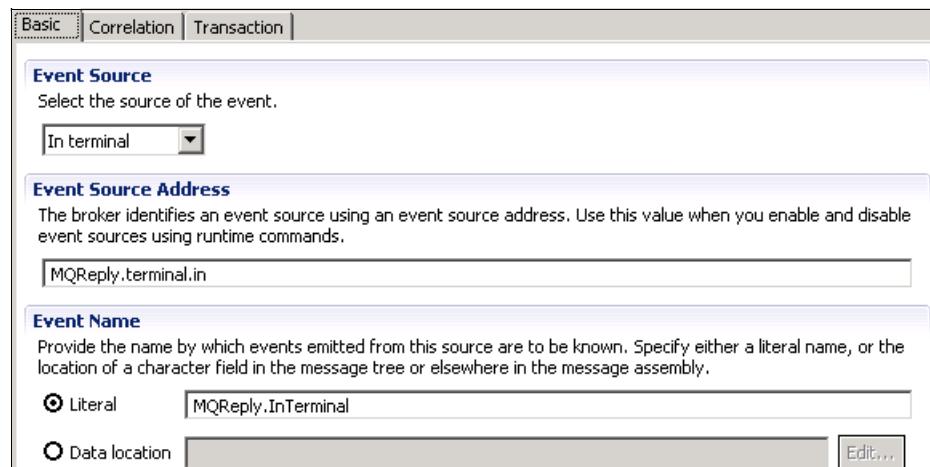


Figure 10-41 MQReply In terminal event properties

2. Add an Event Payload field, as shown in Figure 10-42.

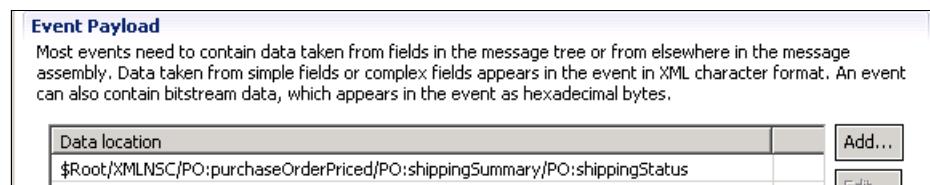


Figure 10-42 MQReply In terminal event payload properties

3. Set the **Transaction → Event Unit Of Work** to be Independent (10.7.4, “Enabling monitoring” on page 468).

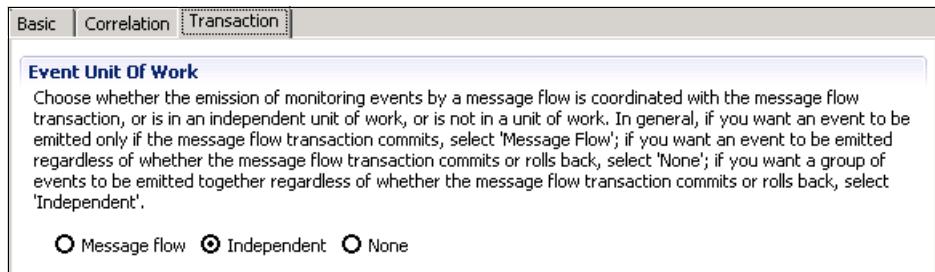


Figure 10-43 MQReply In terminal Transaction properties

10.7.3 Command-line configuration

Optional: This section is optional and provided to highlight the command-line features that WebSphere Message Broker supports in relation to monitoring events.

Using WebSphere Message Broker you can configure and manage monitoring events using the command-line. The benefits of the features are:

- ▶ The ability to set or modify events on an already deployed flow. There is no need to redeploy a bar file with updated events.
- ▶ The ability to reuse monitoring profiles across flows that have common event sources.

The primary interfaces to these features is through the `mqsicchangeflowmonitoring` and `mqsireportflowmonitoring` commands.

In the following set of commands, we demonstrate command-line examples that report and manage events that are already configured on a message flow:

- ▶ Report all configured events for a message flow
`mqsireportflowmonitoring MB7BROKER -e OrderServiceExecGroup -f OrderServiceFlow -n`
- ▶ Report all available events for a message flow
`mqsireportflowmonitoring MB7BROKER -e OrderServiceExecGroup -f OrderServiceFlow -a`

- ▶ Disable a specific event source on a message flow
`mqsicchangeflowmonitoring MB7BROKER -e OrderServiceExecGroup -f OrderServiceFlow -s "MQReply.terminal.in" -i disable`
- ▶ Enable a specific event source on a message flow
`mqsicchangeflowmonitoring MB7BROKER -e OrderServiceExecGroup -f OrderServiceFlow -s "MQReply.terminal.in" -i enable`

Monitoring events can also be defined through the command-line by creating a monitoring profile that is a configurable service and associating it with a given message flow, which allows events to be created or modified on an already deployed message flow.

A monitoring profile consists of a configurable service and profile XML file. The XML file lists the event sources in a message flow that will emit events and defines the properties for each event:

1. One way to create a monitoring profile xml is to use the `mqsicreportflowmonitoring` command to create an equivalent XML for an existing message flow. The following command demonstrates this for the OrderServiceFlow.
`mqsicreportflowmonitoring MB7BROKER -e OrderServiceExecGroup -f OrderServiceFlow -x -p OrderServiceFlowMP.xml`
2. Create the configurable service using:
`mqsicreateconfigurableservice MB7BROKER -c MonitoringProfiles -o OrderServiceFlowMonitoringProfile`
3. Use the following command to associate the configurable service, OrderServiceFlowMonitoringProfile, with the monitoring profile xml that you created in step 1.
`mqsicchangeproperties MB7BROKER -c MonitoringProfiles -o OrderServiceFlowMonitoringProfile -n profileProperties -p OrderServiceFlowMP.xml`
4. Associate the monitoring profile to the message OrderServiceFlowMonitoringProfile to the OrderServiceFlow message flow.
`mqsicchangeflowmonitoring MB7BROKER -e OrderServiceExecGroup -f OrderServiceFlow -m OrderServiceFlowMonitoringProfile`

For more information about monitoring profiles, refer to the WebSphere Message Broker Information Center at:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.e.tools.mft.doc/ac37900_.htm

10.7.4 Enabling monitoring

Deploy the OrderServiceFlow message flow, if it's not already been deployed. By default, monitoring is inactive when a new flow is deployed. The following steps describe how to activate monitoring so that the configured events are emitted:

1. Issue the following command to enable monitoring on the OrderServiceFlow message flow deployed in the OrderServiceExecGroup execution group:

```
mqsicchangeflowmonitoring MB7BROKER -e OrderServiceExecGroup -f  
OrderServiceFlow -c active
```

2. Use the following command to confirm that the monitor events are active for the message flow:

```
mqswireportflowmonitoring MB7BROKER -e OrderServiceExecGroup -f  
OrderServiceFlow
```

Deploying your message flow automatically: If you use the WebSphere Message Broker Test Client and configure it to manage the deployment of the message flow BAR file automatically, it can deactivate monitoring on the message flow. So it is recommended that you deploy your message flow manually and set the Test Client configuration accordingly.

10.8 WebSphere Business Monitor

WebSphere Message Broker provides the capability to generate a WebSphere Business Monitor model based on the event sources that are configured in the message flow.

The generated model contains:

- ▶ Inbound events for each event source that is defined in the message flow. Inbound events contain the correlation expression, event sequence path, filter condition, and event parts that describe the Event Payload elements.
- ▶ Metrics, triggers, and key performance indicators (KPI). Table 10-3 on page 469 details the templates that are used in generating the monitor model.

Table 10-3 Templates for generating the monitor model

Template name	Event sources	Created KPI/Metric
Average Transaction Duration	transaction.Start transaction.End transaction.Rollback	Average Transaction Duration for KPI (stopwatch)
		Average Transaction Duration
Number of Failed Transactions	transaction.Rollback	Number of Failed Transactions(trigger)
		Number of Failed Transactions(metric)
		Failed Transaction Time(metric)
		Number of Failed Transactions - Measure
		Failed Transaction Time Dimension
Message Flow Correlation	transaction.Start	Broker
		ExecutionGroup
		parentTransactionId
		globalTransactionId

10.8.1 Monitor model

In this section, we demonstrate the steps to generate a basic monitor model based on a WebSphere Message Broker message flow. We also extend it to provide the performance measurements that are necessary to satisfy the requirements in 10.3, “Scenario” on page 431.

In WebSphere Message Broker Toolkit, export the monitor model for the OrderService message flow:

1. Navigate to **File → Export → Business Monitoring → Application Monitor Information**.

2. Select the OrderServiceFlow.msgflow to export. Set the Target file to OrderServiceMonitorModel.zip, as shown in Figure 10-44.

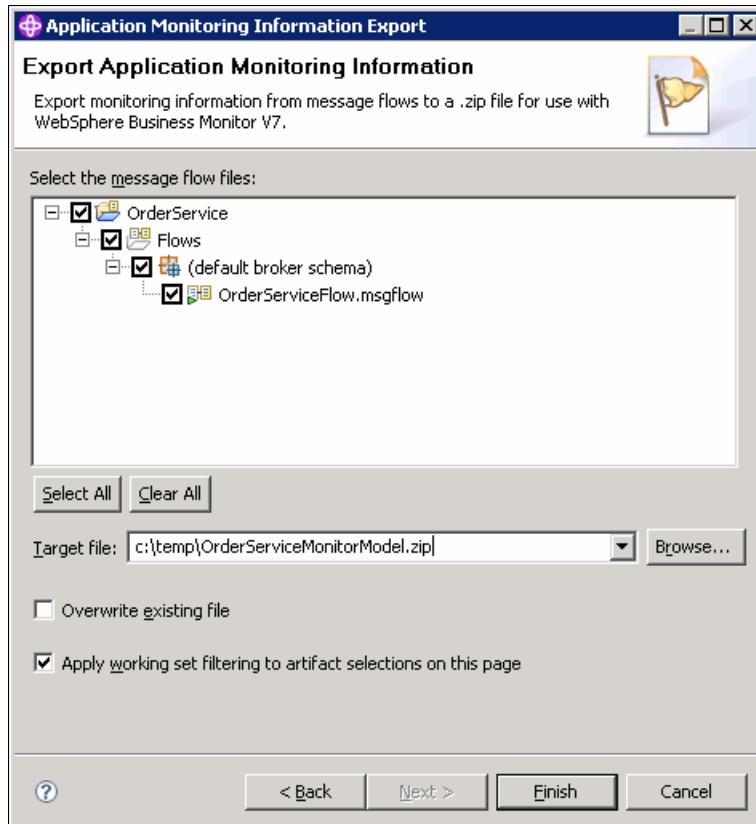


Figure 10-44 Export Application monitoring information

In WebSphere Integration Developer, switch perspectives to the Business Monitoring perspective, and import the monitor model:

1. Navigate to **File** → **Import** → **Business Monitoring** → **Application monitor information .zip file**.
2. Select the OrderServiceMonitorModel.zip file to import, as shown in Figure 10-45 on page 471.

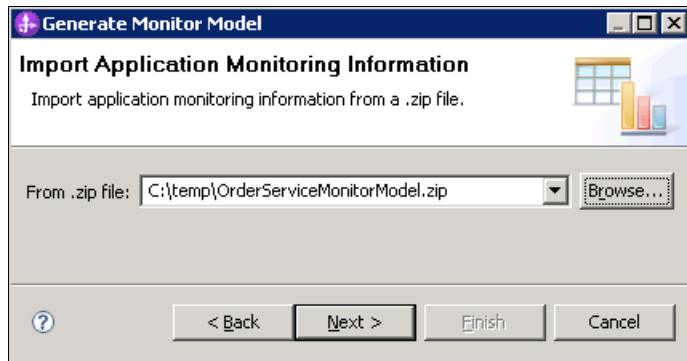


Figure 10-45 Import Application monitoring information

3. Set the Target monitor project to OrderMonitorProject and the Target monitor model name to OrderMonitor, as demonstrated in Figure 10-46.

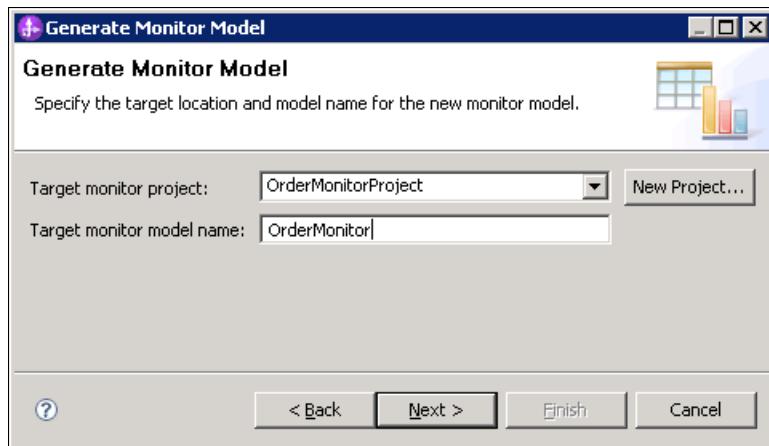


Figure 10-46 Generate Monitor Model view

4. On the Choose What to Monitor panel, Figure 10-47 on page 472, you can select, from the available event sources, the events that you want to monitor in this model. As a part of this exercise, select all of the Monitoring Templates and Emitted Events. Proceed through the rest of the panels reviewing the content. Select **Finish** to complete the import.

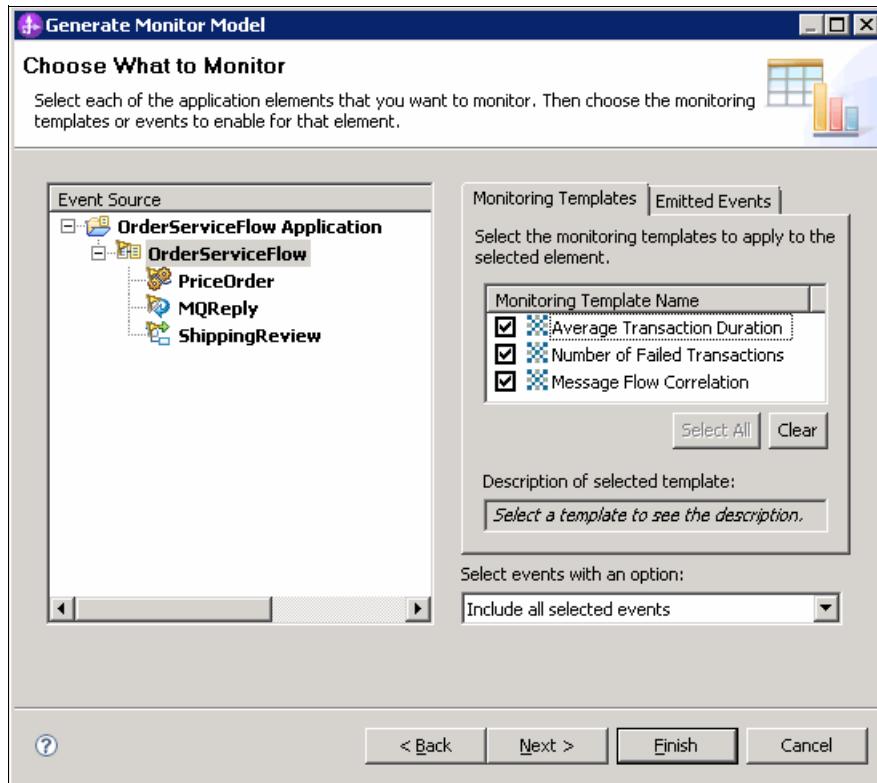


Figure 10-47 Choose What to Monitor properties

Extending the model

In 10.3, “Scenario” on page 431, we discussed the different monitoring requirements. We extend the model to provide the additional metrics and KPIs that are necessary to complete the solution.

The basic monitoring model imported already provides the KPI for *Average Order Processing Time*. In the generated basic model it is called OrderServiceFlow Average Transaction Duration.

Metrics

In this section, we add three new Metrics:

1. Create a new Metric called Customer ID, as shown in Figure 10-48 on page 473.

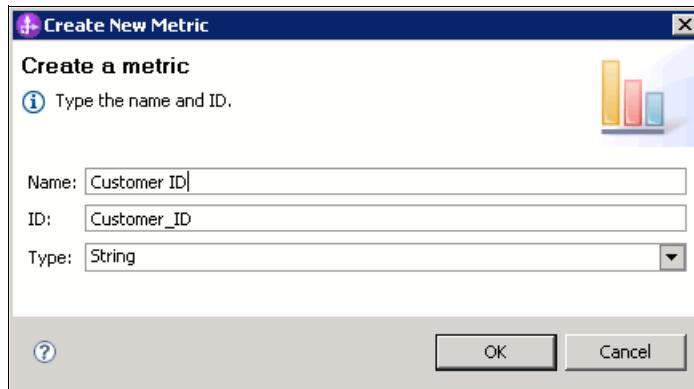


Figure 10-48 Customer ID metric

2. Add a Metric Value Expression of InOrder.transaction.Start/customerID, as shown in Figure 10-49.

Figure 10-49 Customer ID metric value

3. Repeat steps 1 and 2 of this section to create the Metrics in Table 10-4.

Table 10-4 Metric settings

Metric	Metric Value Expression
Customer Type	InOrder.transaction.Start/customerType
Order Shipping Status	MQReply.terminal.in/shippingStatus

Triggers

In this section, we add a new Trigger:

1. Create a new Trigger called High Priority Trigger, as shown in Figure 10-50 on page 474.

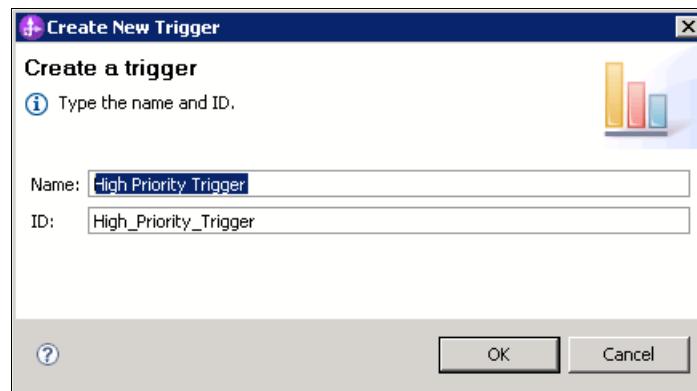


Figure 10-50 High Priority trigger

2. Add a Trigger Source with Source Type of Event and Source PriceOrder.terminal.out, as shown in Figure 10-51.

Figure 10-51 High Priority trigger sources

Adding new Counters

In this section, we add two new Counters:

1. Create a new Counter called High Priority, as shown in Figure 10-52 on page 475.

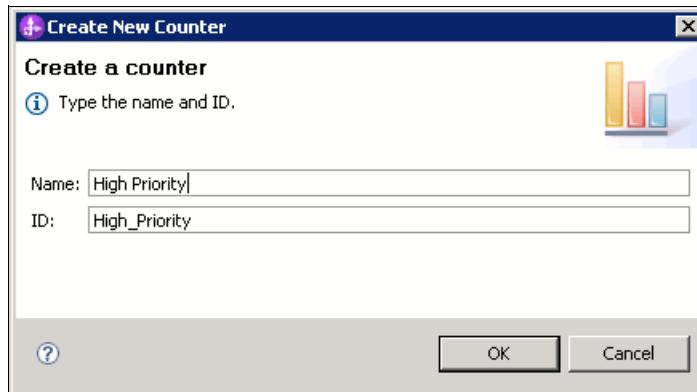


Figure 10-52 High Priority counter

2. Add a Counter Control with Trigger/Inbound Event of High Priority Trigger and a Resulting Action of Add One, as shown in Figure 10-53.

Figure 10-53 High Priority counter control properties

3. Repeat steps 1 and 2 in this section to create the Counter in Table 10-5.

Table 10-5 New counter properties

Metric	Metric Value Expression	Resulting Action
Succeeded	InOrder.transaction.End	Add One

Adding a new stopwatch

In this section, we add a new stopwatch. To add a new stopwatch:

1. Create a new Stopwatch called Order Shipping Time, as shown in Figure 10-54.

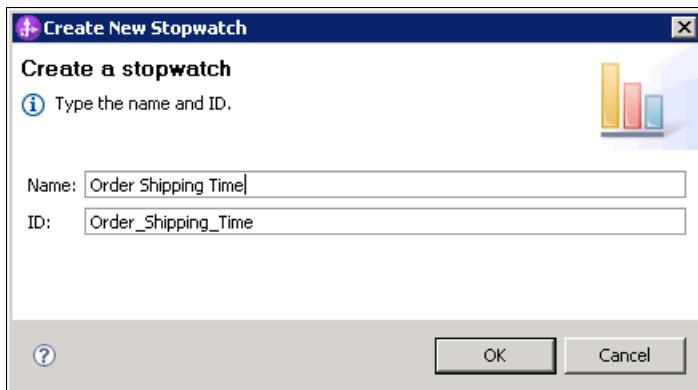


Figure 10-54 Order Shipping Time stopwatch

2. Add two Stopwatch Controls, as shown in Figure 10-55.

Figure 10-55 Order Shipping Time stopwatch control properties

Rename generated Key Performance Indicators

In this section we rename the generated Key Performance Indicators (KPI) and add four new KPIs:

- Under the KPI Model tab, find the OrderServiceFlow Average Transaction Duration for KPI item under **OrderMonitor → Template KPI Context**.

Change the Name to Avg Order Processing Time, as shown in Figure 10-56 on page 477.

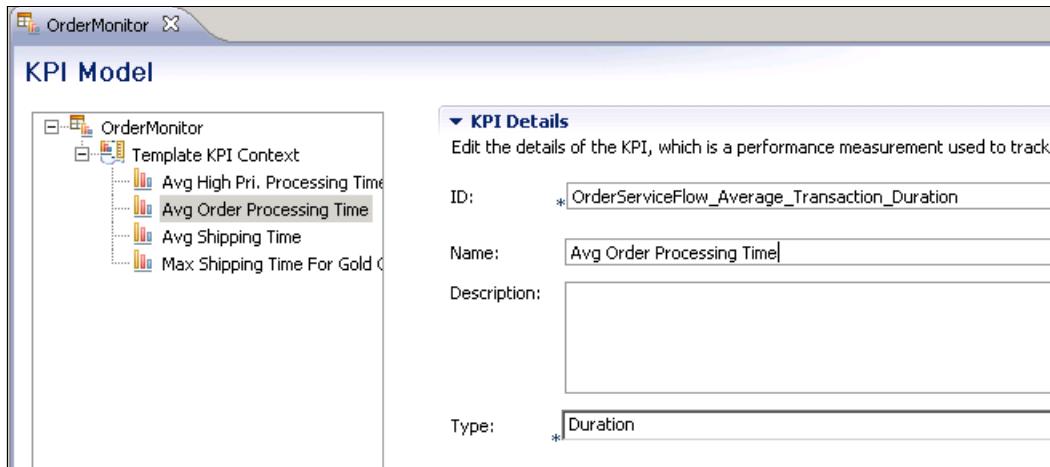


Figure 10-56 Average Order Processing Time KPI

2. Under the KPI Model tab, create a new KPI called Avg High Pri. Processing Time. Set the Type to Duration, as shown in Figure 10-57.

This is a screenshot of a 'KPI Details' dialog box. It contains fields for ID, Name, Description, and Type. The 'ID' field is set to 'Avg_High_Pri_Processing_Time'. The 'Name' field is set to 'Avg High Pri. Processing Time'. The 'Type' field is set to 'Duration'. The 'Description' field is empty.

Figure 10-57 Average High Priority Process Time

3. In the KPI Definition section, Figure 10-58 on page 478, set:
 - Monitor Context to OrderServiceFlow
 - Metric to OrderServiceFlow Average Transaction Duration for KPI
 - Aggregation Function to Average

KPI Definition
Specify how the value of the KPI is set.

KPI Value
Choose how the KPI will get its value:
 Base this KPI on a metric and an aggregation function.
 Write an expression to calculate this KPI based on existing KPIs

KPI Details

Monitoring context:	<input checked="" type="text"/> OrderServiceFlow
Metric:	<input checked="" type="text"/> OrderServiceFlow Average Transaction Duration for KPI
Aggregation function:	<input checked="" type="text"/> Average
Use values from:	<input checked="" type="radio"/> All model versions <input type="radio"/> Only this version of the model

Time Filter
Select a time period over which the KPI should be calculated.

Metric:	<input type="text"/>
Time period:	<input checked="" type="radio"/> None <input type="radio"/> Repeating <input type="radio"/> Rolling <input type="radio"/> Fixed

Data Filter
Select the metrics that you want to use to determine what values to use in the calculation. For example, if you London, you only want to use monitoring contexts where the value of the City metric is London.

Metric	Operator	Values	Case-sensitive
012... High Priority	equals	X+Y =? 1	<input checked="" type="checkbox"/>

Figure 10-58 Average High Priority Process Time properties

4. Create another KPI called Avg Shipping Time. Set the Type to Duration.
5. In the KPI Definition section, Figure 10-59 on page 479, set:
 - Monitor Context to OrderServiceFlow
 - Metric to Order Shipping Time
 - Aggregation Function to Average
 - Data Filter to Succeeded equals 1

KPI Definition
Specify how the value of the KPI is set.

KPI Value

Choose how the KPI will get its value:

- Base this KPI on a metric and an aggregation function.
- Write an expression to calculate this KPI based on existing KPIs

KPI Details

Monitoring context: * OrderServiceFlow

Metric: * Order Shipping Time

Aggregation function: * Average

Use values from: All model versions Only this version of the model

Time Filter
Select a time period over which the KPI should be calculated.

Metric:

Time period:

- None
- Repeating
- Rolling
- Fixed

Data Filter
Select the metrics that you want to use to determine what values to use in the calculation. For example, if you have a KPI called Average Price in London, you only want to use monitoring contexts where the value of the City metric is London.

Metric	Operator	Values	Case-sensitive
012 Succeeded	equals	X=Y =? 1	<input type="button" value="..."/>
34..			

Figure 10-59 Average Shipping Time properties

6. Create another KPI called Max Shipping Time For Gold Customers. Set the Type to Duration.
7. In the KPI Definition section, Figure 10-60 on page 480, set:
 - Monitor Context to OrderServiceFlow
 - Metric to Order Shipping Time
 - Aggregation Function to Maximum
 - Data Filter to Customer Type equals ‘GOLD’

KPI Definition
Specify how the value of the KPI is set.

KPI Value
Choose how the KPI will get its value:
 Base this KPI on a metric and an aggregation function.
 Write an expression to calculate this KPI based on existing KPIs

KPI Details

Monitoring context:	* OrderServiceFlow	<input type="button" value="Browse..."/>
Metric:	* Order Shipping Time	<input type="button" value="Browse..."/>
Aggregation function:	* Maximum	<input type="button"/>
Use values from:	<input checked="" type="radio"/> All model versions <input type="radio"/> Only this version of the model	

Time Filter
Select a time period over which the KPI should be calculated.

Metric:	<input type="text"/>	<input type="button" value="Browse..."/>
Time period:	<input checked="" type="radio"/> None <input type="radio"/> Repeating <input type="radio"/> Rolling <input type="radio"/> Fixed	

Data Filter
Select the metrics that you want to use to determine what values to use in the calculation. For example, if you have a KPI called Average Price in London, you only want to use monitoring contexts where the value of the City metric is London.

Metric	Operator	Values	Case-sensitive
Customer Type	equals	X=Y =? 'GOLD'	<input type="checkbox"/>

Figure 10-60 Maximum GOLD Order Shipping Time properties

Deploying the monitor model

Monitor models are J2EE applications that are deployed on to a WebSphere Application Server that is running the WebSphere Business Monitor Server. To use the monitor model that we created in the previous chapter, we must export it as a J2EE application:

1. In WebSphere Integration Developer, switch perspectives to the Business Monitoring perspective.
2. In the OrderMonitorProject, right-click the OrderMonitor.mm Monitor Model. Select the **Generate Monitor JEE Projects** option, as shown in Figure 10-61 on page 481. Accept the defaults that are provided in the prompt or supply your own project. Write down the JEE Application Project Name, and click **Finish**.

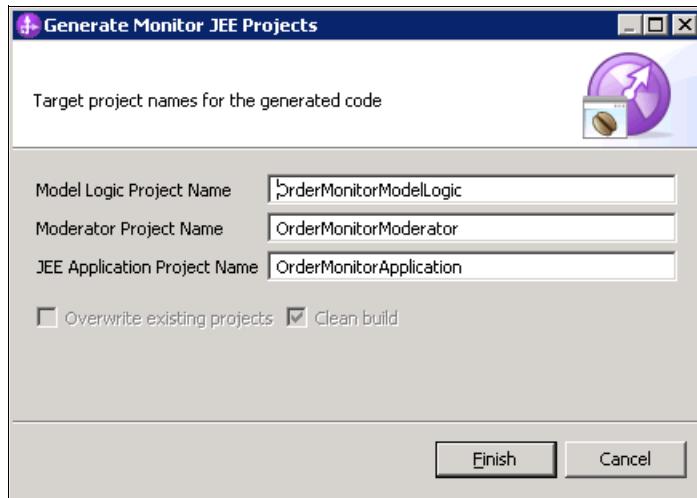


Figure 10-61 Generate JEE Projects prompt

3. Now, switch to the Java EE perspective.
4. Find the JEE Application project that you wrote earlier, right-click it, and select **Export → EAR File**. Confirm the EAR project. If you accepted the defaults, it should be OrderMonitorApplication. Select the destination file name OrderMonitorApp1.ear. Figure 10-62 on page 482 shows the Export prompt.

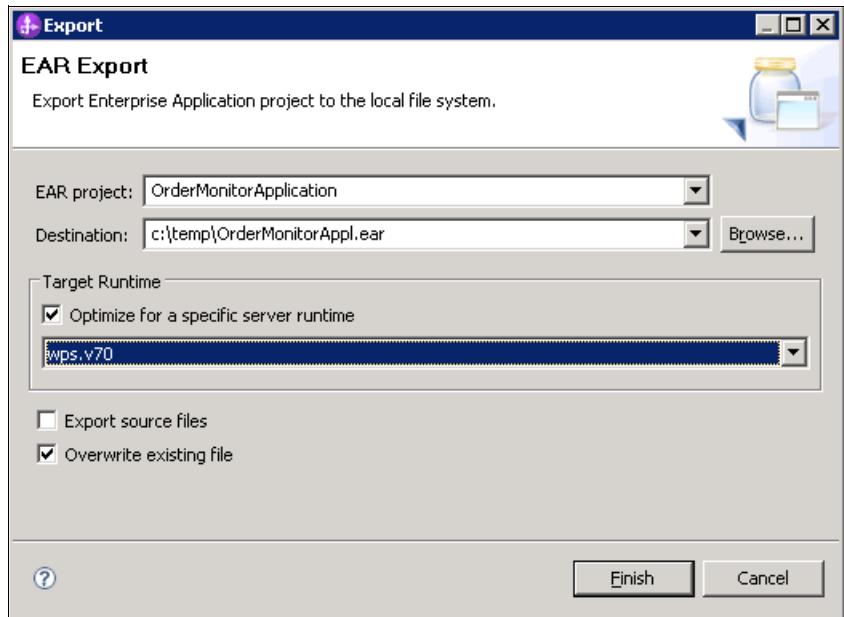


Figure 10-62 OrderMonitor model export

5. Select **Finish**.

Deploy the monitor model to the WebSphere Business Monitor server:

1. Go to the Monitor Server administration console:
 - a. Select **Applications** → **New Application** → **New Enterprise Application**.
 - b. Select the exported EAR file `OrderMonitorAppl.ear`, and click **Next** through the remaining prompts.
 - c. Review the summary, and click **Finish**. Monitor the install, and confirm a successful installation, as shown in Figure 10-63.
 - d. **Save** the configuration.

Application OrderMonitorApplication installed successfully.

Figure 10-63 OrderMonitorApplication Installation message

2. From the **Applications** → **Application Types** → **WebSphere enterprise applications**, select the application (default is `OrderMonitorApplication`), and click **Start**. Confirm the Application Status is now green.

10.8.2 Business Space

In this section, we discuss using the Web-based dashboard to view and monitor the model that was created.

Using a browser access the Business Space Manager:

`https://localhost:<WC_defaulthost_secure>/BusinessSpace`

1. Create a new space by navigating to **Actions → Create Space**. In the Space name, enter Garden Suppliers. Select a Space style and Space Icon. **Save** the completed space.
2. Select **Edit Page** to start adding widgets to view the monitor details.
3. On the Page 1 tab, use the pull-down menu to **Edit Settings**. Change the Page name to be Order Dashboard, as shown in Figure 10-64.

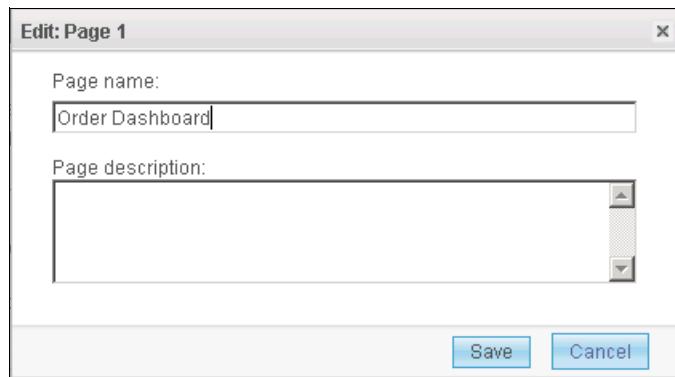


Figure 10-64 Order Dashboard page settings

4. Using the widgets panel, select the **Instances** widget, and click the + sign. Edit the newly added Instance by selecting the **Edit Settings** option, as shown in Figure 10-65 on page 484.

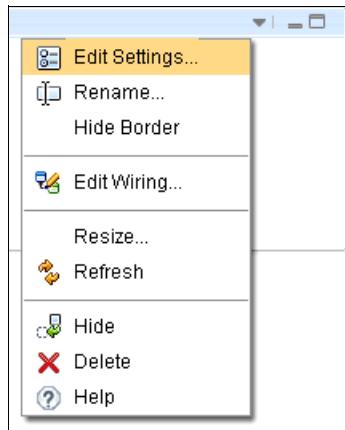


Figure 10-65 Edit Settings

5. Select the **Show/Hide** tab. Navigate OrderMonitor, and select monitoring contexts to personalize and select the OrderServiceFlow. Click **Set as Default**.

Security: If you see the message no monitor models are displayed, either no monitor models are installed or your user ID does not have authorization to view the model:

1. Using the Monitor Server administration console, navigate to the **Security → Monitor Data Security**, and click **Root**.
2. Select the Models in the Model pane. Select the KPI-Administrator in the Roles, and click **Users**.
3. Find the user that is being used, and move the user ID from the Available pane to the Select pane using the > icon. Click **Ok**.

4. Select all of the columns under Available. Click the >> icon to move them to the Selected list box. Click **Ok**.
5. Using the widgets panel, select the **KPIs** widget, and click the + sign. Select the Edit Settings option, and from the pull-down menu, edit the newly added KPI, as shown in Figure 10-65. Select the KPIs listed under the OrderMonitor_<date_timestamp>, as shown in Figure 10-66 on page 485.



Figure 10-66 KPI settings

6. Complete the dashboard editing by clicking **Finish Editing**. Figure 10-67 shows the completed Business Space.

KPI Name	Status	Value	Target	Actions	Value
Avg High Pri. Processing Time					
Avg Order Processing Time					
Avg Shipping Time					
Max Shipping Time For Gold Customers					

broker	COMPLETED	CreationTime	Customer ID	Customer Type	executionGroup	globalTransactionId	High Priority
							0

Figure 10-67 Garden Suppliers dashboard

For more detailed information about developing solutions on WebSphere Business Monitor and building monitor models refer to:

<http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp>

You can also review *Business Activity Monitoring with WebSphere Business Monitor V6.1*, SG24-7638.

10.9 Testing the scenario

In this section, we demonstrate some integration tests that exercise WebSphere Message Broker, WebSphere Process Server, and their monitoring capabilities.

The WebSphere Message Broker Toolkit Test Client is used to test the OrderServiceFlow message flow. To test the scenario:

1. To invoke the Test Client right-click the OrderServiceFlow, and select **Test Message Flow**, as shown in Figure 10-68.

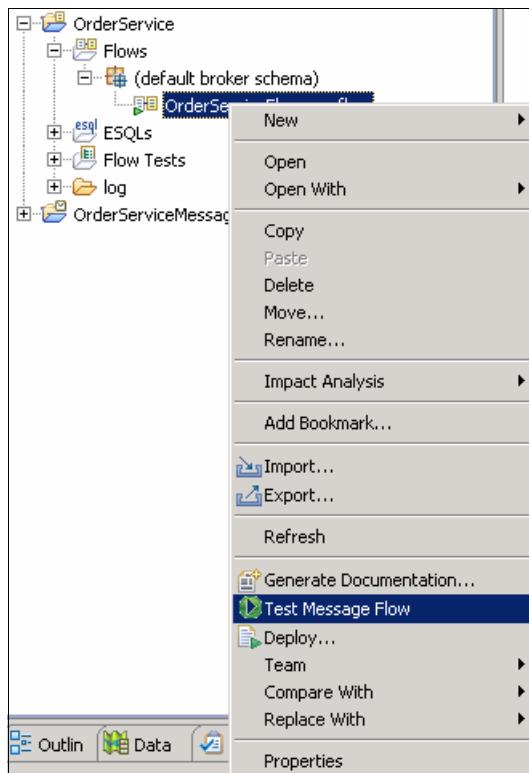


Figure 10-68 Test Message Flow

2. Right-click in the Message pane, and select **Add Message Part** to include a purchaseOrder message, as shown in Figure 10-69 on page 487.

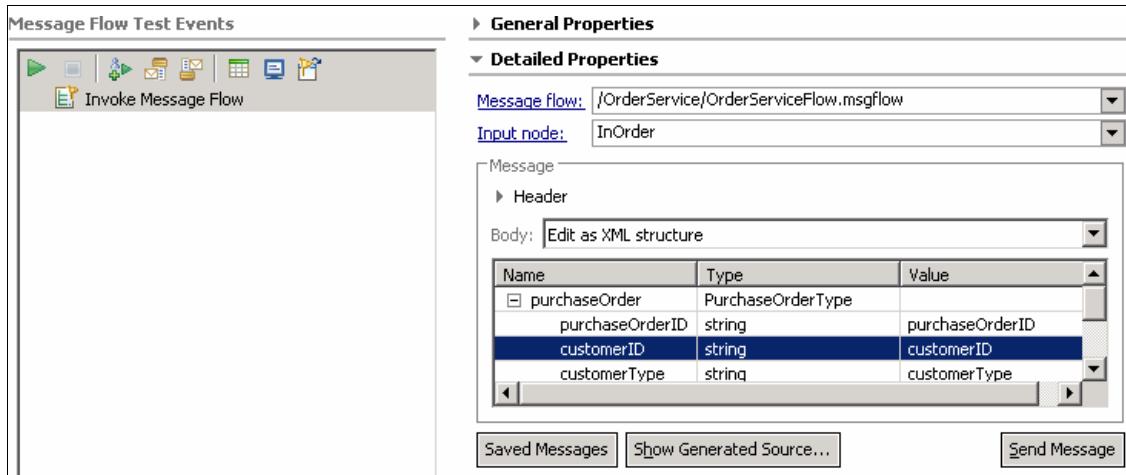


Figure 10-69 Test Client instance settings

3. Select the **Configuration** tab to update the MQ Message Headers Default Header. As shown in Figure 10-70, set the:
 - Reply to queue manager to MB7QMGR
 - Reply to queue name to ERROR_ORDER_Q

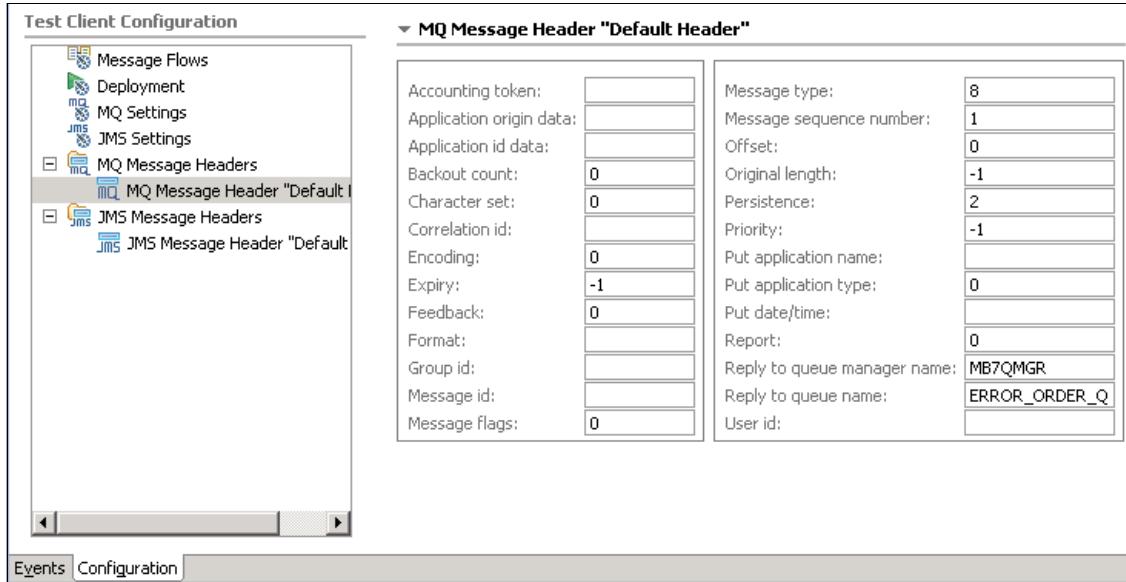


Figure 10-70 Test Client MQMD settings

The WebSphere Message Broker Test Client is configured by default to manage the deployment of the message flow BAR file automatically, which might deactivate monitoring on the message flow during deployment. It is recommended that you deploy your message flow manually, and set the Test Client configuration, as shown in Figure 10-71. Refer to 10.7.4, “Enabling monitoring” on page 468 for information to validate that monitoring is active for the OrderServiceFlow message flow.

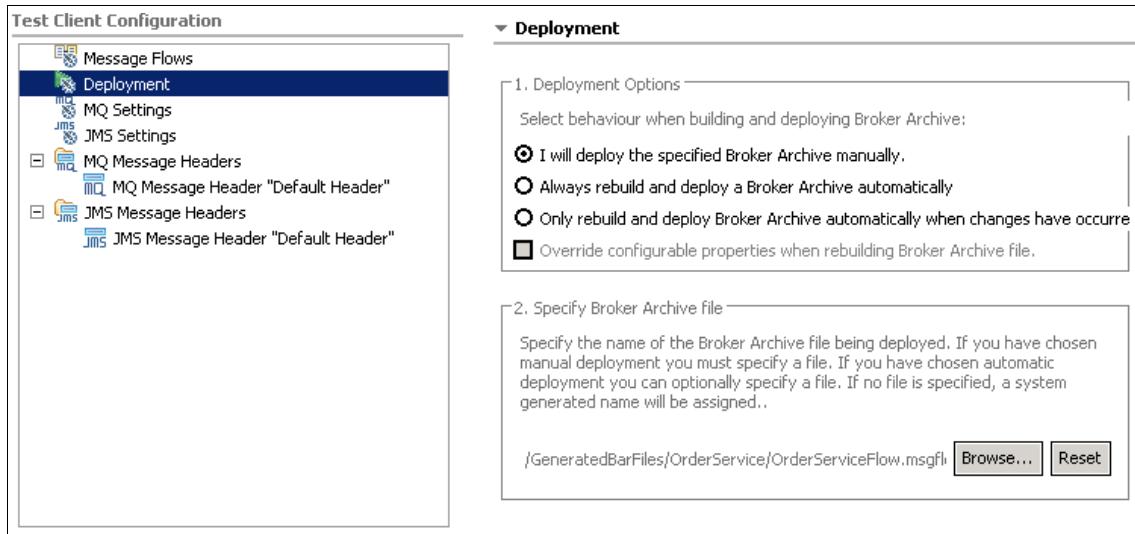


Figure 10-71 Test Client Deployment settings

4. Populate the data elements of the purchaseOrder, which includes setting a unique value for purchaseOrderId and customerId and setting the key values in Table 10-6. The test scenarios provided in this section are just examples that help highlight the different business criteria provided in the solution (refer to 10.3, “Scenario” on page 431).

During the execution of the test cases, monitor the *Garden Suppliers* Business Space that was created using
https://localhost:<WC_defaulthost_secure>/BusinessSpace

To populate the data elements of the purchaseOrder:

1. Send in a GOLD Order with the key elements shown in Table 10-6.

Table 10-6 - GOLD Order

Name	Value
customerType	GOLD

Name	Value
item[0].quantity	10
item[0].price	5.25

2. Repeat the GOLD Order test in step 1 on page 488 multiple times, changing the purchaseOrderID and customerID values as desired, which provides more data to the KPI averages that are being tracked in the monitor.
3. Send in a High Priority, GOLD Order with the key elements in Table 10-7.

Table 10-7 High Priority, GOLD Order

Name	Value
customerType	GOLD
item[0].quantity	175
item[0].price	25.50

4. Repeat the High Priority, GOLD Order test in step 3 a few more times, again changing the purchaseOrderID and customerID values.
5. Send in a Regular Order with the key elements in Table 10-8.

Table 10-8 Regular Order

Name	Value
customerType	REGULAR
item[0].quantity	3
item[0].price	125.75
item[1].quantity	1
item[1].price	20.10

6. Repeat the Regular Order test in step 5 a few more times, changing the purchaseOrderID and customerID values.
7. Send in a High Priority, Regular Order with the key elements in Table 10-9.

Table 10-9 High Priority, Regular Order

Name	Value
customerType	REGULAR
item[0].quantity	20

Name	Value
item[0].price	125.75
item[1].quantity	10
item[1].price	20.10

- Send in an Order that contains too many items for shipping with the key elements in Table 10-10.

Table 10-10 Order with too many items

Name	Value
customerType	REGULAR
item[0].quantity	150
item[0].price	125.75
item[1].quantity	100
item[1].price	20.10

- Access the Garden Suppliers Business Space to monitor the outcome of the test executions, as shown in Figure 10-72 on page 491.

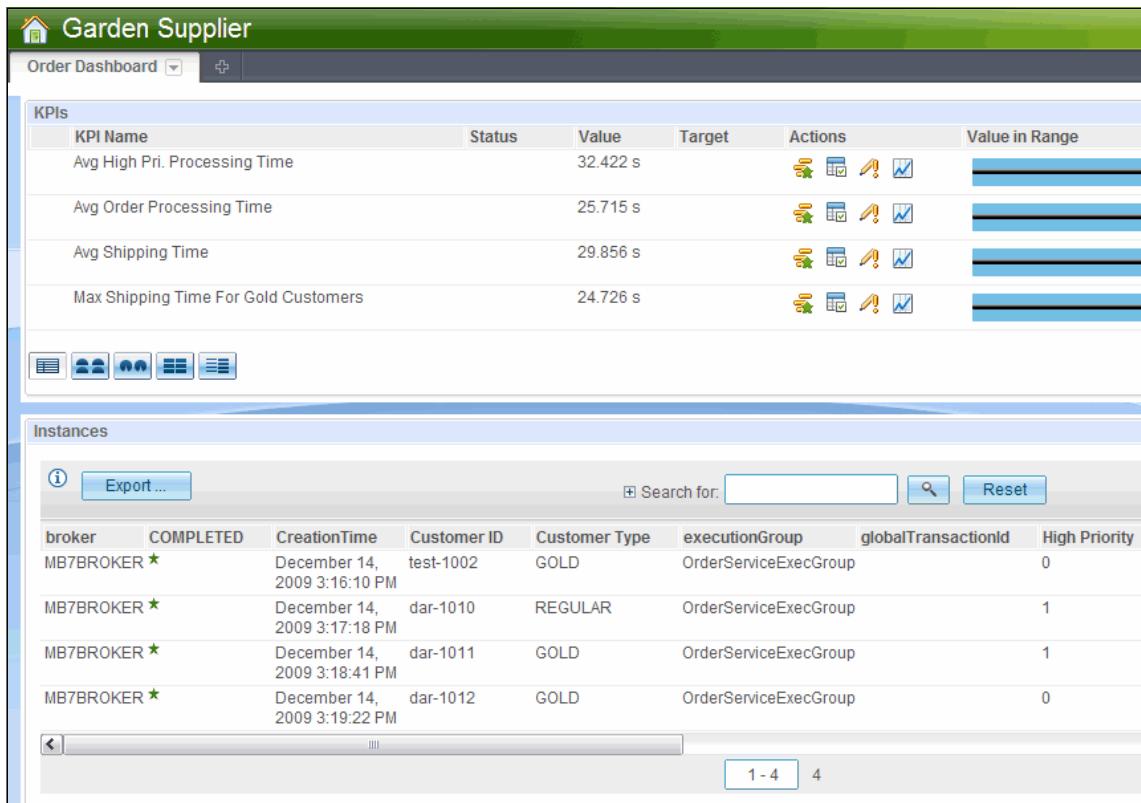


Figure 10-72 Garden Suppliers dashboard sample

10.10 Extension: Monitoring WebSphere Message Broker and WebSphere Process Server

In the original scenario, the shipping time was calculated using two events that were emitted from the OrderServiceFlow message flow - ShippingReview.terminal.in and ShippingReview.terminal.out. In this section, we extend the scenario to demonstrate monitoring of multiple event sources. The WebSphere Process Server business process, shippingServiceProcess, emits a set of events that can be used by an ExtendedOrderMonitor model to track the shipping services KPIs.

10.10.1 WebSphere Process Server events

In this section, we demonstrate how to enable the two events that are emitted from the *shippingServiceProcess* business process. The goal is to emit an event at the start of the business process and one at the end, so that duration can be measured:

1. Using WebSphere Integration Developer, open the *shippingServiceProcess* business process.
2. In the Receive action Properties section, click the **Event Monitor** tab, and enable the Exit Event, as shown in Figure 10-73.

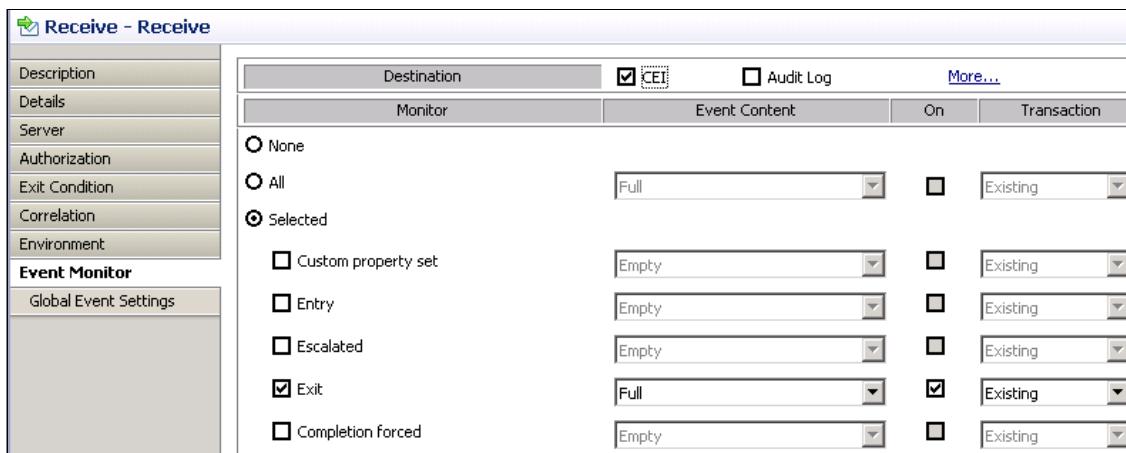


Figure 10-73 Receive activity monitoring properties

3. Reply action Properties section, click the **Event Monitor** tab, and enable the Exit Event, as shown in Figure 10-74 on page 493.

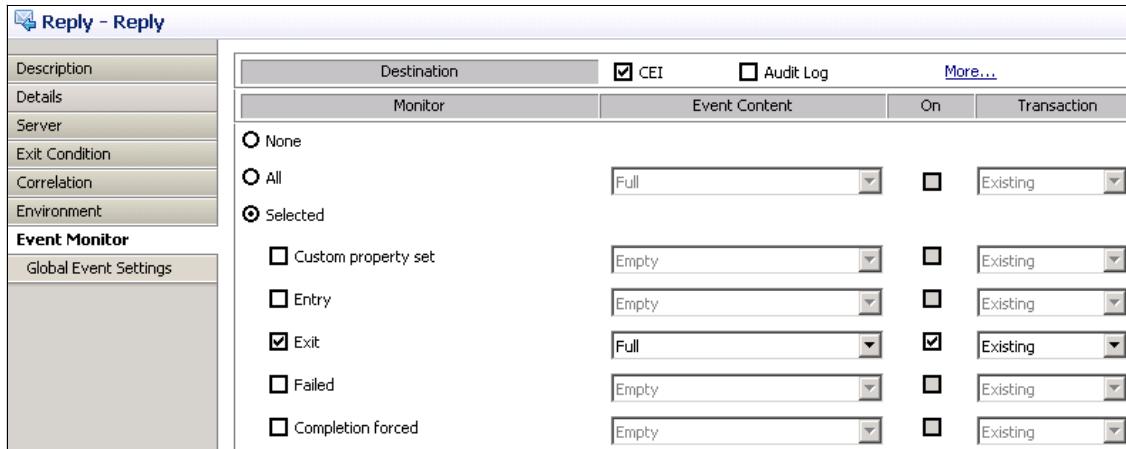


Figure 10-74 Reply activity monitoring properties

- Using the Generate Monitor Model feature that WebSphere Integration Developer contains, generate a monitor model for the ShippingServiceModule, as shown in Figure 10-75.

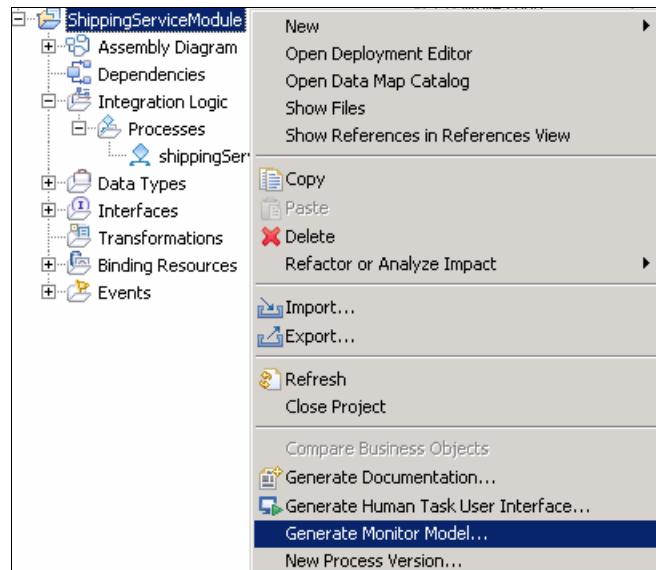


Figure 10-75 Generate Monitor Model

- Generate the model in the OrderMonitorProject with the model name of shippingMonitor, as shown in Figure 10-76 on page 494.

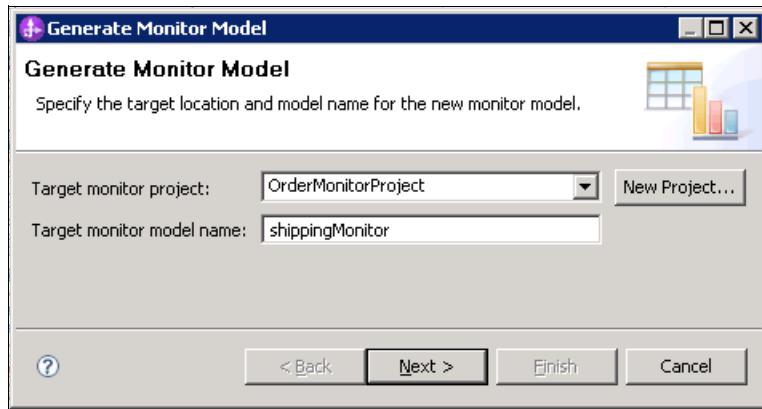


Figure 10-76 Generate Monitor Model properties

6. The Choose What to Monitor panel allows you to select from the available event sources the events that you want to monitor in this model. As a part of this exercise, select the following Emitted Events, as shown in Figure 10-77 on page 495:
 - The ReceiveExit event under **ShippingServiceModule** → **ShippingServiceModuleModule** → **shippingServiceProcess** → **shippingServiceProcess** → **Sequence** → **Receive**.
 - The ReplyExit event under **ShippingServiceModule** → **ShippingServiceModuleModule** → **shippingServiceProcess** → **shippingServiceProcess** → **Sequence** → **Reply**.

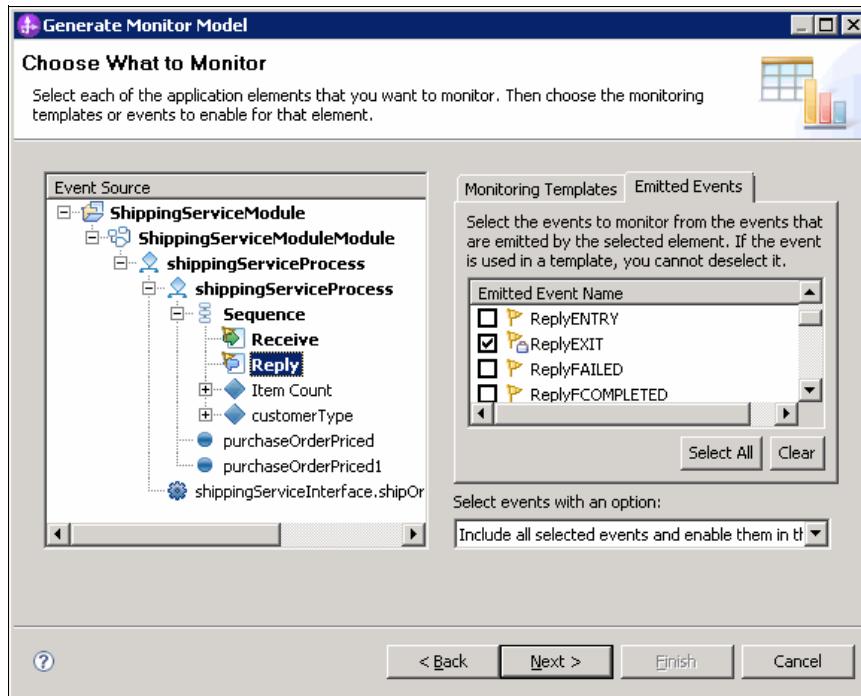


Figure 10-77 Choose What to Monitor properties

7. Proceed through the rest of the panels and review the content. Select **Finish** to complete the import.

10.10.2 WebSphere Business Monitor Model

Now we have two models that are describing similar aspects of the overall solution. In this section, we demonstrate combining those models so that we have a unified business model that leverages both sources of events.

As a part of the previous section, the Generate Monitor Model wizard, should have changed you to the Business Monitoring perspective in WebSphere Integration Developer. To combine the models:

1. Select the two models (OrderMonitor and shippingMonitor), and in the Project Explorer, right-click them, and select **Combine Monitor Models**, as shown in Figure 10-78 on page 496.

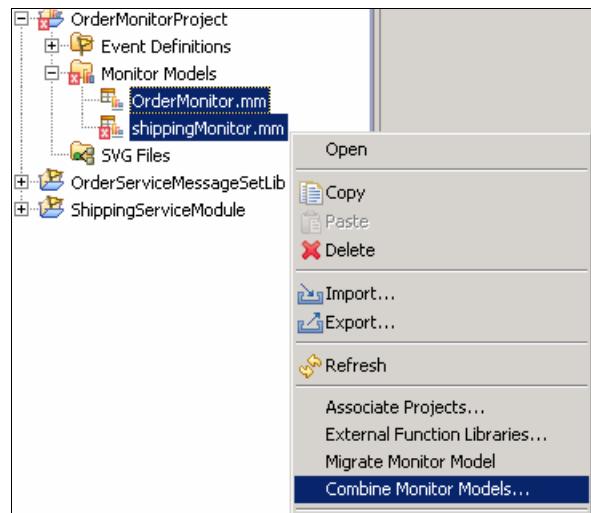


Figure 10-78 *Combine Monitor Models prompt*

2. Merge the results of the two models into a new monitor model named ExtendOrderMonitor under the parent folder OrderMonitorProject, as shown in Figure 10-79.

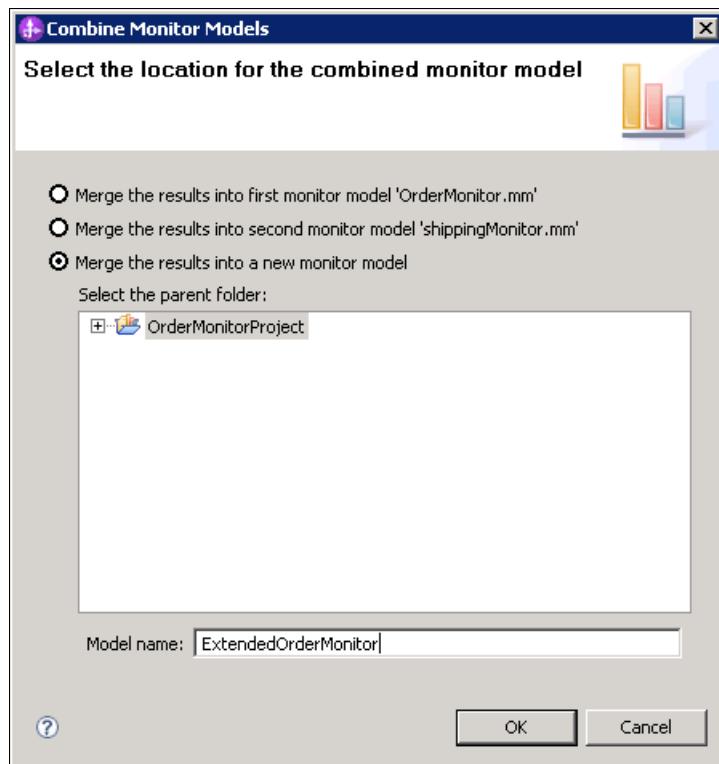


Figure 10-79 Combine Monitor Models properties

To integrate the events from the two models under a single Monitoring context:

1. Under the ExtendedOrderMonitor Receive monitoring context, right-click the **ReceiveExit** inbound event, and select **Copy**, as shown in Figure 10-80 on page 498.

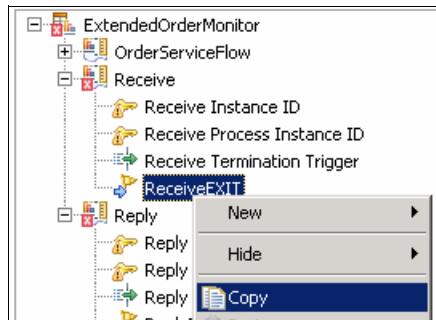


Figure 10-80 *ReceiveExit copy*

2. Right-click the **OrderServiceFlow** monitoring context, and select **Paste**, as shown in Figure 10-81.

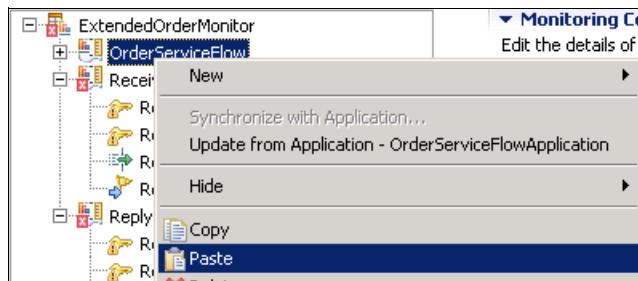


Figure 10-81 *ReceiveExit paste*

3. Repeat steps 1 on page 497 and 2 for the ReplyExit inbound event under **ExtendedOrderMonitor** → **Reply** monitoring context.
4. Now that inbound events are in the OrderServiceFlow monitoring context, the rest of the Receive and Reply monitoring contexts are no longer needed. Delete the Receive and Reply monitoring contexts.
5. Select the **Event Model** tab, and highlight the `/ShippingServiceModule/ExtendedPOInstance.xsd` element. Click **Remove**.

Modify the OrderServiceFlow monitoring context as a result of the new events. The event formats are different, so you must modify some of the assumptions made by the model generation tools. To enable the monitor model to support correlating the events from multiple event sources:

1. Select the **Monitor Details Model** tab.
2. Navigate to **ExtendedOrderMonitor** → **OrderServiceFlow** monitoring context. Delete the **Event Creation Time** XPATH value, as shown in Figure 10-82 on page 499.

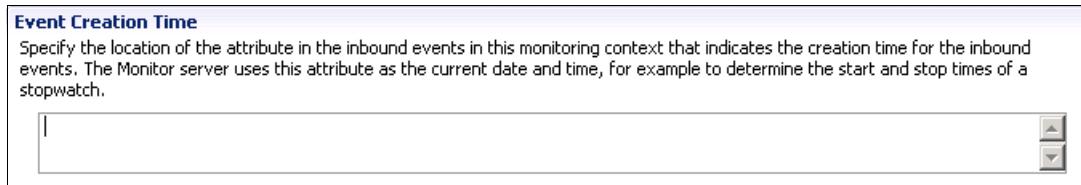


Figure 10-82 ExtendedOrderMonitor Event Creation Time

Some metrics in the model must be updated to support the new events. Because the events are not coordinated across the different event emitters, there are now two inbound events that can create a monitoring instance: the InOrder.transaction.Start and the ReceiveExit inbound events. As a result, the monitoring context's key must be updated to set the key value using one of the two events, which ever occurs first.

3. Add the Key Value Expression

ReceiveEXIT/purchaseOrderPriced/bo:transactionId to the localTransactionId key, as shown in Figure 10-83.

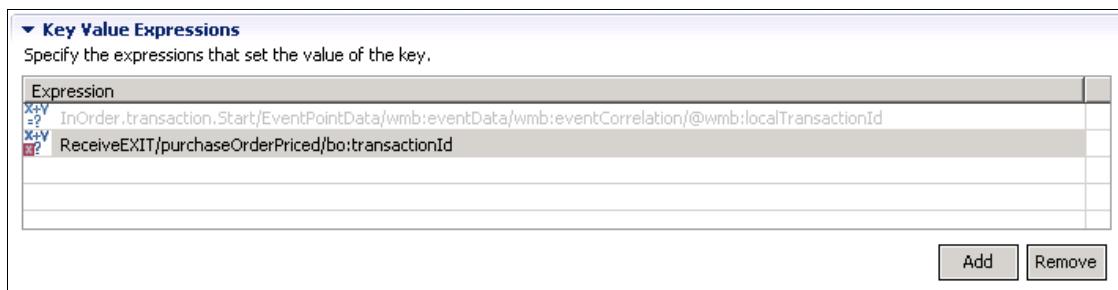


Figure 10-83 The transactionId properties

Note: The error that is generated on the new Key Value Expression change is a result of the default ReceiveExit inbound event configuration. We will correct this in step 4 on page 501.

The inbound events also need updating. The current definition for these events were generated by the templates in the WebSphere Process Server and WebSphere Message Broker tooling. As a part of the steps in this section, we update the Correlation Expression for the inbound events to simplify the correlation of events across event sources. All of the events are correlated around the localTransactionId of the WebSphere Message Broker events and the purchaseOrderPriced/transactionId found in the WebSphere Process Server events.

To update the inbound events:

1. Modify the InOrder.transaction.End Inbound Event:

- a. Remove the Application Element from the InOrder.transaction.End inbound event, as shown in Figure 10-84.

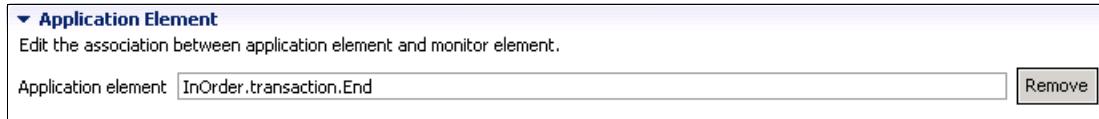


Figure 10-84 Application Element removal

- b. Modify the Correlation Expression value to be:

```
localTransactionId =  
InOrder.transaction.End/EventPointData/wmb:eventData/wmb:eventCorrelation/@wmb:localTransactionId
```

- c. Select the **Override inherited value from parent context** for **Event Creation Time** section, and set the value to be:

```
InOrder.transaction.End/EventPointData/wmb:eventData/wmb:eventSequence/@wmb:creationTime
```

2. Modify the InOrder.transaction.Rollback Inbound Event:

- a. Remove the Application Element from the InOrder.transaction.Rollback inbound event.

- b. Update the Correlation Expression value to be:

```
localTransactionId =  
InOrder.transaction.Rollback/EventPointData/wmb:eventData/wmb:eventCorrelation/@wmb:localTransactionId
```

- c. Select the **Override inherited value from parent context** for **Event Creation Time** section, and set the value to be:

```
InOrder.transaction.Rollback/EventPointData/wmb:eventData/wmb:eventSequence/@wmb:creationTime
```

3. Modify the InOrder.transaction.Start Inbound Event:

- a. Remove the Application Element from the InOrder.transaction.Start inbound event.

- b. Update the Correlation Expression value to be:

```
localTransactionId =
InOrder.transaction.Start/EventPointData/wmb:eventData/wmb:events
correlation/@wmb:localTransactionId
```

- c. Select the **Override inherited value from parent context** for **Event Creation Time** section, and set the value to be:

```
InOrder.transaction.Start/EventPointData/wmb:eventData/wmb:events
sequence/@wmb:creationTime
```

4. Modify the ReceiveExit Inbound Event:

- a. Update the Correlation Expression section, as shown in Figure 10-85:

- i. Set the value to be:

```
ReceiveEXIT/purchaseOrderPriced/bo:transactionId =
localTransactionId
```

- ii. Set the “If no instances are found” field to be Create new instance.

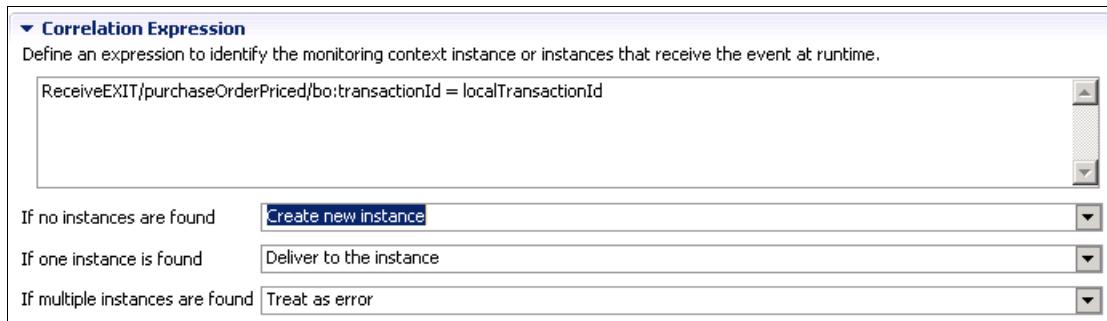


Figure 10-85 Correlation Expression properties

5. Modify the ReceiveExit Inbound Event:

- a. Update the Correlation Expression value to be:

```
ReplyEXIT/purchaseOrderPriced/bo:transactionId =
localTransactionId
```

6. To use the two events to monitor the shipping process duration, modify the Order Shipping Time stopwatch:

- a. Set the Stopwatch Controls to use the ReceiveExit and ReplyExit events instead of the ShippingReview.terminal.in and ShippingReview.terminal.out events, shown in Figure 10-86 on page 502.

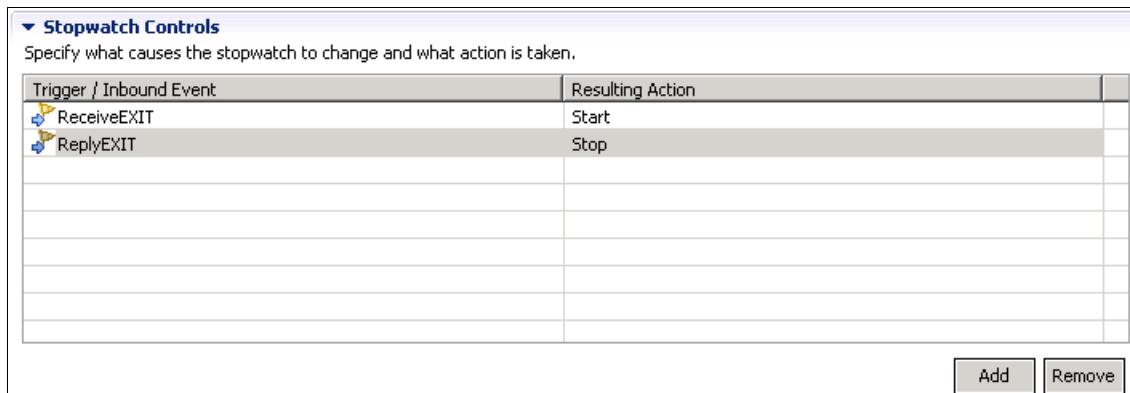


Figure 10-86 Stopwatch controls properties

The finished model should look like Figure 10-87.

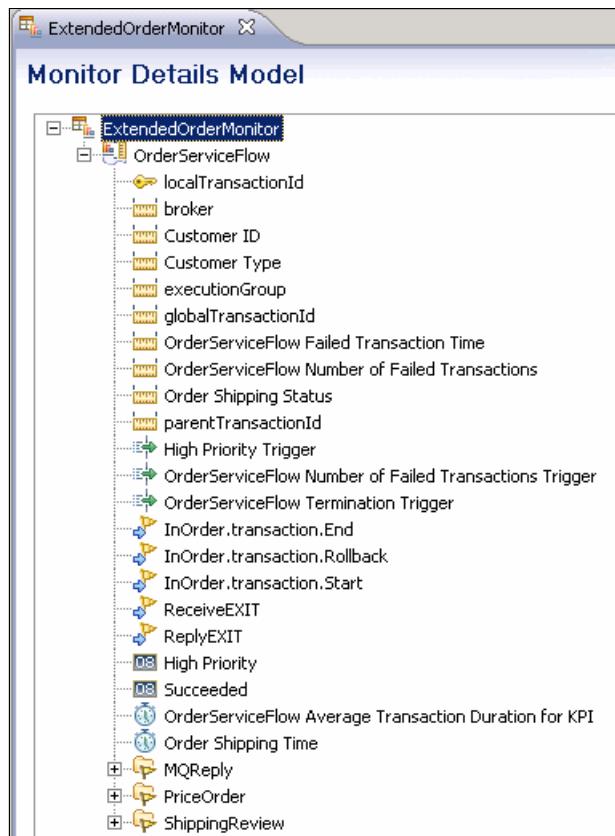


Figure 10-87 ExtendedOrderMonitor model

Generate and deploy the JEE project for the ExtendedOrderMonitor.mm model using the steps in “Deploying the monitor model” on page 480. Keep the JEE Project names distinct from the OrderMonitorModel. For this scenario, use ExtendedOrderMonitorApp1.ear as the EAR file name.

Using a browser access the Business Space Manager, and create a new page called Extended Order Dashboard in the Garden Suppliers Business Space created in 10.8.2, “Business Space” on page 483:

1. In the Garden Suppliers Space, click the + sign next to the Order Dashboard page to create a new page. Name it Extended Order Dashboard, as shown in Figure 10-88. Click **Ok** to complete the page creation.

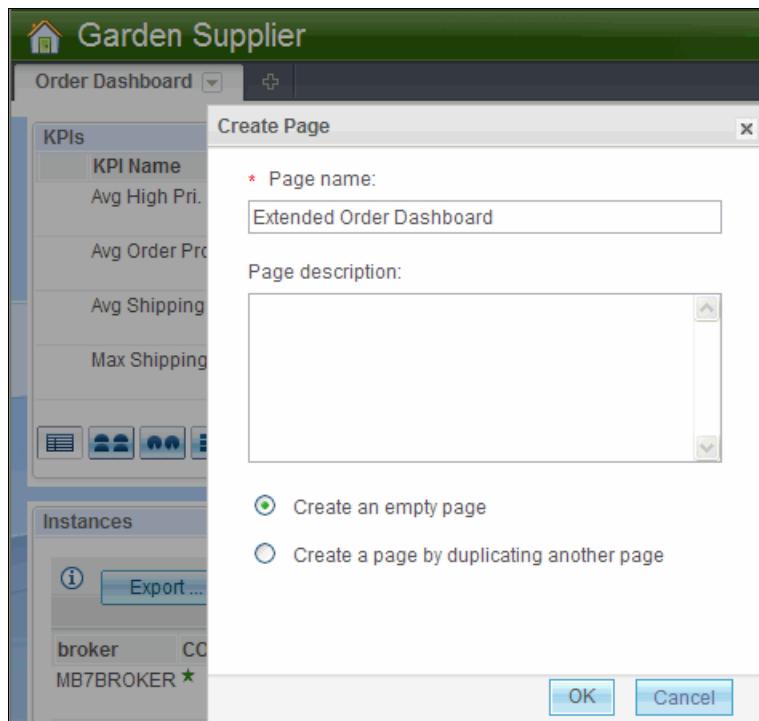


Figure 10-88 Extended Order Dashboard page creation

2. Using the widgets panel, select the **Instances** widget, and click the + sign. Edit the newly added Instance.
3. Select the **Show/Hide** tab. Drill down on the ExtendedOrderMonitor in the Select monitoring contexts to personalize, and select the OrderServiceFlow. Click **Set as Default**.

4. Select all of the columns under Available. Click the **>>** icon to move them to the Selected list box. Click **Ok**.
5. Using the widgets panel, select the **KPIs** widget, and click the **+** icon. Edit the newly added KPI by selecting the Edit Settings option. Select the KPIs that are listed under the `ExtendedOrderMonitor <date_timestamp>`, as shown in Figure 10-89.

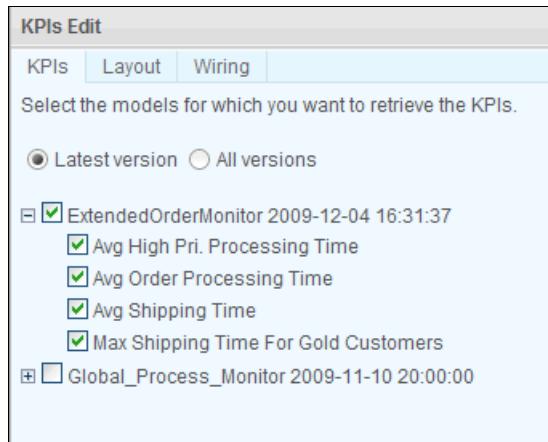


Figure 10-89 KPI settings

6. Complete the dashboard editing by clicking **Finish Editing**.

You can re-execute the test cases in 10.9, “Testing the scenario” on page 486 to exercise the solution again.

10.11 Additional details

In this section, we provide some additional comments that relate to the WebSphere Message Broker event monitoring support and other aspects about this scenario example.

10.11.1 Event sequencing

WebSphere Message Broker version 7 extended its event sequencing capabilities to include two elements: `CreationTime` and `Counter`. In some cases, the `CreationTime` was not granular enough to distinguish the arrival of multiple events. The `Counter` element is now generated automatically along with the

CreationTime element. The two elements reside in the following location in a WebSphere Message Broker event:

/EventPointData/wmb:eventData/wmb:eventSequence/@wmb:counter

/EventPointData/wmb:eventData/wmb:eventSequence/@wmb:creationTime

10.11.2 Audit capabilities

The capability to capture the entire message in an event is a part of the WebSphere Message Broker event format, which you can see in the Event Payload settings as a part of the configuration of a monitoring event. Each has the ability to capture the Headers, Body, or All of the message in event emitted. As shown in Figure 10-90, the message payload can be encoded into an event as either a base64 string, hexadecimal representation, or as a CDATA section of the message.

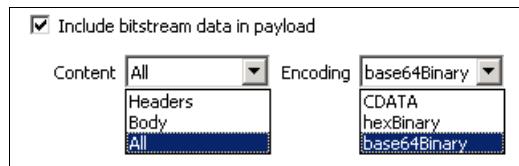


Figure 10-90 Event payload bitstream properties

One possible usage of this feature is to enable a consumer of these events and record them to a datastore. The datastore can then be used as audit reference point or as a repository for enabling message resubmission.

10.11.3 Managing the scenario

In this section, we discuss some common activities that you can use in support of this scenario.

Purging the monitor model instance data

As a part of testing and developing the monitor models, you might experience a situation where you want to remove the existing instance data that is associated with a monitor model:

1. Using a browser, access the WebSphere Business Monitor Administration Console:
<https://localhost:9043/ibm/console/login.do>
2. Click **Applications → Monitor Models**.

3. Select the version of the monitor model for which you are exporting or purging data.
4. Under Managing Monitor Data, select the **Purge and Archive Instance Data**.
5. Under the Time Filter section, select a date in the Purge Instances before field.
6. If you want to archive the data, populate a path to a directory for where you want the instance data stored.
7. Click **Purge**.

For more information about managing instance data, refer WebSphere Business Monitor Information Center at:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/topic/com.ibm.btools.help.monitor.admin.doc/data/man_instdata.html

Managing events

Events are the basis for publishing information in this scenario. As a consequence, often times, research into the events is necessary. The WebSphere Business Monitor server provides tooling in support of this need, including the ability to view events, record and playback events, and handle event sequencing issues.

Event management can be accessed through the Monitor servers administrative console by navigating to **Applications → Monitor Services → Recorded Event Management**.

For more information about managing instance data, refer WebSphere Business Monitor Information Center at:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/topic/com.ibm.btools.help.monitor.admin.doc/admin/managing_events.html



WebSphere Service Registry and Repository as governance

In this chapter, we discuss the integration of WebSphere Message Broker V7 with IBM WebSphere Service Registry and Repository (WSRR) to build dynamic SOA solutions. The concepts that we describe and demonstrate are:

- ▶ “Introduction to the WebSphere Service Registry and Repository” on page 508
- ▶ “WSRR nodes in WebSphere Message Broker” on page 509
- ▶ “Scenario” on page 519

11.1 Introduction to the WebSphere Service Registry and Repository

The WebSphere Service Registry and Repository is a central repository of documents (for example, WSDL, XSD) that describe services, service interfaces (for example, SOAP over HTTP), and associated policies that control access mechanisms (for example, WS-Policy documents associated with either of the previous two).

WebSphere Service Registry and Repository contains a variety of entities, such as WSDLs, XSDs, categories, relationships, their associated user properties, other service-oriented information, and metadata.

WebSphere Message Broker can use this information for arbitrary processing (for example, routing, dynamic transformation) in an up-to-date memory cache. The message flows in a broker can publish, find, enrich, manage, and govern services and policies in a service-oriented architecture (SOA) environment.

WebSphere Service Registry and Repository allows a message flow to dynamically retrieve artifacts from the repository at runtime and to use and expose them within the message flow, which allows you to defer the decision about which artifacts you want to use until runtime, rather than making the decision at deployment time.

WebSphere Service Registry and Repository enables SOA governance to establish decisions that are related to development, deployment, and management of services.

Generic XML documents, WSDL, SCDL, and all other formats that are supported by the WebSphere Service Registry and Repository can be stored in the repository. However, several queries that are submitted to the repository might only apply to certain document types, for example, a query for a port type can only be applied to WSDL documents.

WebSphere Message Broker provides mediation with advanced transformation and integration functionalities and is used as an enterprise service bus for connectivity of enterprise applications over a wide range of protocols and message formats. WebSphere Message Broker plays a key role in SOA implementations and acts as a transformation and connectivity engine.

Integrating WSRR with WebSphere Message Broker allows Message Broker to access services that are registered with WSRR at run time, thus allowing for dynamic connectivity between service consumers and service providers.

WebSphere Message Broker might also dynamically retrieve artifacts from WebSphere Service Registry and Repository at run time and use them in message flow processing.

WebSphere Message Broker supports secured connectivity with WebSphere Service Registry and Repository. HTTPS connections are secured using additional parameters that are available in the broker configurable services and SSL keystores and truststores. JMS connections can be secured, when CacheNotification is enabled, using the `mqsisetdbparams` command.

Note: For more information about connecting WebSphere Message Broker to secure WebSphere Service Registry and Repository, refer to:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac56150_.htm

11.2 WSRR nodes in WebSphere Message Broker

WebSphere Message Broker provides two built-in nodes to create the message flows that dynamically retrieve the service endpoints, or other data, based on the search criteria that is defined by the node properties from WSRR. The two WSRR nodes are:

- ▶ EndpointLookup node: This node retrieves the service endpoints for a WSDL service. It enables users to select single or all endpoints.
- ▶ RegistryLookup node: This node retrieves metadata about any entity that is stored/defined in WSRR, for example, WSDL, XML schema, XSLT, policy documents, and so on.

Besides XML schemas, XSD, XSLT, policies, Web Service Definition Languages (WSDLs), and so on, WebSphere Message Broker V7.0 also supports retrieving MQ Service endpoints from WSRR using RegistryLookup node.

Both the WSRR nodes can accept queries that are specified within the LocalEnvironment of the logical message tree. The properties and search criteria might be defined on the nodes or in the LocalEnvironment. When defined on the nodes, they can be overridden by setting the properties using the ESQL in the LocalEnvironment. A Compute node can be used prior to the WSRR node to override or clear the settings that are specified within the node itself.

Example 11-1 on page 510 shows examples of the ESQL that can be used to override properties in the node.

Example 11-1 ESQL to override properties in the node

```
1. SET LocalEnvironment.ServiceRegistryLookupProperties.Name  
= 'WSRR_WMB'  
2. SET LocalEnvironment.ServiceRegistryLookupProperties.UserProperties.  
ws-platform = 'win-wmb'  
3. SET  
LocalEnvironment.ServiceRegistryLookupProperties.MatchPolicy='All'
```

When deployed to the broker in a message flow, both of the WSRR nodes interact with WebSphere Service Registry and Repository in the following way:

- ▶ Receives message from the previous node in the message flow.
- ▶ Queries and sends a request to WSRR for information.
- ▶ Retrieves information (entity metadata or service endpoint).
- ▶ Executes matching algorithm to identify the provider service for the requesting service.
- ▶ The metadata is propagated to the message flow for further transformation and routing.

The resulting data returned from WSRR is an XML list. When the queried results are retrieved from WSRR, the resulting data contains a ServiceRegistry folder, that is owned by the XMLNSC domain, unlike previous versions, where the results were returned in the XML domain and had to be reparsed using XMLNS. The advantage of this is that it omits any unnecessary information, such as white space and correctly represents binary content as BLOBs.

Note: For more details about configuration and usage of EndpointLookup and RegistryLookup nodes refer to the WebSphere Message Broker V7 Information Center at:

<http://www publib boulder ibm com infocenter wmbhelp v7r0m0 topic com ibm etools mft doc ac56260>

The following RedPaper, redp4558, also discusses the WebSphere Message Broker WSRR nodes:

<http://www redbooks ibm com abstracts redp4558 html>

11.2.1 Query depth support by RegistryLookup node

The WebSphere Message Broker V7 implementation of the RegistryLookup node provides a lot of flexibility when returning matched and related objects as a query result by providing three distinct depth policy options:

- ▶ 0 returns just the matched object
- ▶ 1 returns the matched object and its immediate related objects
- ▶ -1 returns the matched object and all related objects

Figure 11-1 shows the Depth Policy that the RegistryLookup node supports.

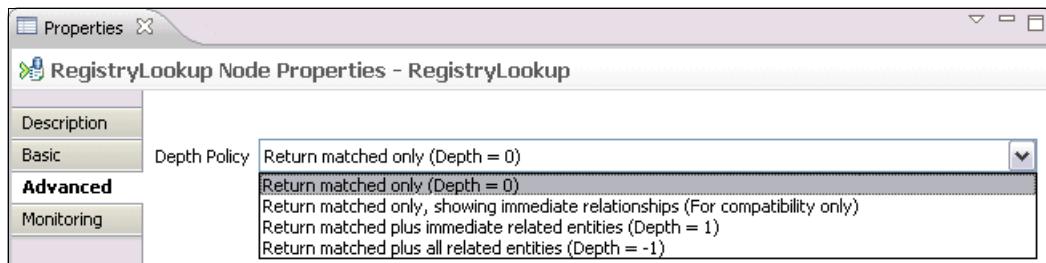


Figure 11-1 RegistryLookup node - Depth Policy

WebSphere Message Broker V6.1 implements a hybrid of depth policy '0' and '1'. To maintain backward compatibility, the property retains the WebSphere Message Broker V6.1 *hybrid* depth (marked as 'compatibility only') as an option in WebSphere Message Broker V7.

The data is retrieved according to search criteria that is defined by node properties, possibly supplemented, or overridden, by local environment definitions at run time. The node *Depth policy* property might also be overridden at runtime through the local environment (like other lookup properties):

`LocalEnvironment.ServiceRegistryLookupProperties.DepthPolicy`

11.2.2 WSRR configurable services

Configurable services define property values that are associated with message flow nodes directly at run time on the Execution Group or broker without the need to redeploy the message flow. The Execution Group or broker is required to restart after the change is made. The DefaultWSRR is a Service Registries configurable service that is supplied for each broker. The following command displays the property attributes of DefaultWSRR:

```
mqsi reportproperties MB7BROKER -c ServiceRegistries -o DefaultWSRR
```

Alternatively, the properties can also be viewed in Message Broker Explorer by right-clicking the configurable service ServiceRegistries/DefaultWSRR under Brokers, and clicking **Properties**. The Configurable services window is displayed, as shown in Figure 11-2.

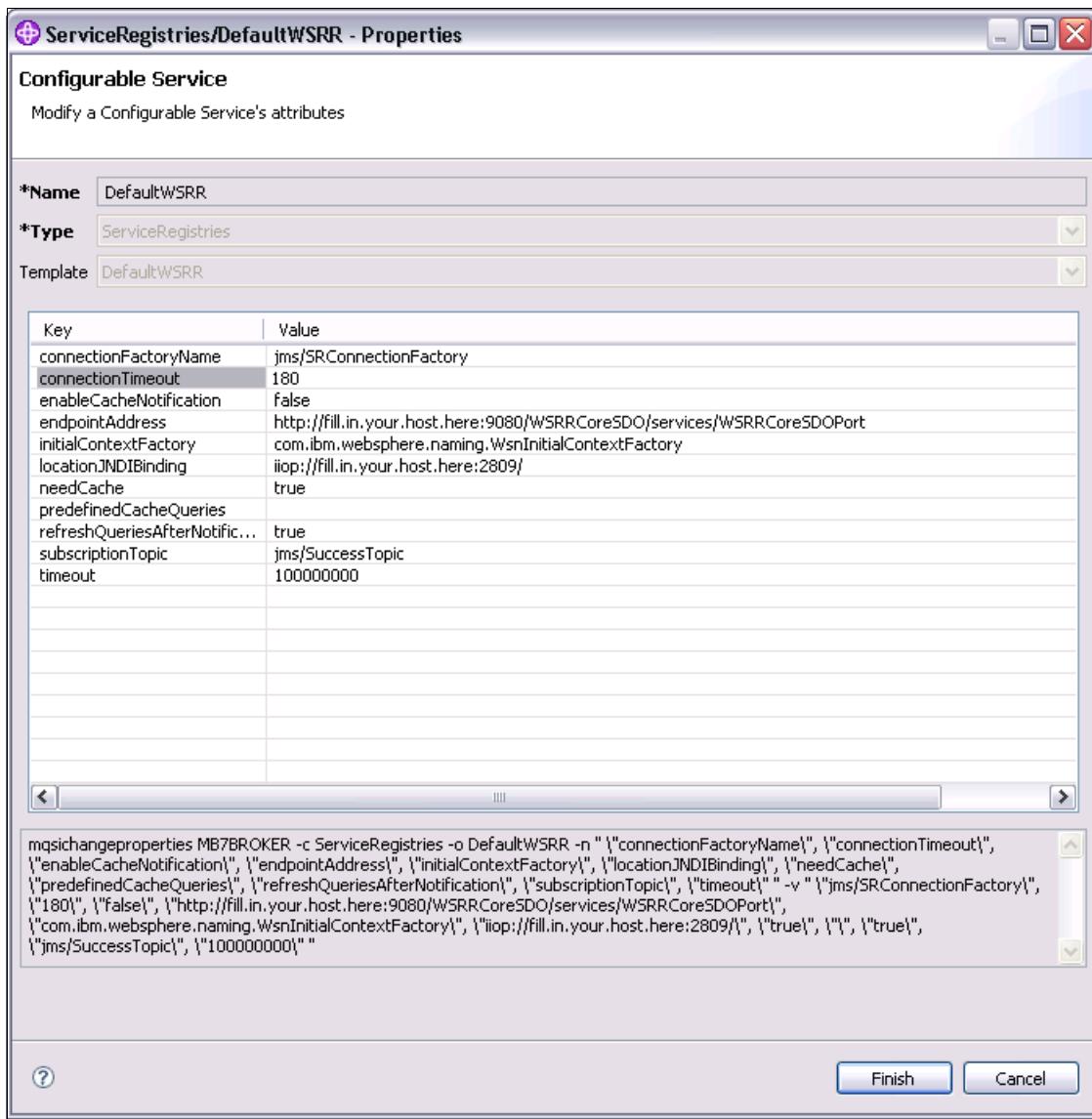


Figure 11-2 ServiceRegistries/DefaultWSRR configurable service properties

Broker WSRR cache

WebSphere Message Broker saves the data that is retrieved from the WebSphere Service Registry and Repository (WSRR) in a local broker WSRR cache. The Service Registry nodes (EndpointLookup and RegistryLookup) can retrieve the data that is stored in the broker WSRR cache by a query. The major advantage to this is improved performance and message throughput.

The first occurrence of each query is always sent to WSRR. At this point, the retrieved data is added to the cache using the WSRR query string as the cache key. Subsequent queries are then compared using the WSRR query string to the cache keys. If a match is found, then the cached WSRR results are returned; otherwise, if no match is found or the cache instance is invalidated because of query timeout settings, the query is resubmitted to the WSRR instance.

The node property settings generate the WSRR query string. So, when trying to optimize the query search, consider performance of the usage and property settings of the nodes (RegistryLookup and EndpointLookup nodes) to use the keying algorithm.

As shown in Figure 11-3 on page 514, you can configure the Message Broker WSRR cache with the parameter *needcache* using Message Broker Explorer or using the command:

```
mqsiclchangeproperties MB7BROKER -c ServiceRegistries -o DefaultWSRR -n  
needCache -v true
```

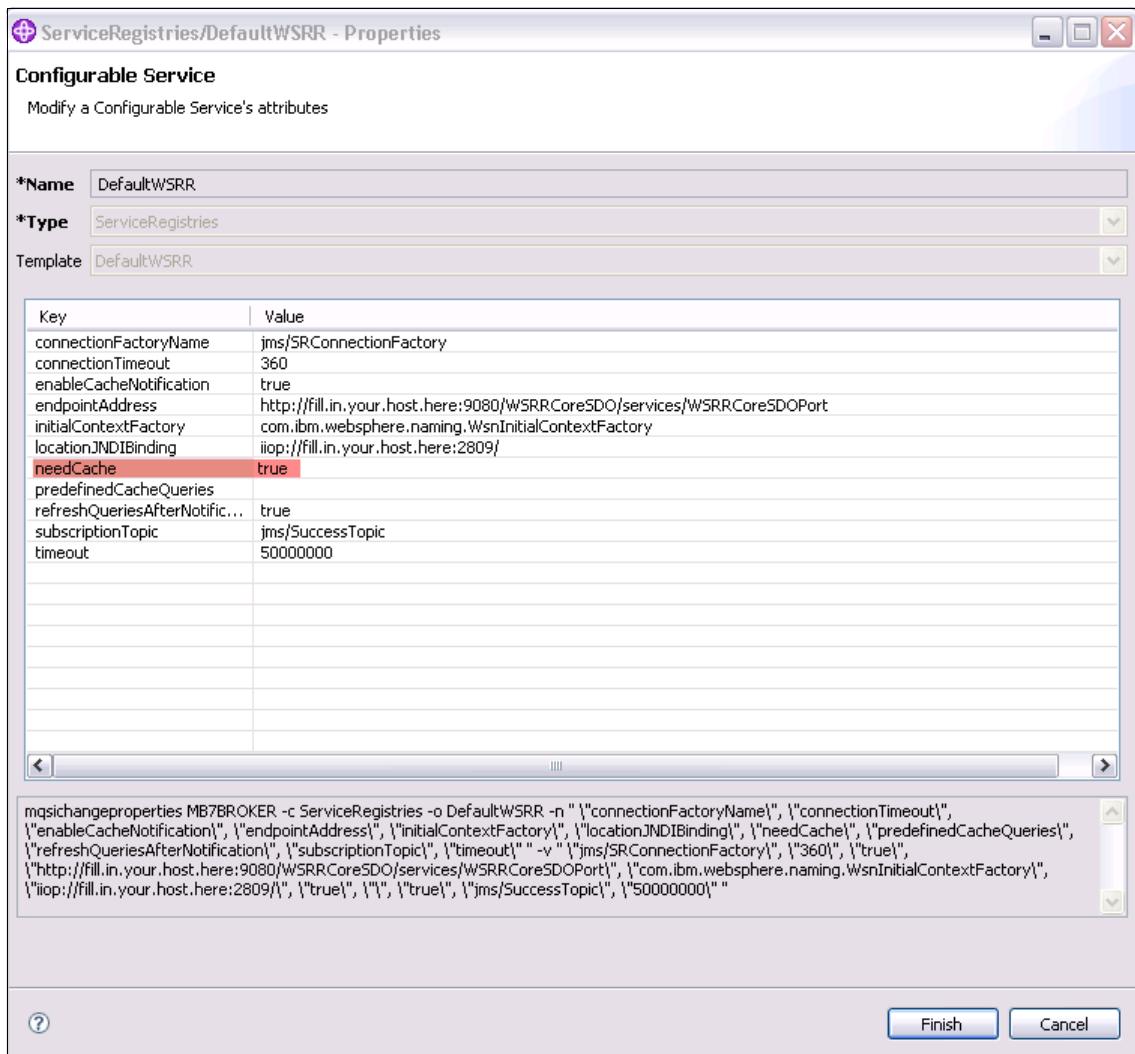


Figure 11-3 Configurable service 'needCache' in MB Explorer

Predefined queries

It is also possible to pre-populate the broker WSRR cache using the predefined queries. The predefined queries are executed when the broker starts or when a message flow that contains WSRR nodes is first deployed to the broker. This populates the cache and is then used by subsequent WSRR nodes. When executing the predefined queries, the broker startup might be slightly delayed, but at run time, the queries are executed directly from the cache, thus enhancing the run time performance.

In WebSphere Message Broker V6.1 these predefined queries always used DepthPolicy = -1, which means, '*Return match and all related entities*'. However, in WebSphere Message Broker V7, with the available options for the DepthPolicy parameter, the predefined queries might now be suffixed with a depth specifier of *0, 1, or -1*. The predefinedCacheQueries parameter is a list of WSRR XPath query expressions that are separated by semicolons, each with an optional depth specification. Predefined queries might be specified on the configurable service for Service Registry using the following command:

```
mqsicchangeproperties MB7BROKER -c ServiceRegistries -o DefaultWSRR -n predefinedCacheQueries -v "//*[@name="ConceptA1"];{depth=1}"
```

The *PredefinedCachequeries* parameter can also be modified from Message Broker Explorer under **Broker** → **Configurable Services** → **ServiceRegistries/DefaultWSRR Properties** → **PredefinedCacheQueries**, as shown in Figure 11-4 on page 516.

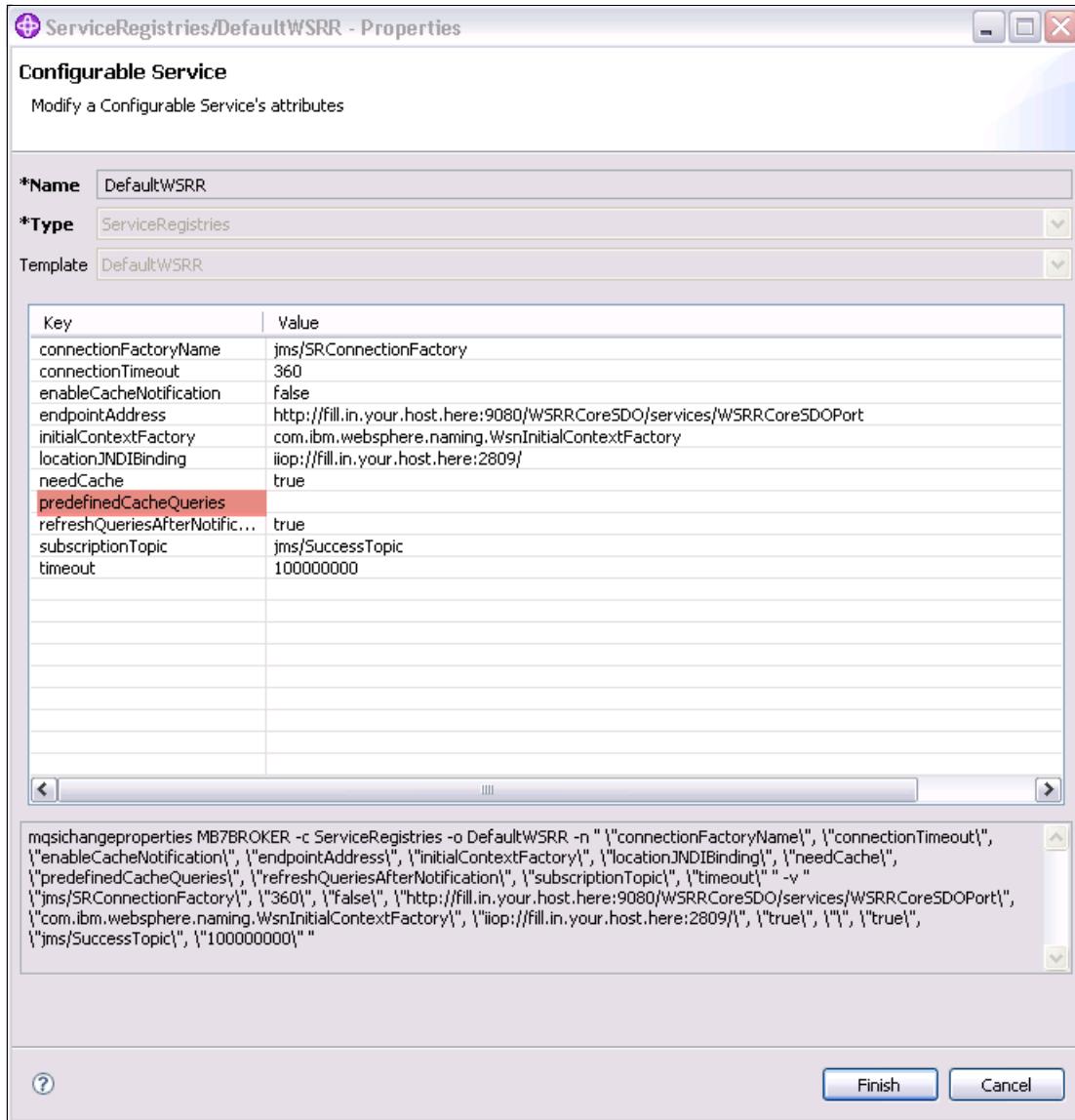


Figure 11-4 Editing PredefinedCache queries

Connection timeout

A system-wide connection timeout was added for WSRR queries that the EndpointLookup and RegistryLookup nodes issued. The connection created between the broker and a WSRR server is shared by all the message flows using EndpointLookup and RegistryLookup nodes. When querying WSRR, if any

exceptions are encountered on WSRR, to prevent the message flows from waiting indefinitely for WSRR to recover, the parameter *connectionTimeout* was added.

The Timeout is exposed as a new property *connectionTimeout* in the configurable service for WSRR. The *connectionTimeout* property can be modified using the following command:

```
mqsicchangeproperties MB7BROKER -c ServiceRegistries -o DefaultWSRR -n connectionTimeout -v 240
```

The parameter can also be configured using Message Broker Explorer by going to the Navigator view, right-clicking the **Configurable Service ServiceRegistries/DefaultWSRR** → **Properties** → **connectionTimeout**, as shown in Figure 11-5 on page 518.

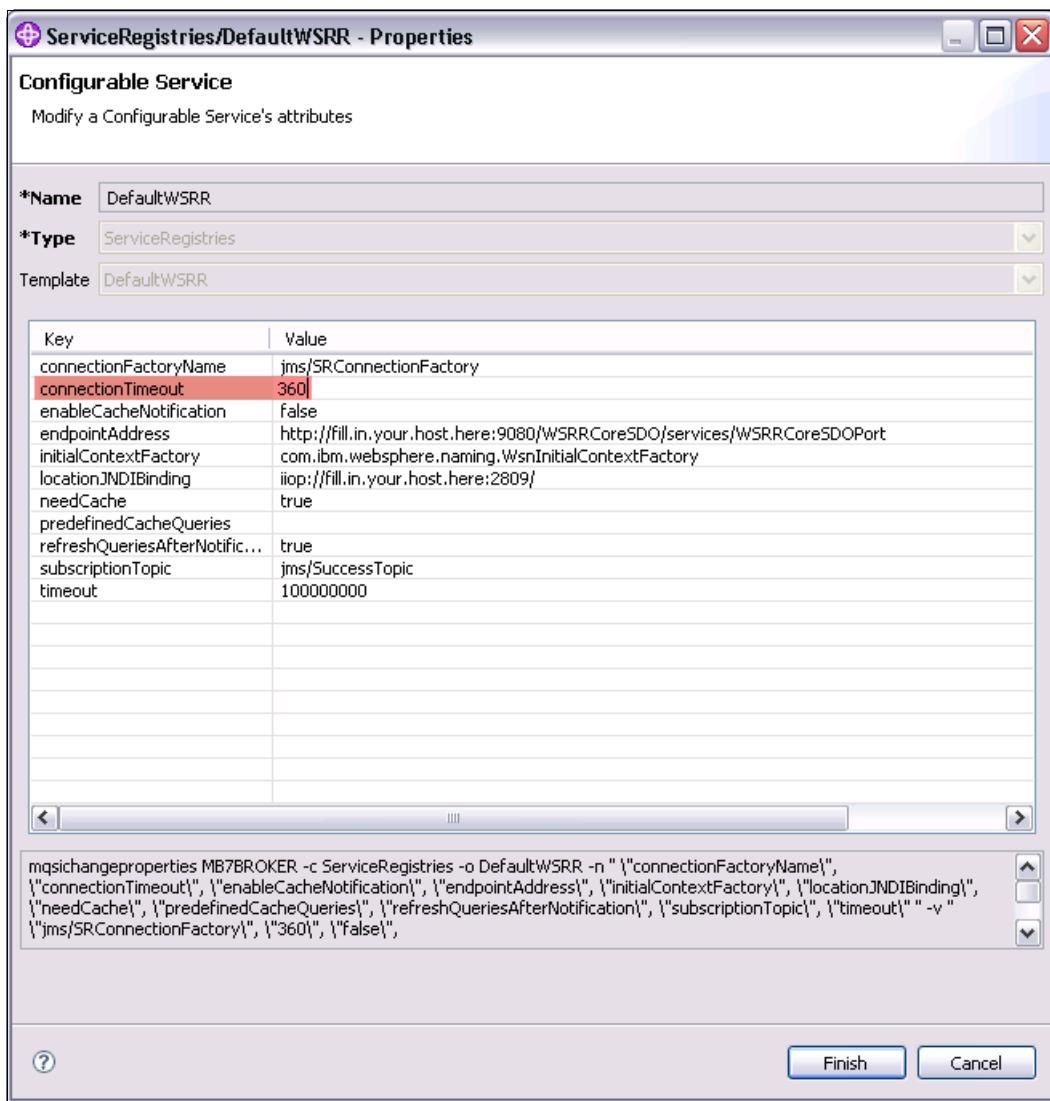


Figure 11-5 Configurable Service: connectionTimeout

There are other configurable service properties for ServiceRegistries/DefaultWSRR, such as Name, Type, connectionFactoryName, endpointAddress, initialContextFactory, refreshQueriesAfterNotification, enableCacheNotification, timeout (cache expiry), and subscriptionTopic, which you can modify using Message Broker Explorer or using the **mqsiclangeproperties** command.

WSRR configurable service properties: For more details about the WSRR configurable service properties, refer to WebSphere Message Broker V7 Information Center:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac56120_.htm

11.3 Scenario

The RedPaper *Integrating WebSphere Service Registry and Repository with WebSphere MQ and WebSphere Message Broker*, redp4558, provides the following two example scenarios:

- ▶ Creating an MQ Service Endpoint and manual MQ endpoint in WSRR to expose the MQ application as a service.
- ▶ Dynamically routing a service request to an available service endpoint using the EndpointLookup node.

WSRR-WebSphere Message Broker scenario and implementation: For a detailed scenario and implementation of WSRR-WebSphere Message Broker integration, refer to *Integrating WebSphere Service Registry and Repository with WebSphere MQ and WebSphere Message Broker*, redp4558, which is available at:

<http://www.redbooks.ibm.com/abstracts/redp4558.html>



Using WebSphere ILOG JRules together with WebSphere Message Broker

In this chapter, we introduce the concept of the Business Rule Management System (BRMS) and an IBM BRMS offering called WebSphere ILOG JRules. We then explain the integration scenarios and benefits with WebSphere Message Broker. Specifically, we discuss:

- ▶ “WebSphere ILOG JRules overview” on page 523
- ▶ “Benefits of using WebSphere Message Broker and WebSphere ILOG JRules together” on page 525
- ▶ “WebSphere ILOG JRules enables WebSphere Message Broker in three key business areas” on page 526
- ▶ “Integration approaches and scenarios” on page 531

12.1 Introduction to this chapter

The concept of Business Rules Management System (BRMS) is introduced with an overview of the IBM BRMS offering called IBM WebSphere ILOG JRules BRMS. We explain the integration benefits of two powerful solutions, one from the WebSphere Message Broker perspective and one from the WebSphere ILOG JRules perspective. We also help you explore various integration approaches.

12.1.1 Introducing the concept of Business Rule Management System

The main purpose of using a Business Rule Management System (BRMS) is to respond faster to business requests for policy changes by separating the life cycle of the rule deployment from the application deployment life cycle.

As companies rely more and more on information technology (IT) to manage their business, IT departments must develop more complex applications and simultaneously accommodate an increasing rate of change in the applications that they support.

Often, the implementation of the business policy in these applications becomes complex, voluminous, and too fast changing for a traditional software architecture. When the maintenance of an application that uses business logic becomes challenging, a BRMS provides solutions to make this management more efficient, both for developers and business users.

With a BRMS, developers and architects can externalize the business logic from the traditional application and develop and run the business logic independently of the application. In BRMS, testing the new rules is often done by the business users themselves, which is in contrast to the other software products, where businesses do not (or rarely) have any direct input or participation on the application testing by the business users.

A complete implementation of a BRMS can go even further and enable business users to manage business policies directly with limited dependence on the IT department. The degree of dependence can range from a limited review by business users of policies that developers implement, to complete control over the specification, creation, testing, and deployment of the policy by business users.

12.1.2 WebSphere ILOG JRules overview

WebSphere ILOG JRules is the IBM technology for creating, maintaining, and deploying decision services and business rules. It is also the IBM BRMS offering that enables both business users and developers to manage the rules that drive their business.

Business rules are an expression of business policy in a form that is understandable to business users and that can be executed by a rule engine. From a business perspective, a business rule is a precise statement that describes, constrains, or controls some aspect of users' business. From the IT perspective, business rules are a package of executable business policy statements that can be called from an application. A business policy can be expressed as several business rules. Business rules formalize a business policy into a series of "if-then" statements.

The WebSphere ILOG JRules suite provides two BRMS product lines: the JRules for Java and the JRules for .NET. Each provides a set of tools and environments that are specifically designed for the unique needs of the Java community and .NET community. In addition, the rules for the COBOL module extend the reach of the BRMS solution to the mainframe, System z. It provides a way to build rule artifacts from existing COBOL structures that are defined in copybooks. The rules are generated as executable COBOL code that can easily be integrated with existing System z-based applications that are running in CICS or IMS.

WebSphere ILOG JRules comprises a set of modules that operate in different environments but also work together to provide a comprehensive BRMS. A full WebSphere ILOG JRules BRMS product family consists of:

- ▶ IBM WebSphere ILOG JRules:
 - Rule Studio: Eclipse-based development environment
 - Rule Execution Server: Managed execution environment
- ▶ IBM WebSphere ILOG Rule Team Server: Business user rule management environment
- ▶ IBM WebSphere ILOG Decision Validation Services: Testing, Simulating, and Auditing functions that are integrated with Rule Studio, Rule Team Server, and Rule Execution Server
- ▶ IBM WebSphere ILOG Rule Solutions for Office: Guided authoring and editing of rules through Microsoft Office Word and Excel

Figure 12-1 on page 524 shows the different modules in the environment on which they are used and how they work together through synchronization and deployment.

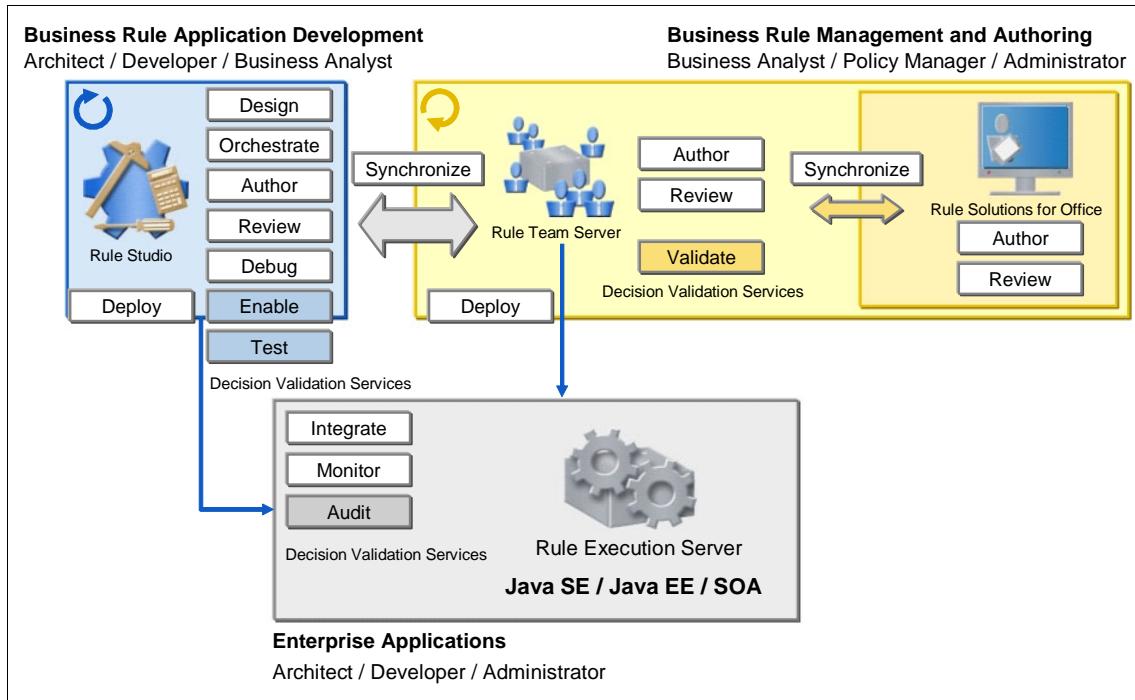


Figure 12-1 Usage of WebSphere ILOG JRules modules

An WebSphere ILOG JRules includes three primary components, as shown in Figure 12-2 on page 525:

- ▶ A repository that allows rules to be externalized from core application code. The repository allows decision logic to be managed as an enterprise asset, making it easier to understand and update decision logic. Along with consolidating decision logic from disparate applications and information silos so that it can be shared and re-used across the organization.
- ▶ Tools (Eclipsed-based and Web-based) that allow business experts to define and manage decision logic that was previously buried in code. These tools give business functions the ability to define an application behavior and also provides the ability for business and IT to work collaboratively on application development and maintenance.
- ▶ A runtime engine that allows production systems to access and execute decision logic that is managed within the BRMS. The “rule engine” allows complex and inter-related rules to be executed based on specific business context using a combination of data inputs, sets of applicable rules, and execution algorithms that define how to process the data and rules to provide an output.

A WebSphere ILOG JRules life cycle is also divided into two main periods:

- ▶ Build-time: When the application is constructed.
- ▶ Change-time: instead of run-time, users have change-time, when the application is deployed, available to its users, and continuously adapted to new and evolving business policies.

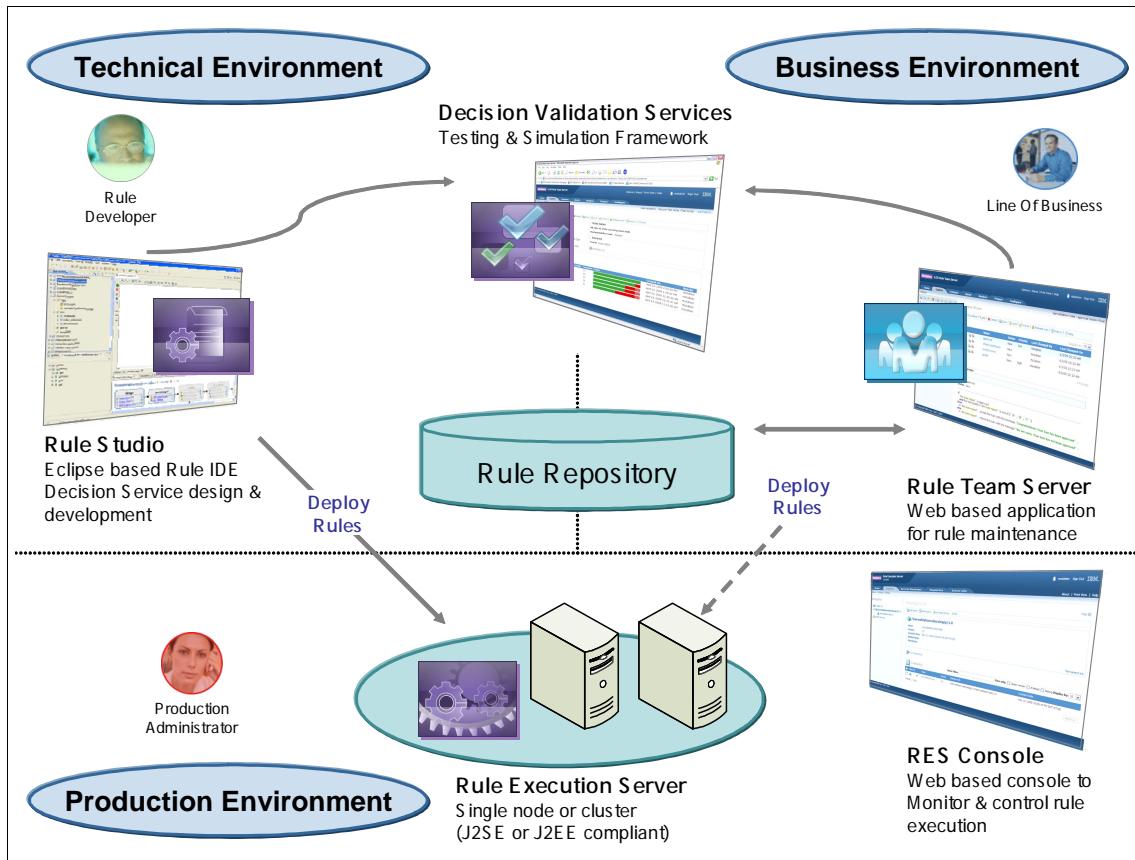


Figure 12-2 WebSphere ILOG JRules Architecture components

12.2 Benefits of using WebSphere Message Broker and WebSphere ILOG JRules together

Many business solutions today are reliant on the dynamic transformation and routing capabilities provided by Message Broker. Very often processing characteristics of the solution are based on functionality that is coded into

Message Broker flows. Within these flows reside decision points that govern the way that the solution responds to a particular request. Just as with business applications, good management of these embedded decision points is essential to creating a truly agile business. A BRMS in conjunction with Message Broker can provide a highly agile business solution at all levels from application business logic processing down to dynamic routing decisions and data transformation.

A Message Broker solution can also be used to augment and extend the capabilities of an existing BRMS system. It can provide more efficient integration rules with existing applications. It also can extend the reach of rules to additional applications, which allows multiple application types to share the benefits of rule execution under a BRMS.

In the following sections, we describe the benefits and approaches when each product is in conjunction with the other.

12.2.1 WebSphere ILOG JRules enables WebSphere Message Broker in three key business areas

In this section, we discuss the three key business areas where WebSphere ILOG JRules enables WebSphere Message Broker.

Specifying dynamic routing decisions in business terms

Using business rules to augment Message Broker routing decisions allows you to directly participate in authoring and change management that is associated with routing decisions.

Frequently the choice of destination can have business implications in terms of the cost of processing or the ability to maintain service level agreements. Making changes to these routing rules might be required to keep the business agile and competitive in the marketplace, as shown in Figure 12-3.

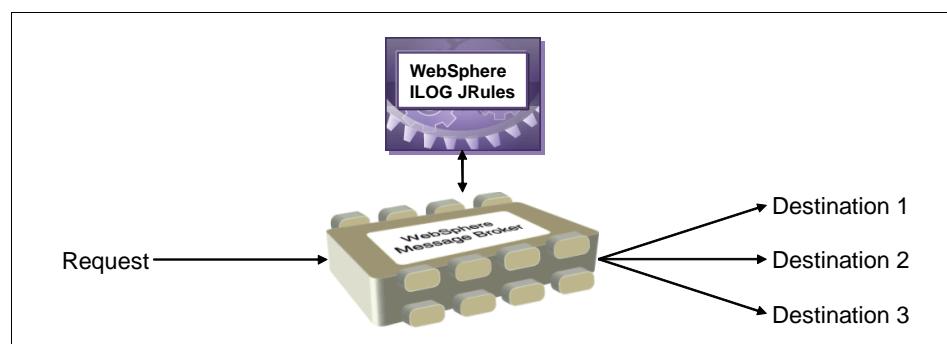


Figure 12-3 Dynamic routing with Message Broker and ILOG JRules

Scenario

A large Web-based electronics outlet can receive orders from any location in North America. When an order is placed and payment is received, a message is sent to a Message Broker-based flow to route the request to a fulfillment partner to supply the ordered item. The wholesale price of electronics equipment changes frequently between different suppliers. Also the cost of shipping a large electronic item to a customer is based on the physical distance between the supplier and the customer. To maintain its profitability, the selection of supplier must be carefully done to minimize both the wholesale purchase price and shipping costs.

To model this complex pricing model, the supplier selection decision is modeled in JRules. At runtime, the order request is received by Message Broker and details of the order are passed to JRules. JRules returns an indicator of the supplier to be used to fulfill this order. Message Broker then transforms the order request into the required format for that supplier and routes the request. Externalizing the rules also allows the business analyst to frequently update the rules that govern the supplier decision as wholesale prices and shipping costs change.

Augmenting and transforming messages based on business decisions

Often there can be information that is required in the message for the final consumer that cannot be supplied by the client. This information can be derived from the supplied message using business rules, as shown in Figure 12-4, for example, there might be an element of the final price that must be calculated individually for each customer based on their demographics: Customer Location = Iowa therefore tax = X%.

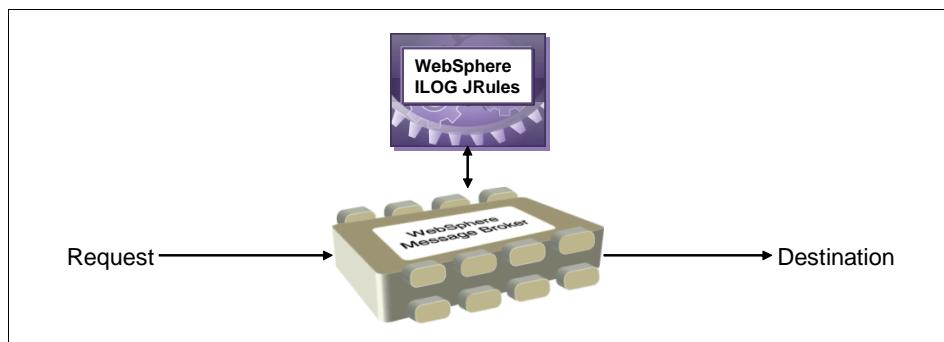


Figure 12-4 Data augmentation and transformation with Message Broker and ILOG JRules

Scenario

A large health care provider integrates with a number of partners to deliver its services. Reports and records of services provided must all be stored and maintained in government-defined standard document formats. At some point, the document standard moved from version 7 to version 8. The healthcare provider decides to move to the new format on a particular date. Unfortunately, not all of the partners can migrate their systems to the new format by the date that the provider decided. The provider must therefore support both the old and the new document formats for a period of time. The integration with the healthcare partners is currently based on a Message Broker-based infrastructure.

To minimize the complexity of support for both the old and new document formats simultaneously, the decision is made to encapsulate the data transformation within Message Broker. To expedite the development of the mapping routines, the complex mapping rules are modeled in JRules, which best enables the business analysts to validate that the correct mapping rules are applied in all situations to ensure the integrity of the data. At runtime, the transformation of the document from one format to another is shared between Message Broker and JRules.

Providing business-level validation rules for messages

Certain values of fields might only be valid in conjunction with particular values or other fields. If you process an insurance claim that has a field called Claim code, a value of XYZ123 can only be valid if the claimant is over 24, the value of the claim is less than \$5,000, and the claim was filed in the state of Iowa, as illustrated in Figure 12-5.

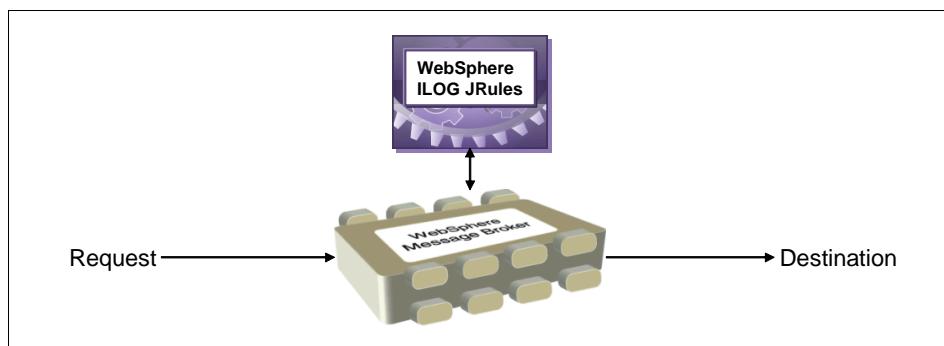


Figure 12-5 Message validation with Message Broker and ILOG JRules

Scenario

A large insurance company insures clients across a diverse geographical area with a large number of variations in policy clauses. To reduce operational costs, it

is centralizing its claim processing centers. Claim requests are routed from the geographical data centers to the central claim processing center with a Message Broker-based backbone. To reduce processing cost of invalid claims, the company wants to reject invalid claims at the first possible opportunity.

To achieve this requirement, the automatable business rules that are associated with the insurance claims are modeled in JRules. At the runtime on receiving a message, Message Broker makes a call to JRules to check the validation of the claim. If a claim is deemed to be invalid, it is routed back to the geographical data center for processing and no further action is needed.

12.2.2 WebSphere Message Broker enables and extends ILOG BRMS connectivity

Message Broker can enhance a rules-based solution, WebSphere ILOG JRules, by providing connectivity between clients and rule servers. We provide some key examples in this section.

Enrich data augmentation and decision request for rule invocation

You can use Message Broker to augment requests for rule execution, for example, a decision must be made to determine a discount to apply to a particular customer order. To make that decision, this customer requires the historical information about previous orders. This information is probably not available in the client application; therefore, Message Broker can be used between the client and the decision service to augment the request for a discount code by performing a database look up to enrich the request message with the historical order information transparently to the requesting client, as illustrated in Figure 12-6.

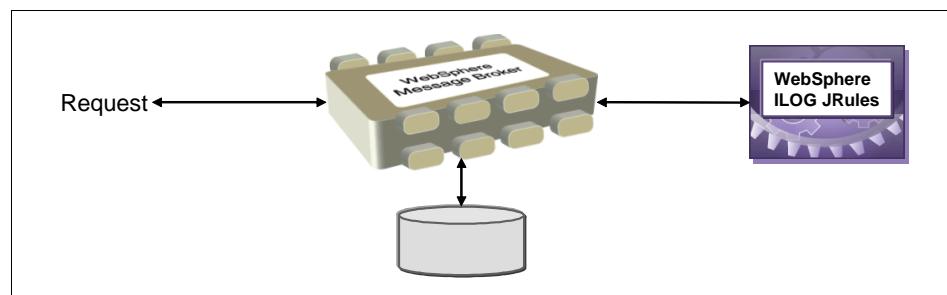


Figure 12-6 Database lookup for decision request with Message Broker and ILOG JRules

Scenario

An online store wants to increase its revenue by ensuring that loyal customers always return when they want to make additional purchases. To achieve this, the store offers promotional discounts and incentives at checkout time. The decision on what level of discount or incentive to apply is complex based on the frequency and sizes of previous orders and the types of items ordered.

The rules that govern the promotional scheme are modeled in JRules. When a customer places an order, the rules must be executed to return a promotion code to apply. However, the rules require historical order data that is not available in the commerce application that runs the Web-based shop. A Message Broker flow is placed between the request from the commerce application and the invocation of the decision service to add the historical order information for the customer to the request.

Message transformation for rule invocation

Using a Message Broker between the decision requester and ILOG JRules enables the request to be transformed into a format that can be consumed by the JRules application. The powerful data mapping capabilities of Message Broker can then be used to extend the reach of the JRules decision service to a wide variety of disparate requesting clients simultaneously. Message Broker can also provide the interfaces to the client application and the JRules server, as shown in Figure 12-7.

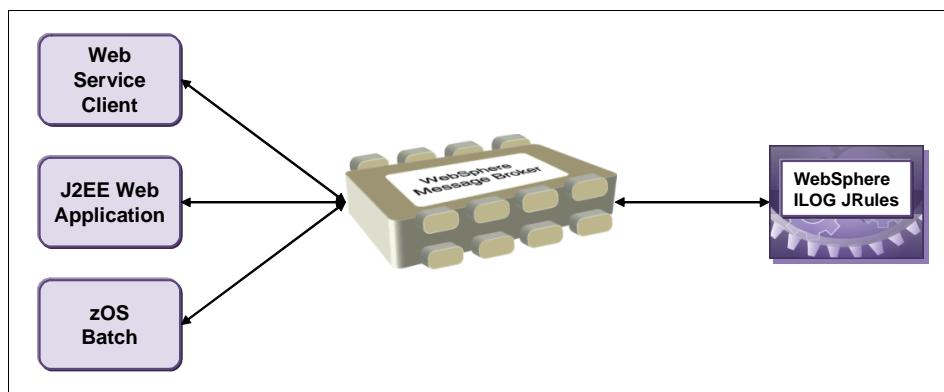


Figure 12-7 Decision request transformation with Message Broker and ILOG JRules

Scenario

A large insurance company has a large investment in its mainframe CICS infrastructure to implement its underwriting application for pricing of insurance quotes. With the advent of price comparison Web sites, the insurance company wants to provide a dedicated J2EE Web service application for servicing

requests for quotes from these partners. It must ensure that the price of quotes that are supplied to the comparison Web sites match those in its core business in the CICS application.

To achieve this and to gain more business agility, the choice is to extract the pricing rules from the CICS application and host them in JRules on System z. The CICS COBOL applications are re-engineered to make an MQ call to Message Broker, which transforms the COBOL data structure of the request to XML before calling the JRules decision service. The new Web service application is written to call the Message Broker, which converts the supplied XML into the correct format before calling the decision service in JRules. This solution allows both a COBOL-based System z application and an XML-based Web Services J2EE to share the execution of the same decision service, ensuring that the same pricing results in each case.

12.3 Integration approaches and scenarios

Currently, IBM recommends several possible approaches to integrate Message Broker and ILOG JRules. Start with the architecture design review or workshop first, and then select the best integration approach to fit into each scenario.

12.3.1 Using JRules through Web Services

Message Broker for Web Services makes an external call to a Rule Execution Server. This approach is recommended when Message Broker is used as a connectivity layer to connect to the WebSphere ILOG JRules rule servers, and it gives the benefit from the flexibility of accessing rules from multiple end-points, with no additional development effort.

Scenario

Using a SOAP node in Message Broker can:

- Call a Rule Application Server as a Web Service either through:
 - A hosted transparent decision service in JRules
 - Native Web Service in Application Server
- Full capability of the J2EE JRules Execution Server is available

12.3.2 Message Broker calling JRules through JMS

This approach uses Message Broker to support for JMS to make an external call to a Rule Execution Server.

Scenario

Using JMS Output and JMS Input nodes in Message Broker to call JRules that are hosted in Application Server through a Message Driven Bean (MDB) that can:

- ▶ Support full capability JRules Execution Server
- ▶ Support loosely coupled decisions and rules can be shared and re-used across other applications

12.3.3 Message Broker calling JRules J2SE Rule Execution Server

Message Broker provides the capabilities to incorporate Java code within Java Compute node and Custom node. JRules provides a Rule Execution Server (RES) environment for hosting in a standard J2SE environment. Message Broker can be configured as such that it can host a J2SE RES within its internal JVM environment. Using the standard POJO interface to JRules calls can then be made directly from a Java Compute node or a Custom node to the RES hosted in the same JVM. This approach deploys a JRules Rule Execution Server (RES) in a Message Broker JVM and accesses the rules using the Rule Session API. It gives the best performance because there is no overhead and latency that is associated with leaving the Message Broker environment to call the JRules functionality.

Scenario

This approach uses the Java Compute (or Custom) node in Message Broker to host JRules Rule Execution Server in a J2SE environment. For accessing the rules using the Rule Session API, it can:

- ▶ Partially reduce capability JRules Execution Server compared to J2EE deployment
- ▶ Allow Message Broker to integrate directly with deployed rules



A

Additional material

This book refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

<ftp://www.redbooks.ibm.com/redbooks/SG247826>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247826.

Using the Web material

The additional Web material that accompanies this book includes the following files:

<i>File name</i>	<i>Description</i>
V61_flows.zip	Use in section 9.3.2. It contains the message flows V61_publish.msgflow, V61_publish.esql, V61_subscribe.msgflow, and V61_subscribe.esql.
V7_flows.zip	Use in section 9.5.1. It contains the message flows V7_publish.msgflow, V7_publish.esql, V7_subscribe.msgflow, and V7_subscribe.esql.
V7_flows_MQ.zip	Use in section 9.5.2. It contains the message flows V7_publish_MQ.msgflow, and V7_publish_MQ.esql.
BedMonitoring Application.zip	Use in section 7.3.1. It contains the message flows BedMonitoring.msgflow, Feeds.msgflow, BuildXMLMessage.esql, and bedmonitoringapp.bar.
TelnetToWMQ.zip	Use in section 7.3.2. It contains the message flows Telnet_to_Queue.msgflow, Telnet_to_Queue.esql, and telnettoqueue.bar.
Bedapp.jar	Use in section 7.3.2. It contains the Server.java and Server.class files.
SetupTransferRequest _SCARequest.zip	Use in section 5.4.4. It contains the JAVA code used in the scenario.
SetupTransferReply _SCAReply.zip	Use in section 5.4.4. It contains the JAVA code used in the scenario.

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 536. Note that some of the documents referenced here may be available in softcopy only.

- ▶ Using IBM WebSphere Message Broker as an ESB with WebSphere Process Server, SG24-7527
- ▶ Integrating WebSphere Service Registry and Repository with WebSphere MQ and WebSphere Message Broker REDP-4558

Online resources

These Web sites are also relevant as further information sources:

- ▶ WebSphere Message Broker Version 7.0 in the WebSphere Message Broker Information Center
http://publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ab20551_.htm
- ▶ *Enterprise Connectivity Patterns: Implementing integration solutions with IBM's Enterprise Service bus products*
<http://www.ibm.com/developerworks/webservices/library/ws-enterpriseconnectivitypatterns/index.html>
- ▶ WebSphere Adapters
<http://www-01.ibm.com/software/integration/wbiadapters/>
- ▶ WebSphere MQ SupportPacs page
<http://www-1.ibm.com/support/docview.wss?rs=849&uid=swg27007205>
- ▶ Logical tree structure, Information Center
http://www publib.boulder.ibm.com/infocenter/wmbhelp/v7r0m0/topic/com.ibm.etools.mft.doc/ac00490_.htm

- ▶ WebSphere Message Broker Product Support Web site
<http://www-1.ibm.com/support/docview.wss?rs=849&uid=swg27007205>
- ▶ Eclipse
<http://www.eclipse.org/pdt/downloads/>
- ▶ Project Zero, open source tools for PHP development
<http://www.projectzero.org>
- ▶ XDEBUG EXTENSION FOR PHP
<http://xdebug.org>
- ▶ WebSphere MQ Information Center
http://www publib boulder ibm com infocenter wmqv7/v7r0/topic/com.ibm.mq.amqnar.doc/ps22040_.html
- ▶ WebSphere Business Monitor
<http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp>

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM



Redbooks

Connecting Your Business Using WebSphere Message Broker V7 as an ESB

(1.0" spine)
0.875" <-> 1.498"
460 <-> 788 pages



Connecting Your Business Using WebSphere Message Broker V7 as an ESB



SOA interoperability with WebSphere BPM products

This IBM Redbooks publication points out the key features that make WebSphere Message Broker a powerful choice as an enterprise service bus (ESB) solution in a service-oriented architecture (SOA) environment. In this book, we illustrate the interoperability between the WebSphere Message Broker and the applications in the SOA environment.

Functional scenarios around PubSub and file processing

We use realistic examples to show the ESB capabilities of WebSphere Message Broker. We also show how to integrate WebSphere Message Broker with a variety of enterprise applications, which include WebSphere Process Server and ESB systems including SAP and Siebel, WebSphere Business Monitor, and WebSphere Service Registry and Repository.

TCP/IP-based and ERP system connectivity

We wrote this book for architects who are planning an SOA solution and application designers who are implementing an SOA solution with WebSphere Process Server and WebSphere Message Broker.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks