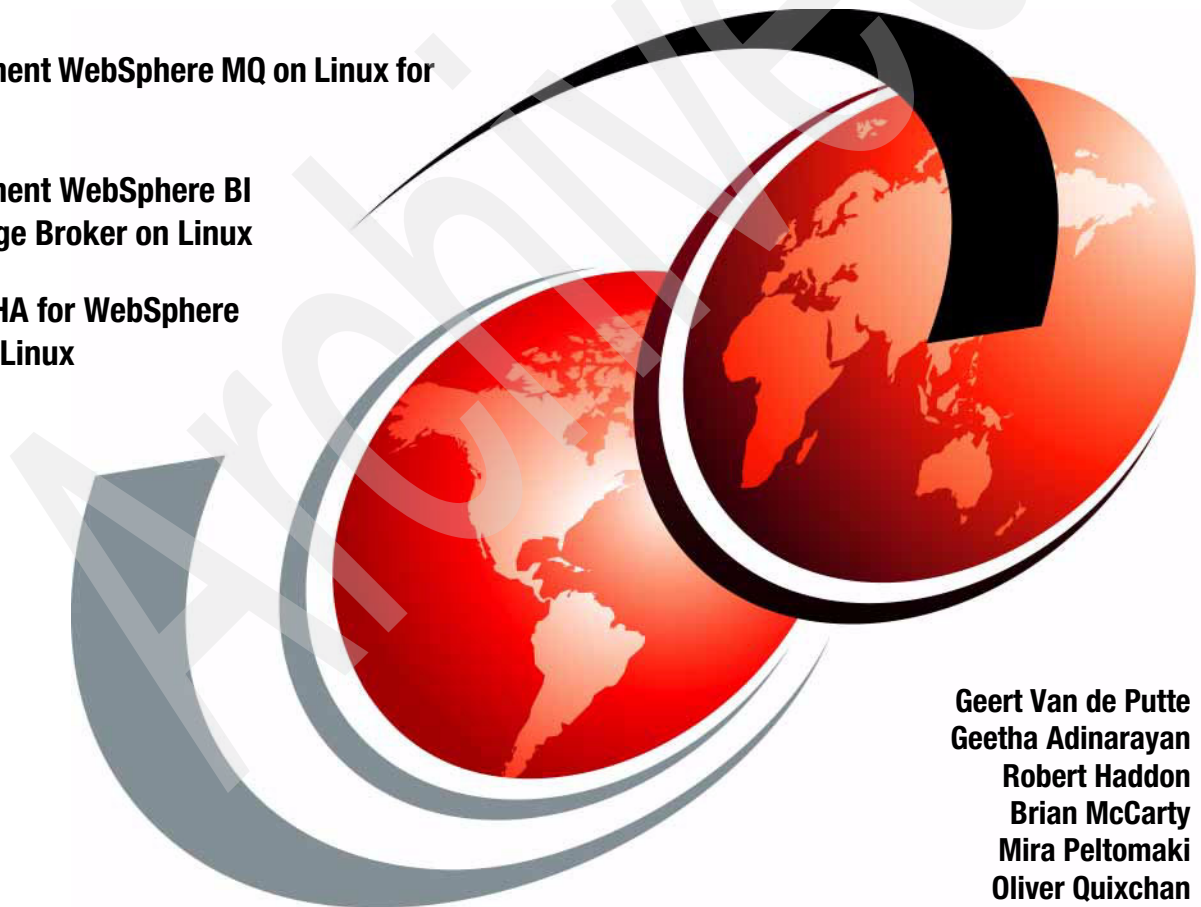


# Messaging Solutions in a Linux Environment

Implement WebSphere MQ on Linux for  
Intel

Implement WebSphere BI  
Message Broker on Linux

Learn HA for WebSphere  
MQ on Linux



Geert Van de Putte  
Geetha Adinarayan  
Robert Haddon  
Brian McCarty  
Mira Peltomaki  
Oliver Quixchan





International Technical Support Organization

## **Messaging Solutions in a Linux Environment**

July 2005

Archived

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

### **First Edition (July 2005)**

This edition applies to IBM WebSphere MQ for Linux for Intel, Version 5.3 and to IBM WebSphere Business Integration Message Broker Version 5.0 (product number 5724-E26).

**© Copyright International Business Machines Corporation 2005. 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

<b>Notices</b> .....	ix
Trademarks .....	x
<b>Preface</b> .....	xi
The team that wrote this redbook .....	xi
Become a published author .....	xiv
Comments welcome .....	xiv
<b>Chapter 1. Introduction to Linux and messaging</b> .....	1
1.1 Introduction to the world of Linux .....	2
1.1.1 Linux .....	2
1.1.2 Linux kernel .....	3
1.1.3 Linux operating system .....	7
1.1.4 Linux distributions .....	9
1.2 Concepts of Message Oriented Middleware .....	12
1.2.1 Point-to-point messaging .....	13
1.2.2 Publish/subscribe .....	13
1.2.3 A conceptual paradigm view .....	14
1.3 Introducing WebSphere MQ .....	14
1.3.1 WebSphere MQ support for Linux .....	15
1.3.2 WebSphere MQ objects and security .....	15
1.3.3 WebSphere MQ administration and management .....	16
1.3.4 WebSphere MQ intercommunication and remote queuing .....	17
1.3.5 WebSphere MQ transport types .....	17
1.3.6 WebSphere MQ application programming interfaces .....	18
1.4 Message broker concepts .....	21
1.5 Introducing WebSphere BI Message Broker family .....	22
1.5.1 Architecture overview .....	23
1.5.2 Runtime environment .....	25
1.6 Introducing WebSphere Application Server .....	29
<b>Chapter 2. Linux systems and advanced technologies</b> .....	33
2.1 Scalability versus high availability .....	34
2.1.1 Scalability .....	34
2.1.2 High availability .....	35
2.2 Linux technologies for scalability .....	37
2.2.1 Storage area network .....	37
2.2.2 IBM hardware offerings for Linux scalability .....	38
2.2.3 IBM software offerings for Linux scalability .....	39

2.2.4	Open source offerings for Linux scalability . . . . .	41
2.3	Linux technologies for high availability . . . . .	42
2.3.1	Journalized file systems. . . . .	42
2.3.2	IBM software offerings for Linux high availability . . . . .	47
2.3.3	Open source offerings for Linux high availability. . . . .	47
2.4	Creating a highly available and scalable solution . . . . .	48
<b>Chapter 3. Implementing HA queue managers: Part 1 . . . . .</b>		<b>51</b>
3.1	Scenario one overview . . . . .	52
3.1.1	Using WebSphere MQ in bindings mode . . . . .	52
3.1.2	Using WebSphere MQ in client mode. . . . .	52
3.2	Using Linux-HA . . . . .	53
3.2.1	Features of Linux-HA . . . . .	53
3.2.2	Planning an implementation . . . . .	54
3.2.3	Perform an implementation. . . . .	55
3.2.4	Heartbeat configuration. . . . .	56
3.2.5	Heartbeat authentication . . . . .	56
3.2.6	High availability resource configuration. . . . .	57
3.2.7	Network mirror configuration. . . . .	58
3.2.8	Validating heartbeat failover and failback . . . . .	64
3.2.9	HA configuration summary . . . . .	68
3.3	Installing and configuring WebSphere MQ . . . . .	68
3.3.1	Preinstallation steps . . . . .	69
3.3.2	Installation steps . . . . .	70
3.3.3	Postinstallation steps and install verification . . . . .	70
3.4	Message generation application . . . . .	71
3.4.1	Design of the message generation application . . . . .	71
3.4.2	Developing the message generation application . . . . .	71
3.4.3	Compiling and running the message generation application . . . . .	72
3.5	Message retrieval application . . . . .	73
3.5.1	Design of the message retrieval application . . . . .	74
3.5.2	Developing the message retrieval application. . . . .	74
3.6	WebSphere MQ HA configuration and scripts. . . . .	79
3.6.1	Normal scenario . . . . .	79
3.6.2	Configuring wmq1 and wmq2 for WebSphere MQ HA . . . . .	80
3.6.3	Failover . . . . .	82
3.6.4	Failback. . . . .	84
3.7	Persistent messages on WebSphere MQ queues . . . . .	85
3.8	Bindings versus client tests . . . . .	86
3.9	Running the Bindings mode test . . . . .	86
3.9.1	Queue Manager configuration. . . . .	86
3.9.2	Configuration verification. . . . .	91
3.9.3	WebSphere Application Server configuration . . . . .	93

3.9.4 Queue Manager in action . . . . .	99
3.9.5 Summary of results . . . . .	111
3.10 Running the Client mode test . . . . .	112
3.10.1 Queue manager configuration. . . . .	112
3.10.2 Configuration of WebSphere Application Server. . . . .	114
3.10.3 WebSphere Application Server in action. . . . .	115
3.10.4 Summary of results . . . . .	118
3.11 Summary . . . . .	118
<b>Chapter 4. Implementing HA queue managers: Part 2.</b> . . . . .	<b>121</b>
4.1 Scenario two overview . . . . .	122
4.2 Implementing Linux-HA on SUSE . . . . .	122
4.2.1 Planning an implementation . . . . .	122
4.2.2 Installing Linux High-Availability . . . . .	124
4.2.3 Configuring Linux-HA . . . . .	125
4.2.4 Configuring and testing the shared SCSI drives . . . . .	128
4.2.5 Validating heartbeat failover and failback . . . . .	131
4.2.6 HA configuration summary . . . . .	134
4.3 Installing and configuring WebSphere MQ . . . . .	134
4.4 Message generation application . . . . .	134
4.4.1 Compiling and running the message generation application . . . . .	134
4.5 Messaging retrieval application . . . . .	135
4.6 WebSphere MQ HA scripts and configuration. . . . .	135
4.6.1 Normal scenario . . . . .	135
4.6.2 Configuring brk1 and brk2 for WebSphere MQ HA. . . . .	136
4.7 Running the shared-disk test. . . . .	136
4.7.1 Queue manager configuration. . . . .	137
4.7.2 Configuration verification. . . . .	140
4.7.3 WebSphere Application Server configuration . . . . .	141
4.7.4 Application in action . . . . .	142
4.8 Summary . . . . .	149
<b>Chapter 5. Implementing HA queue managers: Part 3.</b> . . . . .	<b>151</b>
5.1 Scenario three overview . . . . .	152
5.2 Planning an implementation . . . . .	152
5.2.1 Hardware setup. . . . .	153
5.2.2 Cluster components . . . . .	153
5.2.3 GPFS file systems. . . . .	156
5.3 Performing an implementation . . . . .	158
5.3.1 Time synchronization . . . . .	158
5.3.2 Secure communications . . . . .	160
5.3.3 Support programs for the FASTT device . . . . .	160
5.3.4 RSCT and GPFS. . . . .	164

5.3.5	Configuring WebSphere MQ	174
5.3.6	Message generating application	175
5.3.7	Compiling and running the message generation application	175
5.3.8	Messaging retrieval application	175
5.3.9	Adding Linux high availability	175
5.3.10	HA configuration summary	176
5.4	Running the shared-disk test	177
<b>Chapter 6. Using WebSphere MQ clustering</b>		
6.1	Scenario four overview	180
6.1.1	Using WebSphere MQ clustering for high-availability	180
6.2	Introduction to WebSphere MQ Cluster	182
6.2.1	WebSphere MQ clusters benefits	182
6.2.2	WebSphere MQ cluster terminology	185
6.3	Installing and configuring WebSphere MQ	186
6.4	Message generation application	186
6.5	Messaging retrieval application	187
6.6	WebSphere MQ cluster setup	187
6.6.1	Defining the full repository queue managers	187
6.6.2	Defining the cluster partial repository queue managers	188
6.6.3	Configuration verification	190
6.7	Configuration of WebSphere Application Server	194
6.8	Application in action	194
6.9	Summary	198
<b>Chapter 7. Implementing highly available brokers</b>		
7.1	Scenario overview	200
7.1.1	Using brokers with WebSphere MQ clustering for high availability	200
7.1.2	Using hardware failover for high-availability in brokers	201
7.2	Introduction to WebSphere BI Message Broker	202
7.2.1	WebSphere BI Message Broker architecture	203
7.2.2	The broker	204
7.2.3	Introducing message flows	205
7.2.4	Introduction to message modeling and message sets	214
7.2.5	Message Brokers Toolkit for WebSphere Studio	217
7.2.6	WebSphere BI Message Broker configuration manager	221
7.2.7	User name server	221
7.3	Installing and configuring the broker	222
7.3.1	Installing and configuring WebSphere MQ	222
7.3.2	Installing and configuring DB2	222
7.3.3	Installing and configuring WebSphere BI Message Broker	222
7.4	Implementation steps	228



7.5	Creating databases and tables . . . . .	229
7.5.1	Creating databases on the WebSphere Application Server . . . . .	229
7.5.2	Database setup on Windows win6336 . . . . .	230
7.5.3	DB2 client setup on hosts brk1 and brk2 . . . . .	231
7.6	Design and development of the message set . . . . .	231
7.6.1	Designing the message set. . . . .	231
7.6.2	Developing the message set. . . . .	232
7.6.3	Creating the message definition file . . . . .	235
7.6.4	Creating the logical structure of the message. . . . .	239
7.7	Design and development of message flow . . . . .	246
7.7.1	Message flow design. . . . .	246
7.7.2	Message flow development. . . . .	247
7.8	Configuring the broker domain . . . . .	270
7.8.1	Queue manager configuration. . . . .	270
7.8.2	Creation and configuration of configuration manager. . . . .	271
7.8.3	Broker creation . . . . .	272
7.8.4	Connecting brokers to the configuration manager . . . . .	272
7.9	Deploying the message flow and message set. . . . .	279
7.9.1	Creating a message broker archive . . . . .	280
7.9.2	Deploying a bar file . . . . .	284
7.10	WebSphere Application Server configuration . . . . .	287
7.11	Message generation application . . . . .	294
7.12	Message retrieval application . . . . .	294
7.12.1	The servlet MonitorStatusServlet.java . . . . .	295
7.12.2	The class. . . . .	295
7.13	Application in action . . . . .	298
7.13.1	Normal scenario . . . . .	298
7.13.2	Failover scenario. . . . .	300
7.13.3	Failback scenario . . . . .	301
7.14	Summary. . . . .	303
	<b>Chapter 8. Managing a distributed Linux messaging infrastructure . . . . .</b>	<b>305</b>
8.1	Planning monitoring. . . . .	306
8.2	Implementing WebSphere MQ event monitoring . . . . .	306
8.2.1	Queue manager events. . . . .	307
8.2.2	Performance events . . . . .	308
8.2.3	Channel events . . . . .	311
8.2.4	Monitoring program. . . . .	311
8.3	Implementing monitoring tool Monit . . . . .	312
8.3.1	Installing Monit . . . . .	312
8.3.2	Configuring Monit . . . . .	313
	<b>Appendix A. Hardware and software configuration . . . . .</b>	<b>317</b>

Hardware and software used for first scenario . . . . .	318
Hardware and software used for the second scenario . . . . .	318
Hardware and software used for the third scenario . . . . .	319
<b>Appendix B. Using external SCSI storage enclosure with Linux . . . . .</b>	<b>321</b>
Configuring ServerRAID controllers for clustering . . . . .	322
<b>Appendix C. Configuring a FASTT200 disk system for Linux . . . . .</b>	<b>327</b>
Configuring SANbox2 to use static IP address . . . . .	328
Configuring the FASTT200 disk enclosure to use a static IP-address . . . . .	328
Installing the Fiber Host Adapter FC2/133 . . . . .	329
Updating the host adapter BIOS . . . . .	329
Install the host adapter drivers . . . . .	330
Install the FASTT_MSJ . . . . .	332
<b>Appendix D. Additional material . . . . .</b>	<b>333</b>
Locating the Web material . . . . .	333
Using the Web material . . . . .	333
System requirements for downloading the Web material . . . . .	334
How to use the Web material . . . . .	334
<b>Abbreviations and acronyms . . . . .</b>	<b>335</b>
<b>Related publications . . . . .</b>	<b>339</b>
IBM Redbooks . . . . .	339
Other publications . . . . .	339
Online resources . . . . .	339
How to get IBM Redbooks . . . . .	340
Help from IBM . . . . .	340
<b>Index . . . . .</b>	<b>341</b>

# Notices

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

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

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

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

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

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

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

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

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

## COPYRIGHT LICENSE:

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

# Trademarks

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

AIX 5L™  
AIX®  
BladeCenter™  
BookMaster®  
DB2®  
e(logo)server®  
ESCON®  
@server®  
Eserver®  
@server®  
eServer™  
Everyplace®  
FlashCopy®

HACMP™  
ibm.com®  
IBM®  
iSeries™  
MQSeries®  
Netfinity®  
Perform™  
Power PC®  
POWER™  
pSeries®  
Redbooks (logo)™  
Redbooks (logo) ™  
Redbooks™

SAA®  
ServeRAID™  
ServicePac®  
SupportPac™  
Tivoli®  
TotalStorage®  
WebSphere®  
XDE™  
xSeries®  
z/OS®  
zSeries®

The following terms are trademarks of other companies:

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

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

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

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

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

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

# Preface

Recently, the adoption of Linux as a platform for business-critical applications has increased significantly. This evolution has also introduced requirements and technologies that were previously not seen in a Linux environment. Messaging and integration products, such as WebSphere MQ and WebSphere BI Message Broker, can now be used on Linux to solve integration problems that include Linux platforms.

This IBM Redbook starts with an introduction of messaging and integration technologies to Linux users. It also introduces the world of Linux to people that are familiar with messaging and integration technologies on other platforms.

The following chapters discuss several implementations of WebSphere MQ that use Linux technologies to solve high-availability problems. These scenarios rely on software and hardware products such as disk-mirroring, shared-disks using GPFS, and WebSphere MQ clustering. Within this environment, we also demonstrate the use of WebSphere BI Message Broker, C++ applications using WebSphere MQ and Java™ applications using JMS within WebSphere Application Server on Linux.

This book ends with a discussion of a few monitoring techniques that can be implemented in a Linux environment, so that a messaging solution can be monitored.

## The team that wrote this redbook

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



*Left to right: Oliver, Brian, Geert, Rob, Geetha and Mira*

**Geert Van de Putte** is an IT Specialist at the ITSO, Raleigh Center. He is a subject matter expert for messaging and business integration and has eight years of experience in the design and implementation of WebSphere® MQ-based application integration solutions. He has published several IBM Redbooks™ about messaging and business integration solutions. Geert has also taught several classes about messaging, business integration and workflow. Before joining the ITSO, Geert worked at IBM Global Services, Belgium, where he designed and implemented EAI solutions for customers in many industries. Geert holds a Master of Information Technology degree from the University of Ghent in Belgium.

**Geetha Adinarayan** is a software Engineer at IBM Software Labs, Bangalore, India. She has five years of experience in IBM messaging middleware products. She holds a degree in Information Systems from BITS, Pilani, India. Geetha is a Certified Software Test Engineer and IBM Certified System Administrator for WebSphere Business Integration Message Broker V5. Her interests are architecting messaging solutions and test automation.

**Robert Haddon** is an IT Specialist in United States. He has nearly twenty years of experience in IT, the last seven spent with IBM. His areas of expertise include Linux and IP networking.

**Brian McCarty** is a Senior Programmer from Texas with USAA. He has seven years of experience with Enterprise Application Integration and Messaging Solutions in banking, investment and manufacturing. His interests are developing highly available applications for WebSphere Application Server, WebSphere MQ and WebSphere BI Message . Previously, Brian contributed to the IBM Redbook *WebSphere MQ Integrator Deployment and Migration*, SG24-6509.

**Mira Peltomaki** is an IBM IT Architect and leads an Advanced Media Research lab within the Interdisciplinary Institute for BroadBand Technology (IBBT) in Ghent, Belgium. She has 14 years of experience in Linux, clustering, parallel file systems, IBM hardware and middleware products. She holds a masters degree in mathematics. She has published several white papers about the storage performance and scalability for digital media platforms and contributed to the IBM Redbook *GPFS in a Digital Media Environment*.

**Oliver Quixchan** is an e-business consultant for Smart Business Solutions in Guatemala. He holds a Bachelor of Science in Computer Science from Universidad Francisco Marroquin, Guatemala. He has six years of experience in application development, Web-oriented solutions using J2EE and WebSphere software platform, banking software development, and business integration. His areas of expertise include WebSphere Application Server, integration to older systems, XML, J2EE, WebSphere MQ. Previously he contributed to the Redbook *The XML Files: Development of XML/XSL Applications Using WebSphere Studio Version 5*.

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

Wouter Cloetens  
Mind NV

Tonko De Rooy  
IBM Adanced Technical Support, Americas

Vailamur N Kanakagiri  
IBM Software Lab, India

Ben Smith  
Program Director, eServer™ Linux Clusters Group

Sharma Vemuri  
Senior Software Engineer, eServer Linux Clusters Group

## Become a published author

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

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

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

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

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

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

[ibm.com/redbooks](http://ibm.com/redbooks)

- ▶ Send your comments in an Internet note to:

[redbook@us.ibm.com](mailto:redbook@us.ibm.com)

- ▶ Mail your comments to:

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





# Introduction to Linux and messaging

This chapter provides an overview of Linux and messaging solutions and describes some typical implementations. In addition it also describes in some detail the specific solutions used in the remainder of this book. We describe each component and then show what role it plays in an overall solution.

**Note:** This chapter introduces a Linux professional to messaging technologies. At the same time, it is also introduces messaging and WebSphere MQ professionals to Linux technologies.

# 1.1 Introduction to the world of Linux

In 1991 Linus Torvalds, back then a university student of Computer Science at the University of Helsinki, sent a historic post to the MINIX<sup>1</sup> news group announcing that he was working on a new free operating system and was interested in getting feedback of features that should be included into Linux. A few weeks later Linux version 0.01 was released on the Internet and its development became a shared effort as developers downloaded code, tested it, tweaked it and sent it back to Linus who then integrated them into his project.

New versions came rapidly and the amount of followers grew as the code went worldwide through FTP sites. Soon Linux became popular among students and developers and the project expanded. Linux was licensed under the GNU General Public License (GPL) ensuring that the source code had to be included and it would be free for everyone to copy, study and change. The GPL can be read at:

<http://www.gnu.org/copyleft/gpl.html>

Although Linux itself was free, commercial vendors moved in soon in order to make money by gathering and compiling software and hence creating a package easy to install and distribute more like the other more familiar commercial operating systems.

Today, Linux has been ported to many other platforms, from hand-held devices to mainframes. Added support for clustering have enabled highly available and scalable solutions using Linux.

## 1.1.1 Linux

Distributions, operating systems, kernels; Linux terminology can be confusing. For example when people talk about Linux they mean any of the following three:

<b>Kernel</b>	The base layer that controls hardware, manages files, creates processes and so forth.
<b>Operating System</b>	Kernel combined with sets of utilities, libraries and applications to form a complete operating system.
<b>Distribution</b>	Operating system bundled with configuration programs and package management.

---

<sup>1</sup> Small UNIX® clone designed for IBM PCs and compatibles. It is ideal for educational purposes due to its small size, microkernel-base design and full documentation. Although MINIX comes with the source code, it is copyrighted software.

Similarly, when we talk about versions of Linux we either mean the version of the Linux kernel or a version of a specific Linux distribution. It is more accurate to talk about Linux kernel version 2.4 and Red Hat Enterprise Linux version 3 than to refer to Linux 2.4 or Linux 3.

## 1.1.2 Linux kernel

The Linux kernel has a monolithic design, which means that most of the operating system is a single file that runs in *kernel mode*, other examples of such operating systems are DOS, UNIX and VMS. The binary contains:

- ▶ Low-level hardware support for devices such as the processor, interrupt controller, real-time clock, input devices, CD-ROM drives and so on
- ▶ Memory management
- ▶ Task management and task scheduling
- ▶ Networking protocols such as TCP/IP and Netbios
- ▶ Bus support for PCI, ISA, MCA, USB, FireWire
- ▶ File system support.
- ▶ System call interface, a layer between the kernel and the programs running on top of it, and its implementation
- ▶ Device driver management
- ▶ Device drivers themselves, which are interacted with by programs through device specific files, such as SCSI and IDE disk drivers
- ▶ Power management

In order to better understand Linux kernel and its constant development, let us become familiar with some definitions:

<b>Linux kernel mailing list</b>	A virtual space where Linux developers from around the world discuss the kernel features, exchange experience, ask questions and help each other
<b>Bitkeeper repository</b>	A source control system used for the official, and many of the unofficial, development trees
<b>Patch</b>	A new kernel feature or modification of an existing one, sometimes a temporary fix for a bug until the problem can be solved more thoroughly with an architectural change
<b>Patch set</b>	A collection of patches against an official kernel release with a certain purpose, such as

additional experimental drivers, low latency for enhanced desktop interaction, and so on.

### **Kernel release**

A new official kernel level that has been tested to be stable. In every new version or release there are improvements or changes to already implemented architectures and support for new emerging technologies and optionally for old obsolete technologies

### **Custom kernel**

Some of the major distributions and many Linux users create their own kernels adding and deleting features in order to create a kernel best fit for the additional programs they ship.

### **Changelog**

A list of changes and new features compared to the previous official release.

As an example, we can examine the latest stable Linux kernel, 2.6.5, and its development process:

1. Developers around the world are fixing bugs and writing new features or polishing up ones that had not previously been accepted to be included in the official kernel.
2. All these *patches* and *patch sets* are submitted to Linus Torvalds himself who works on the *Linus development tree* on *Bitkeeper* together with a few other trusted associates.
3. Linus himself made the decision which *patches* are going to be included in the new stable kernel.
4. At some point in time, Linus decides to cut off the patch submission process, effectively freezing the functionality, and releases a *release candidate* version. There may be several of such versions.
5. When the new functionality is tested to a satisfactory degree, the new *kernel* is officially released; announcement is made and source tree together with *changelog* is made available at Linux Kernel Archives and its mirrors all around the world. Many developers who like to follow the *bleeding edge* immediately download the kernel and compile it to take advantage of the new features it offers. On the Web the Linux kernel archives is found at:  
<http://www.kernel.org/>
6. Major changes and additions according to the *changelog* in this kernel include:
  - Updates to several file systems including reiserFS, nfs, ext2 and ext3.
  - Addition of Netpoll interface.

- A change to the DMA interface.
- Improvements to hotplug CPU support.

For more information about the kernel development process, please visit:

<http://kerneltrap.org/>

## **New threading model introduced in kernel release 2.6**

Traditional Unix programming dogma dictates that single-threaded programming, with the use of an asynchronous programming model to handle multiple events consecutively, is more resource-efficient than multithreaded programming. While this is true on a single-CPU system, it is not for *SMP systems*.

There may also be other architectural or technological reasons why you would prefer or need a multithreaded model, such as an application source code base ported from another OS, the use of the Java programming language which has a strong multithread vision, or simply programmer experience with one model rather than the other. UNIX is also an operating system in which new processes are spawned at an enormous rate. Hence the operating system architecture of UNIX-like operating systems, including Linux, tends to be oriented towards extremely efficient process creation.

As a result of these two factors, support for multithreading on Linux was originally implemented as so-called *Light-Weight Processes* (LWP). This model, called *LinuxThreads*, starts a new process for every thread, but maintains a shared address space, open file descriptor table and so on between all the processes. Communication between threads and signal processing was managed using an extra management thread. The LinuxThreads model fails to provide a foundation for a fully compliant implementation of the *POSIX threads* standard. For one, every thread has a different process ID. Also, signals are not handled correctly. An in-depth discussion about the limitations of the LinuxThreads model and the design requirements for the new threading model can be found at:

[http://www.redhat.com/partners/pdf/POSIX\\_Linux\\_Threading.pdf](http://www.redhat.com/partners/pdf/POSIX_Linux_Threading.pdf)

After much debate and benchmarking of alternative implementations, it was decided that the Linux kernel needed proper support for multithreading. As of kernel 2.6, this is now the case. These improvements are exploited by the *Native POSIX Thread Library* (NPTL), which is fully POSIX-compliant and reduces overhead.

However, LinuxThreads also benefits from the kernel improvements. NPTL no longer requires an extra management thread, and while there is still a one-on-one mapping between user-space threads and kernel threads, there is now a single process ID for a multithreaded program.

If we inspect this in practice, we find the following output for the ps program for a process that runs in eight threads. See Example 1-1 for threads on kernel 2.4 (LinuxThreads):

*Example 1-1 Kernel 2.4 LinuxThreads*

---

```
mqm 6033 0.0 0.2 22284 524 pts/242 S 16:48 0:00 ./threadexample
mqm 6034 0.0 0.2 22284 524 pts/242 S 16:48 0:00 ./threadexample
mqm 6035 0.0 0.2 22284 524 pts/242 S 16:48 0:00 ./threadexample
mqm 6036 0.0 0.2 22284 524 pts/242 R 16:48 0:00 ./threadexample
```

---

See Example 1-2 for the same output on kernel 2.6 (NPTL):

*Example 1-2 Kernel 2.6 NPTL threads*

---

```
mqm 5956 0.0 0.2 83484 460 pts/242 S 16:45 0:00 ./threadexample

brkl:~$ ls -l /proc/5956/
task total 0
dr-xr-xr-x 3 mqm users 0 Apr 26 16:44 5956
dr-xr-xr-x 3 mqm users 0 Apr 26 16:44 5957
dr-xr-xr-x 3 mqm users 0 Apr 26 16:44 5958
brkl:~$ cat /proc/5956/status
...
Threads: 11
```

---

Because both provide implementation of the standard POSIX threads interface, NPTL and LinuxThreads are mostly binary compatible, and programs do not need to be recompiled or relinked to take advantage of NPTL.

However, some programs depend on LinuxThreads behavior, and these need to be ported. Alternatively, it is possible to force the pthread library to revert to the LinuxThreads implementation by setting the environment variable `LD_ASSUME_KERNEL` to an older version of the kernel. See Example 1-3

*Example 1-3 Using LinuxThreads implementation with 2.6 kernel*

---

```
brkl:~$ LD_ASSUME_KERNEL=2.4.1 ./threadexample
```

---

This is specifically required for WebSphere MQ. See 3.3, “Installing and configuring WebSphere MQ” on page 68.

Both kernel 2.6 and a newer version of the pthread library, part of the GNU libc library, are required for NPTL to work.

**Tip:** To inspect which threading implementation is supported, use the following command:

```
getconf GNU_LIBPTHREAD_VERSION
```

On 2.4, this will return something similar to:

```
brk1:~$ linuxthreads-0.10
```

While on 2.6, output such as this is expected:

```
wmq1:~$ NPTL 0.60
```

### 1.1.3 Linux operating system

In addition to the kernel, you need a set of applications, libraries and utilities. If you want to build your own custom operating system from available packages, it should contain at least the following:

- ▶ Boot loader
- ▶ C library
- ▶ Kernel-specific utilities such as *modutils* for kernel module management, *net-tools* for networking configuration, the system log daemon, the kernel log message daemon and so on
- ▶ The *init* program, first program to run and bootstrap the operating system's boot script
- ▶ The boot script starting the boot time configuration scripts and services
- ▶ Configuration files for the configuration scripts and services
- ▶ A *getty* program to provide a login prompt
- ▶ A *login* program to validate users against the provided password and set up their initial environment.
- ▶ A shell such as the GNU Bourne Again shell (bash) or the Korn shell (ksh)
- ▶ A large set of typical file manipulation, text and shell utilities
- ▶ A text editor such as vi or emacs
- ▶ A set of networking services such as inetd, the internet super server, an ssh server, and utilities such as ping, telnet, ftp, and ssh clients

The following packages are optional but provide features that make Linux more user-friendly:

## X-Windows

A windowing system which provides an easy-to-use graphical interface to the underlying Linux operating system, such as *Enlightenment* or *Sawfish*.

## Desktop environment

The leading desktop environments for Linux are *GNOME* (GNU Network Object Model Environment) and *KDE* (Kool Desktop Environment).

Information and downloads for GNOME are available here:

<http://www.gnome.org>

For KDE, visit:

<http://www.kde.org>

In addition, there are thousands of packages from which the user can pick and choose the ones that fit to his specific usage needs.

## Help for operating tasks

There are various ways to find help with any task at hand. The most popular ones include:

- ▶ *HOW-TO* is a selection of detailed documents on how to install, run, and operate Linux. Normally every package contains a single text file called HOW-TO.
- ▶ *Man pages* is installable online help for each and every Linux command explaining its usage and giving practical examples.

There are also several excellent tutorials available on the Web. These tutorials focus on various Linux-related tasks. Here is a list of few that are needed to complete the tasks described in the scenarios of this book:

- ▶ Kernel compiling tutorial

[http://www-106.ibm.com/developerworks/edu/1-dw-linuxkernel-i.html?S\\_TACT=104AHW22&S\\_CMP=EDU](http://www-106.ibm.com/developerworks/edu/1-dw-linuxkernel-i.html?S_TACT=104AHW22&S_CMP=EDU)

- ▶ Compiling and installing software from source:

<http://www-106.ibm.com/developerworks/edu/os-dw-linuxcompiling-i.html>

- ▶ Configuring TCP/IP under Linux:

[http://www-106.ibm.com/developerworks/edu/1-dw-linixtcpip-i.html?S\\_TACT=104AHW18&S\\_CMP=ZHP](http://www-106.ibm.com/developerworks/edu/1-dw-linixtcpip-i.html?S_TACT=104AHW18&S_CMP=ZHP)

- ▶ Online guides about any Linux related task on the Linux Documentation project:

<http://www.tldp.org>



- ▶ *Linux Handbook: Guide to IBM solutions and resources*, SG24-7000  
<http://www.redbooks.ibm.com/abstracts/sg247000.html?Open>

## 1.1.4 Linux distributions

Linux distributions all have one thing in common; they provide Linux in source code and binary form. The base code is freely redistributable under the terms of GNU General Public License (GPL) and any non-free and contributed software has to be clearly marked as such. Distributions divide into two main categories:

- ▶ *Non-Commercial distributions* tend to focus on creating technical excellence and follow a design philosophy or manifesto. These distributions do not offer any direct support other than mailing lists and bug reports. They are freely available on the Internet for download and most can be purchased as shrink-wrapped boxes with manuals.
- ▶ *Commercial distributions* offer training, certification, and support services in addition to their software, which is commercially distributed through official dealer channels or downloadable from the company's Web sites for payment. Most commercial distributions base themselves on non-commercial ones and are available for free download. However, any commercial software bundled with the product is only available on a separate CD for payment.

Some of the best known and widely-used distributions include Red Hat, SUSE, United Linux and Debian.

### Red Hat

Red Hat is among the biggest of the commercial distributions, supported by most computer manufacturers that certify their hardware to run a Red Hat distribution. Their distributions provide a graphical user interface (GUI) for installation and most administration tasks. It includes the RPM packaging system which allows you to cleanly install, deinstall, and upgrade operating system components, including the kernel and base operating system. Red Hat Enterprise Linux is available in four different distributions:

- ▶ Red Hat Enterprise Linux WS for desktop client systems
- ▶ Red Hat Enterprise Linux ES for small and midrange servers
- ▶ Red Hat Enterprise Linux AS for high-end and mission-critical systems
- ▶ Red Hat Professional Workstation for personal use

Red Hat also has an extensive software catalog including products for clustering, development and systems management. They also offer a distribution for non-Intel platforms such as AMD64, Power PC® and zSeries® servers. For more information, visit:

<http://www.redhat.com>

As of this writing, the latest Red Hat Enterprise Linux version is 3. This version builds on a hybrid kernel that combines the latest 2.6 kernel features with the more stable 2.4 kernel. For more information please visit:

<http://www.redhat.com/software/rhel/kernel26/>  
<http://www.redhat.com/software/rhel/21features/>

## **SUSE**

SUSE is a large commercial distribution, especially popular in Europe. It bases itself on technical excellence and makes an effort for the operating system to support all the latest hardware. Applications included offer Internet, multimedia and office tools that support all the common formats. It ships with YaST (Yet another Setup Tool), an advanced tool for graphical installation and system management. SUSE comes in three different flavors:

- ▶ SUSE Linux Personal
- ▶ SUSE Linux Professional
- ▶ SUSE Linux Enterprise

SUSE also offers Linux distributions for nonIntel platforms like AMD64 and mainframes. Additionally it is possible to purchase a program from SUSE that allows cross platforming between Windows®-based games and office products.

For more information about SUSE distributions, visit:

<http://www.suse.com>

As of this writing, the latest version for the SUSE Enterprise edition is version 8 and version 9 for the Professional and Personal editions. These latest versions run on kernel 2.6 and include the latest version 3.2 of the popular graphical desktop environment KDE.

## **United Linux project**

SUSE also participates in the United Linux project together with Connectiva, SCO and Turbolinux. This is not a single distribution. Instead it is a combination of the kernel and some applications which serve as a common core, to which each company then distributes their own add-on disks. For more information, visit:

<http://www.unitedlinux.com>

## **Debian GNU/Linux**

Debian is the most popular of the noncommercial distributions and has gained importance as scientific institutions, universities and some large international corporations have put their faith in it.

Debian is based on the Linux kernel, but most of its basic operating system tools come from the GNU project (<http://www.gnu.org>). It is produced by volunteer developers from all over the world and is popular among technical people due to its open design process which ensures that it reflects the needs of the user community. It closes the gap between the developers and users by letting the Debian package maintainers channel all the bug reports to the original software authors.

The distribution process is controlled in fashion similar to the kernel development itself. There are rigid sets of policies, with automated checking violation tools, that ensure issues such as dependencies and placement of the files on the file system, making the distribution stable and robust. Debian's package manager, DPKG, together with the APTtool makes upgrading packages or the whole distribution easy. The whole process can be automated, for example for security updates, thus reducing maintenance. Debian is available in these three distributions:

- ▶ Unstable  
This is the latest development effort.
- ▶ Testing  
This distribution phase is functions as the test to see if packages from the unstable phase work well together.
- ▶ Stable  
This distribution is released every two years or so after the testing distribution has been frozen, meaning that all the conflicts have been solved and known bugs have been fixed.

For more information about the Debian development process, visit:

<http://www.debian.org>

As of this writing, the latest Debian stable version is 3.0 which ships with both 2.2 and 2.4 kernels. They support eleven different architectures. New releases are made available simultaneously, unlike most commercial distributions.

### **Other popular Linux distributions**

There are literally hundreds of Linux distributions out there. The most popular ones include but are not limited to:

- ▶ Fedora
- ▶ Gentoo
- ▶ Knoppix
- ▶ Linux Pro
- ▶ Slackware
- ▶ LinuxGT

- ▶ KRUD
- ▶ Mandrake
- ▶ Libranet
- ▶ Turbolinux
- ▶ Xandros

ore information and a complete list of Linux distributions can be found at:

<http://www.distrowatch.org>

## 1.2 Concepts of Message Oriented Middleware

Message Oriented Middleware (MOM) is a set of services that provide a transport layer above the networking layers such as TCP/IP and SNA. A MOM system assures that packets of data or *messages* are delivered from one destination to another regardless of the state of the network or the recipient of the message. If the message can not be delivered to the receiver at the time the sender sends it, the message will wait in a queue until it can be delivered by *channels* that connect the MOM services together. The sender and receiver have no knowledge of each other's physical location or platform specifics. This is because the MOM system is responsible for providing an API that is reusable across computing platforms and dealing with issues such as networking protocols, message encoding and character translations. However, the MOM system alone does not provide any message transformation or dynamic routing.

MOM can be thought of as e-mail between applications through the use of queuing. *Queuing* is the process whereby messages are held until an application is ready to process them.

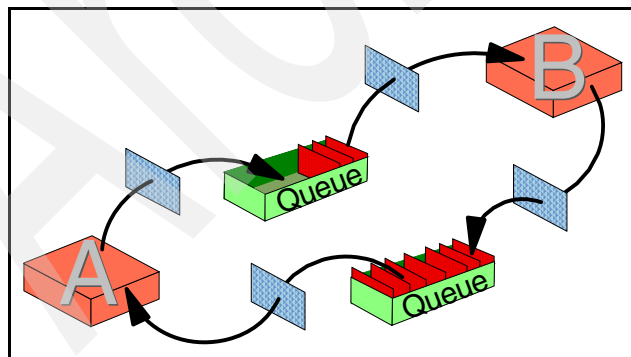


Figure 1-1 The usage of queues

MOM provides several important benefits:

- ▶ Communication between programs can be done without writing any communication code.
- ▶ The program can select the order in which messages are to be processed.
- ▶ Loads can be balanced across systems arranging for one or many programs to send or receive from a queue when volumes increase or decrease.
- ▶ The availability of data is increased by arranging for alternative systems to service a queue when a primary application is unavailable.

Further information about Message Oriented Middleware can be found in the IBM publication *MQSeries® Introduction to Messaging and Queuing*, GC33-0805-01.

### 1.2.1 Point-to-point messaging

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

### 1.2.2 Publish/subscribe

*Publish/subscribe* is the sending of data by one application to any number of receivers. This introduces a much more complex relationship between the sender and the receiver, where there is an abstraction between them so that they are unaware of each other. This abstraction layer is usually provided by a broker service which manages incoming data (publications) from sending applications (publishers) and matches them to interested receivers (subscribers). The broker then forwards the data to the registered subscribers. The correlation between publishers and subscribers is defined by using topics, which are contained in a hierarchical arbitrary namespace (that is, application-defined). Subscribers register interest in a topic or topics with the broker, and publishers publish messages using those topics. The names of topics usually relate to the information carried within them. The broker routes publications on particular topics to all those subscribers that have registered an interest in those topics.

The only restriction within the topic-space is that the structure is hierarchical. Beyond this, topic names within a level in the hierarchy are arbitrary. However, it is useful to give meaning to the hierarchy by naming the topics in a way that is indicative of the data that will be published on them. Subscribers can subscribe to a single topic, multiple topics, a single sub-tree of the topic hierarchy, or multiple sub-trees of the hierarchy. Topic-space sub-trees are specified by

inserting a wildcard into the topic identifier. A subscriber that uses a wildcard subscription will receive all publications that fall under that partition of the topic namespace.

A publication is published to a single topic. That is, wildcards are not permitted when publishing data, although a publisher can choose to publish messages to multiple different topics. A publisher can publish messages on a topic to which there are no subscribers. In this case, the broker will usually discard the messages although there are some cases that this may not happen. The point emphasized here is that the publishers and subscribers are completely de-coupled from each other.

### 1.2.3 A conceptual paradigm view

Another way of looking at the differences between point-to-point and publish/subscribe is to identify who has the responsibility of defining how the message flow is organized and driven. In the point-to-point paradigm, messages are pushed from the sending side, and the sending application decides which applications will receive the messages.

However, in the publish/subscribe paradigm, the senders and the receivers are decoupled, therefore the senders have no knowledge and no control of who receives their messages. In this situation, the receivers decide what information they receive, and therefore pull the messages from the source, subject to their authorization for receiving them.

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

## 1.3 Introducing WebSphere MQ

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

WebSphere MQ was developed to include the following features:

- ▶ Assured delivery  
Message persistence and transactional control allowed business critical applications to depend on messaging, where before they could not due to its inherent unreliability.
- ▶ A single API  
On all 35+ platforms supported by the WebSphere MQ brand, it has a common interface that hides all platform-specific complexities from the programmer.
- ▶ Asynchronous communication  
Applications sending and receiving messages do not have to be online at the same time to communicate with messaging. This solved the business need at the time, and although still relevant today, understanding of messaging and what it can provide as a technology has expanded enormously.

**Tip:** For a complete summary of all the WebSphere MQ documentation and references, see the IBM publication *WebSphere MQ Bibliography and Glossary* (SC34-6113-02).

### 1.3.1 WebSphere MQ support for Linux

At the time of this writing, WebSphere MQ is supported on any Linux for Intel® distribution running a version 2.4 kernel with glibc version 2.2. For installation, the Red Hat Package Manager is required. To use the WebSphere MQ for Java Classes and JMS support, a Java Runtime Environment 1.4 or higher is recommended.

**Important:** For current updates to the supported platforms and applications lists for WebSphere MQ, visit:

[http://www-306.ibm.com/software/integration/mqfamily/platforms/supported/wsm\\_q\\_for\\_linux\\_intel\\_5\\_3.html](http://www-306.ibm.com/software/integration/mqfamily/platforms/supported/wsm_q_for_linux_intel_5_3.html)

### 1.3.2 WebSphere MQ objects and security

The WebSphere MQ server component is called a *Queue Manager* and is responsible for managing WebSphere MQ *objects*: *queues*, *channels* and *processes*.

The Queue Manager can also provide transaction coordination, link-level security through *Secure Sockets Layer* (SSL) and localized or object security through the *Object Authority Manager* (OAM).

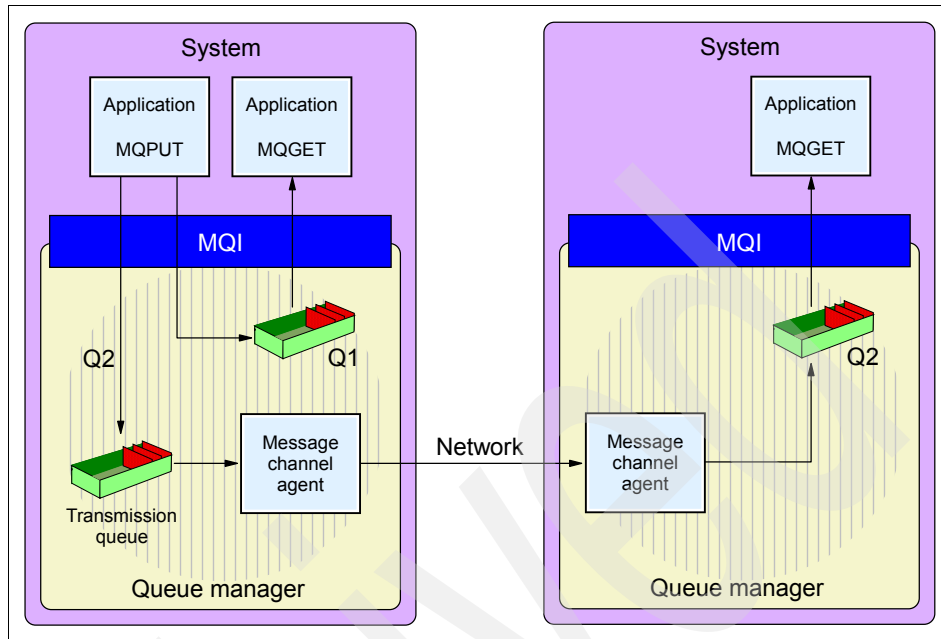


Figure 1-2 Queue Managers and objects

Further information about WebSphere MQ queue manager administration and security can be found in the IBM publications:

- ▶ *WebSphere MQ System Administration Guide* (SC34-6068-01)
- ▶ *WebSphere MQ Security* (SC34-6079-01)

### 1.3.3 WebSphere MQ administration and management

The queue manager also provides nearly platform independent management interface through *MQ Script Command* (MQSC) but can be programmatically administered through the *Programmable Command Format API* (PCF).

Further information about scripting and programming administrative tasks can be found in the IBM publications:

- ▶ *WebSphere MQ Script (MQSC) Command Reference*, SC34-6055-03
- ▶ *WebSphere MQ Programmable Command Formats and Administrative Interface*, SC34-6060-03



### 1.3.4 WebSphere MQ intercommunication and remote queuing

Messages can be sent from one queue manager to another through the use of *intercommunication*. To facilitate intercommunication, WebSphere MQ channels need to be created. In addition, the local queue manager must have an object definition called a *remote queue* that references the physical queue on a remote queue manager.

For example, a WebSphere MQ application program running on the same system with a queue manager sends messages to another queue manager on the same or a different system. This configuration requires channels to be defined for both queue managers, plus transmission queues and probably remote queues defined on the sending queue manager.

WebSphere MQ clusters reduce the system administration required with WebSphere MQ remote queuing.

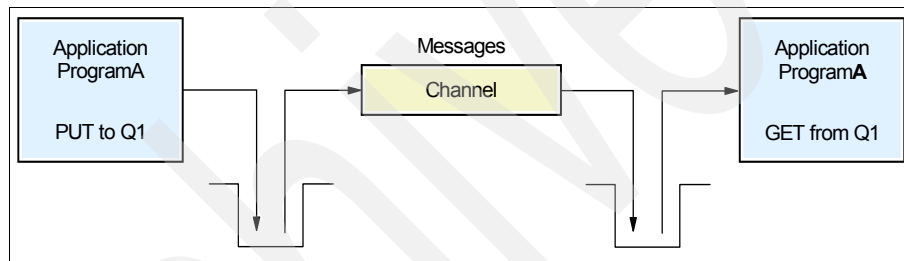


Figure 1-3 Channels and remote queue usage

**Restriction:** Channels should be created using TCP/IP as the network transport when using queue managers on Linux.

Further information about creating channels and communication between Queue Managers can be found in the IBM publication *WebSphere MQ Intercommunication* (SC34-6059-03).

### 1.3.5 WebSphere MQ transport types

WebSphere MQ supports *bindings* and *client* transport types.

- ▶ The bindings transport type requires that the application program runs on the same computer as the queue manager and uses a native interface to the queue manager. The advantages for using this is that it allows for asynchronous messaging and is faster.

- ▶ The client transport type allows an application program to run on a computer separate from the computer where the queue manager is running. The queue manager must have a server connection channel defined. This is done in a synchronous fashion. The application program on Linux normally uses a TCP/IP network connection to connect to the queue manager.

### 1.3.6 WebSphere MQ application programming interfaces

The three high-level APIs that are supported are the Message Queuing Interface (MQI), Java Messaging Service (JMS) and the Application Management Interface (AMI) that is a downloadable extension. The AMI is fully supported by IBM but is no longer actively developed.

#### Introduction to the MQI

One of the enormous benefits of WebSphere MQ is that communicating between applications can be done with a very small number of programming verbs across multiple languages. These verbs are also the foundation for object-oriented language support such as C++, WebSphere MQ for Java and JMS. There are 13 verbs in all but the most frequently used ones are pictured in Figure 1-4:

- ▶ *MQCONN* - for connecting a program to a queue manager
- ▶ *MQOPEN* - for opening a connection to an object such as a queue
- ▶ *MQPUT* - for placing a message in a queue
- ▶ *MQGET* - for retrieving a message from a queue
- ▶ *MQINQ* - for retrieving state information about an object such as a queue
- ▶ *MQSET* - for setting attributes on an object such as a queue
- ▶ *MQCLOSE* - for closing a connection to an object such as a queue
- ▶ *MQDISC* - for releasing a programs connection to a queue manager

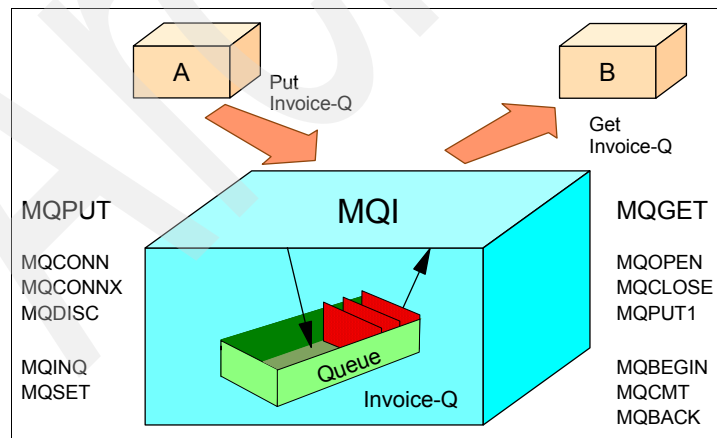


Figure 1-4 MQI verbs

For programmatic control of transactions, these additional verbs in Figure 1-5 are also supported:

- ▶ *MQBEGIN* - to notify a queue manager that a transaction is about to start
- ▶ *MQCOMMIT* - to commit the transaction with a queue manager
- ▶ *MQBACK* - to roll-back a transaction with a queue manager

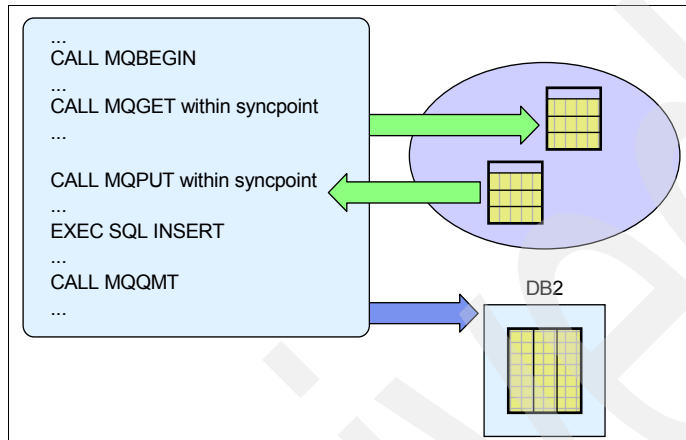


Figure 1-5 MQI Transactional Verbs

For further information about programming with WebSphere MQ API's can be found in IBM publications:

- ▶ *WebSphere MQ Application Programming Guide*, SC34-6064-03
- ▶ *WebSphere MQ Application Programming Reference*, SC34-6062-03

### Introduction to C++

The WebSphere MQ C++ classes encapsulate the MQI. There is a single header file (*imqi.hpp*) that provides all of the available classes. For C++ applications written using the C++ classes of WebSphere MQ to work on Linux, the following compiler and libraries are required:

- ▶ g++ version 2.95.2 or a version 3 compiler
- ▶ libstdc ++ Version 3.0

For further information about programming with WebSphere MQ C++ classes can be found in the IBM publication *WebSphere MQ Using C++* (SC34-6067-02). Up to date information about using C++ and the C++ classes of WebSphere MQ on Linux can be found at:

[http://www-306.ibm.com/software/integration/mqfamily/platforms/supported/wsmq\\_for\\_linux\\_intel\\_5\\_3.html](http://www-306.ibm.com/software/integration/mqfamily/platforms/supported/wsmq_for_linux_intel_5_3.html)

## Introduction to Java Message Service (JMS)

There are several different vendor implementations of MOM, each with its own unique benefits and drawbacks. In order to provide a common API across many different MOM vendors, the Java Message Service (JMS) API was developed by Sun Microsystems with large contributions by IBM and other vendors. JMS is only a specification to describe usage but requires an implementation for use. WebSphere MQ is IBM's implementation of JMS and fully supports the JMS 1.1 specification when using WebSphere MQ 5.3 with *Corrective Service Disk* (CSD) 06.

The JMS specification is now included in the J2EE 1.3 specification, meaning that a J2EE application server vendor must also include a JMS implementation.

IBM's J2EE 1.3 compliant application server, WebSphere Application Server, implements the JMS requirement with two different methods. For WebSphere Application Server releases other than the Express Edition, a *JMS Server* is included through the *Embedding Messaging Server and Client Services*. In addition, with *WebSphere Application Server Enterprise Edition* a full version of WebSphere MQ is also provided.

When the Embedding Messaging Service is used as the JMS Server, it is licensed to be used only by WebSphere Application Server for JMS messaging. It is not to be used by non-JMS applications. The Embedded Messaging Service is commonly referred to as the WebSphere Application Server JMS Provider. When a full installation of WebSphere MQ is used, it is referred to as the WebSphere MQ JMS Provider by the WebSphere Application Server installation. For this book, we use only a full version of WebSphere MQ and do not discuss Embedded Messaging in detail.

**Tip:** While the Embedded Messaging Service could be sufficient for introductory usage or J2EE-to-J2EE only development, most major production implementations should use the full WebSphere MQ queue manager as the JMS provider because of the advanced management tooling and monitoring. In addition, the need to communicate with native applications is likely to be required.

For assistance with choosing a messaging implementation that meets your organization's needs, see the IBM publication *Selecting the most appropriate JMS provider for your applications*, G325-2318-00.

For further information about JMS, see the IBM publication *WebSphere MQ Using Java*, SC34-6066-02, and the IBM Redbooks:

- ▶ *MQSeries Programming Patterns*, SG24-6506 (ISBN 0738423661)

<http://www.redbooks.ibm.com/abstracts/sg246506.html?open>

- ▶ *MQSeries Publish/Subscribe Applications*, SG24-6282 (ISBN 0738423149)  
<http://www.redbooks.ibm.com/abstracts/sg246282.html?open>
- ▶ *Programming J2EE APIs with WebSphere Advanced*, SG24-6124 (ISBN 0738422975)  
<http://www.redbooks.ibm.com/abstracts/sg246124.html?open>

For further information about WebSphere Application Server Embedded Messaging comparison and interoperability with WebSphere MQ, see the IBM Redbook: *WebSphere Application Server and WebSphere MQ Family Integration*, SG24-6878 (ISBN 0738427624).

<http://www.redbooks.ibm.com/abstracts/sg246878.html?open>

The benefits of JMS are as follows:

- ▶ Both point-to-point and publish/subscribe messaging models are available, and are both mapped to the same simple underlying structure.
- ▶ Applications written correctly in JMS can be moved between messaging providers, and therefore burden of infrastructure and platform knowledge is removed from the developer.
- ▶ High-level message classes remove the complexity of byte-level data management.
- ▶ Message selection allows receivers to filter incoming messages based on logical constructs.
- ▶ Asynchronous message delivery (callback when a message is available) is provided as an option.
- ▶ There is a choice of transactional semantics: client-controlled acknowledgement, locally transacted, or external transaction coordination.

**Tip:** For the complete description of the Java Message Service version 1.1. Specification, visit:

<http://java.sun.com/products/jms/docs.html>

## 1.4 Message broker concepts

The purpose of a broker is to take incoming messages from applications and perform some action on them before sending them on to their destination application. The following are examples of actions that might be taken in the broker:

- ▶ Route messages to one or more of many destinations.
- ▶ Transform messages to an alternative representation.
- ▶ Interact with an external repository to augment a message or store it.
- ▶ Invoke Web services to retrieve data.
- ▶ Respond to events or errors.
- ▶ Provide content and topic-based routing using the publish/subscribe model.
- ▶ Perform message aggregation, decomposing messages into multiple messages and send them to their destination, then recomposing the responses into one message to return to the user.

## 1.5 Introducing WebSphere BI Message Broker family

WebSphere BI Message Broker consists of the following installable components illustrated in Figure 1-6 on page 23:

- ▶ **WebSphere BI Event Broker**  
The WebSphere BI Event Broker provides publish/subscribe capabilities, including content and topic-based message routing. If you only require the WebSphere BI Event Broker features, you can install this component without installing the full WebSphere BI Message Broker component.
- ▶ **WebSphere BI Message Broker**  
The WebSphere BI Message Broker includes the capabilities of the WebSphere BI Event Broker. In addition it provides a broad range of broker functionality, including message transformation capabilities, content dependent message routing, and database integration for logging and message enrichment.
- ▶ **Message Brokers Toolkit for WebSphere Studio**  
The third major component is the Broker Toolkit. This Eclipse-based Workbench tool provides both the development environment and the broker administration environment. The Broker Toolkit has the same look and feel as the whole line of WebSphere Studio integrated development products.

**Tip:** For the remainder of this book, the broker version used is referred to as the WebSphere BI Message Broker because message transformation, content routing and database integration are demonstrated along with publish/subscribe applications.

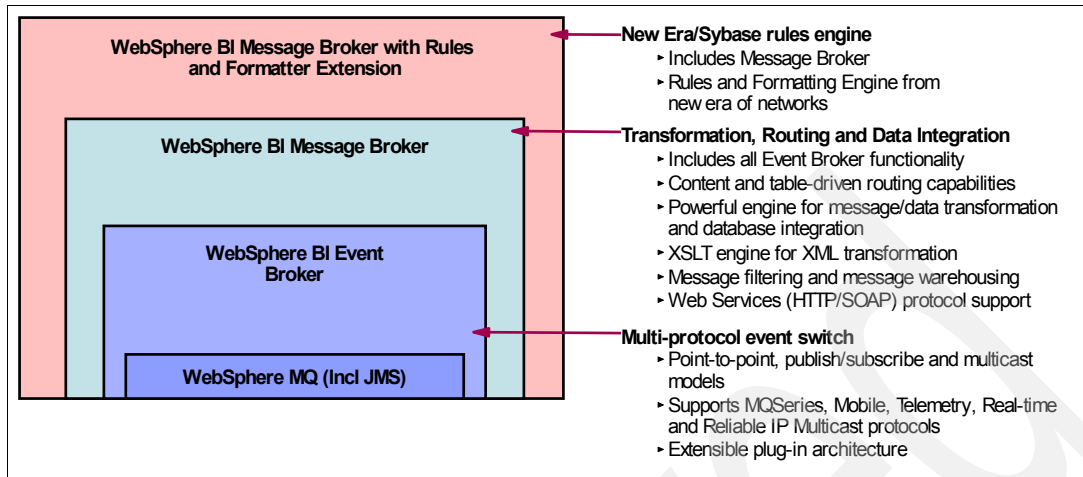


Figure 1-6 WebSphere BI Message Broker family

## 1.5.1 Architecture overview

The purpose of a broker is to take incoming messages from applications and perform some action on them before sending them on to their destination application. The following are examples of actions that might be taken in the broker:

- ▶ Route messages to one or more of many destinations.
- ▶ Transform messages to an alternative representation.
- ▶ Interact with an external repository to augment a message or store it.
- ▶ Invoke Web services to retrieve data.
- ▶ Respond to events or errors.
- ▶ Provide content and topic-based routing using the publish/subscribe model.
- ▶ Perform message aggregation, decomposing messages into multiple messages and send them to their destination, then recomposing the responses into one message to return to the user.

WebSphere BI Message Broker provides both the runtime and development environment necessary to provide broker functionality. Actions taken on messages are done by message flows executing in the runtime component. As messages arrive from applications over a supported transport, they are processed by the appropriate message flow, then sent on to their destination.

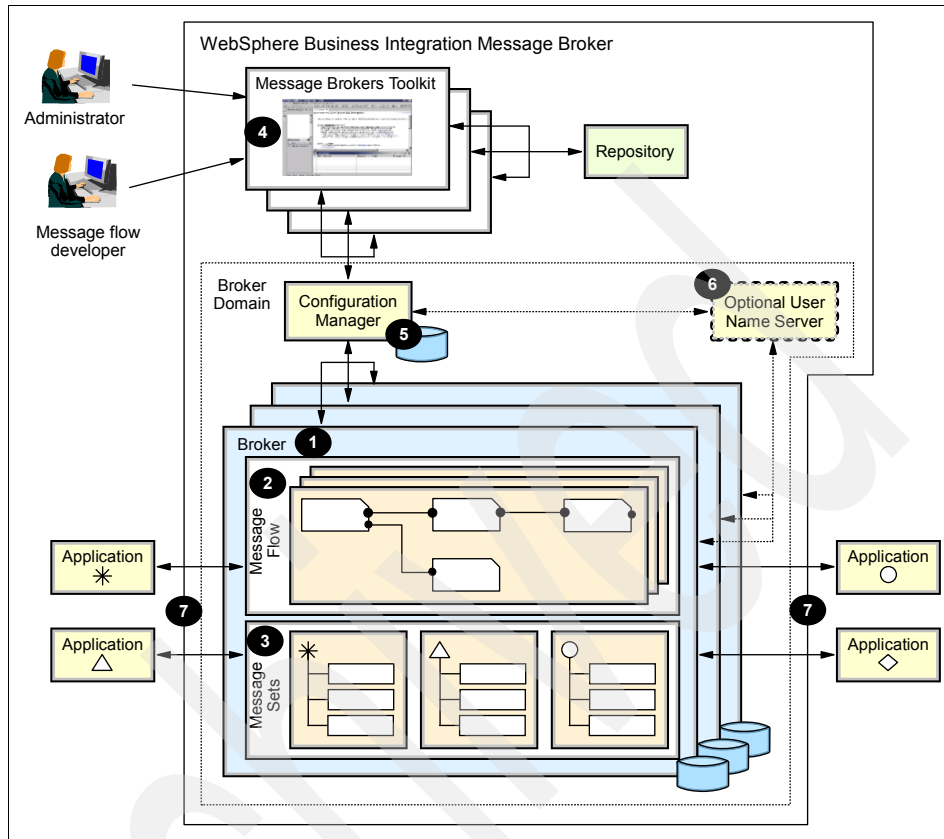


Figure 1-7 WebSphere BI Message Broker architecture

1. The primary runtime component is the broker. Brokers contain a number of *execution groups*, processes in which message flows are run. Each broker uses a database to store the information it needs to process messages at runtime.
2. Messages are processed by message flows. *Message flows* are developed to provide specific functionality by wiring a series of nodes together. Each node has a specific job to do within the scheme of the message flow. Nodes for input and output are designed to take the messages from specific transport types. Other nodes can perform computations, message enhancement, or make routing decisions.
3. Messages must have a defined structure which is known and agreed by the sender and the receiver. In order for the broker to process messages it must also understand their format. A *message set* contains message definition files that describe the messages. As the messages are processed, message flows



apply the appropriate message definitions to them at various stages during its progress along the flow.

4. The Broker Toolkit provides an integrated development environment for message flow development and runtime administration. Message flows and message sets can be developed using the Broker Toolkit, then packaged for execution, and deployed to the runtime environment using a connection established between the Workbench and the Configuration Manager.
5. The Configuration Manager coordinates all activity between the Workbench and brokers within its domain. An example of such a change would be changes to a message set. Brokers are grouped into broker domains. Each domain is coordinated by a Configuration Manager. It uses a database as a repository to store information relating to its broker domain.
6. If you have applications that use the publish/subscribe services of a broker, you can apply an additional level of security to the topics on which messages are published and subscribed. This additional security, known as topic-based security, is managed by the User Name Server. It provides administrative control over who can publish and who can subscribe.
7. Transport support facilities provide the interface between the client applications and the message flows.

WebSphere MQ queue managers provide the underlying transport infrastructure for the WebSphere BI Message Broker. WebSphere MQ messaging is used between the Workbench, the Configuration Manager, and the brokers. WebSphere MQ can also be used for communication between applications and brokers as well.

## 1.5.2 Runtime environment

Figure 1-8 shows an overview of the runtime environment.

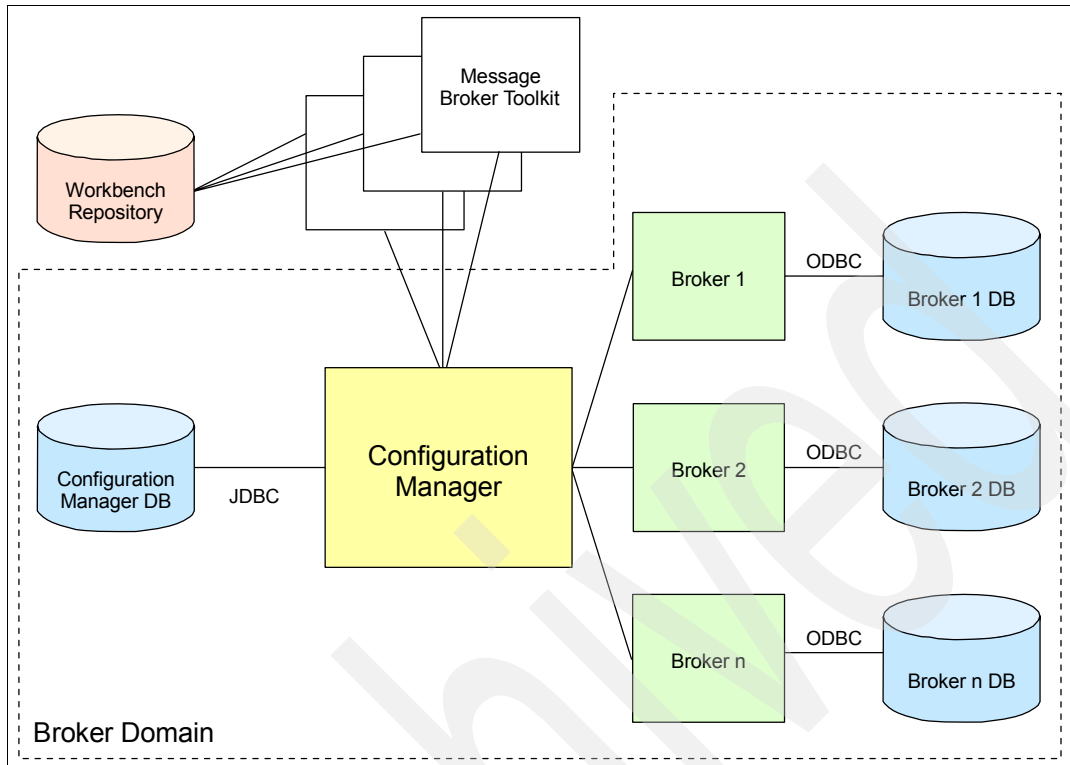


Figure 1-8 Configuration Manager relationships

## Configuration Manager

The Configuration Manager maintains the broker domain configuration and provides the interface between the Workbench, the configuration repository and an executing set of brokers. In addition, the Configuration Manager is required to deploy message flows, message sets, and domain configuration information to a broker. The Configuration Manager has four main functions:

- ▶ Maintains configuration details in the configuration repository, a set of DB2® database tables that provide a central record of the broker domain components.
- ▶ Deploys WebSphere BI Message Broker resources from the Workbench to the broker domain.
- ▶ Reports to the Workbench on the results of deployment operations and the status of the brokers.
- ▶ Communicates with the other components in the broker domain through WebSphere MQ.

For each broker domain you must create and start one, and only one, Configuration Manager. The Configuration Manager can only be created on Windows systems and uses a JDBC connection to a DB2 database for its configuration repository.

Create the Configuration Manager using `mqsicreateconfigmgr` from the Window's command prompt, or using the **create a default configuration** hyperlink from the Toolkit Getting Started page.

As part of the Configuration Manager creation process, the repository tables are created in the DB2 database and a set of WebSphere MQ queues are defined on the queue manager which hosts the Configuration Manager. These queue names all begin with SYSTEM.BROKER. The rest of the name will indicate the purpose of the queue and be unique. The hosting queue manager must be on the same Windows machine as the Configuration Manager.

A single server connection channel (SYSTEM.BKR.CONFIG) is created to allow the Workbench to communicate with the Configuration Manager. You will need to create manually sender/receiver channels to brokers that are hosted by remote queue managers to allow you to administer them from the local Configuration Manager. Alternatively, you could put the all the broker queue managers in a WebSphere MQ cluster with the Configuration Manager queue manager.

Message flow development can be done in the Broker Toolkit without an active connection to a Configuration Manager but a connection must be available to be able to deploy and test message flows on a broker.

## Broker

A *broker* is the named resource that executes the business logic defined in the message flows. Applications send and receive messages to and from a broker using WebSphere MQ queues, WebSphere MQe, Web services or HTTP as the methods of communication. The messages used in a broker are usually defined in a message set.

More than one broker can be defined for each broker domain, either on the same or on a different physical system. If you plan to use the publish/subscribe service, you can connect a number of brokers together into a collective, allowing optimization of the subscriber connections.

Brokers contain execution groups that run as separate operating system processes, providing an isolated runtime environment for a set of deployed message flows. Flows running within an execution group are threads. By default, only one execution group is defined, but more can be easily added.

Message sets and message flows are packed into a broker archive file (BAR file) for deployment. You can logically group message flows and message sets by placing them in different BAR files. Several different BAR files can be assigned to an execution group, although using different execution groups provides better application isolation. A particular message flow can be configured to run more than one instance of itself inside the execution group. For even more isolation between applications, you can use different brokers.

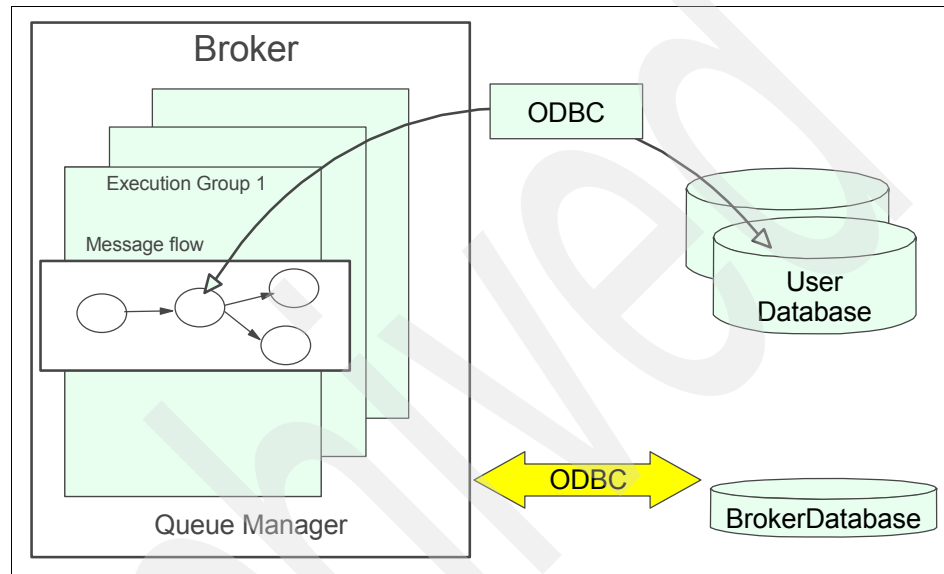


Figure 1-9 Broker overview

Each broker, as in Figure 1-9, requires:

- ▶ A set of tables in a database to hold the broker's local data. This set of tables is accessed through an ODBC connection. They can be created using a number of database products depending on the broker platform:
  - DB2 UDB
  - Microsoft® SQL Server (Windows only)
  - Oracle
  - Sybase
- ▶ A WebSphere MQ queue manager. A broker can share the queue manager hosting either the Configuration Manager or the optional User Name Server, or both. However, since the broker uses a set of predetermined queue names, brokers can't share a queue manager with another broker.
- ▶ A set of named queues on the queue manager associated with the broker domain. These are created automatically when the `mqsicreatebroker` command is issued.

- ▶ For operation, a broker requires both configuration and initialization data. Configuration and initialization data are logically separate and stored in different physical repositories.

Each broker instance has an assigned, permanent and fixed name. This is the broker instance name. This name, which is similar to the static identifier assigned to a database before it is created, is used to distinguish tables pertaining to one broker from other tables where multiple brokers have been set up using the same database.

Creating a broker on the target execution platform does not itself update the configuration repository; you need to create a reference to it using the Message Brokers Toolkit. Once this is done and the broker is deployed, message flows and sets can be assigned to the broker. Brokers provide published event messages in response to changes; these changes can be used by the Broker Administration perspective to update the management agents as to the status of the broker components.

**WebSphere MQ MA0C:** WebSphere MQ Publish/Subscribe product extension is not covered in this redbook because its performance and scalability are not of the same standard as WebSphere Business Integration Message Broker (although it is a perfectly functional queue manager-based publish/subscribe engine). WebSphere MQ MA0C is covered in the redbook *MQSeries Publish/Subscribe Applications*, SG24- 6482 at this link:

<http://www.redbooks.ibm.com/redbooks/pdfs/sg246282.pdf>

In Chapter 7, “Implementing highly available brokers” on page 199 there is a more detailed description of developing and deploying the runtime of WebSphere BI Message Broker.

## 1.6 Introducing WebSphere Application Server

Over the past several years, many software vendors have collaborated on a set of server-side application programming technologies that help build Web-enabled, distributed, multiplatform applications. These technologies are collectively branded as the Java 2 Platform, Enterprise Edition (J2EE) platform. This contrasts with the Java 2 Standard Edition (J2SE) platform, which most customers are familiar with and that supports the development of client-side applications with rich graphical user interfaces (GUIs). The J2EE platform is built on top of the J2SE platform. It consists of application technologies for defining business logic and accessing enterprise resources such as databases, Enterprise Resource Planning (ERP) systems, messaging systems, e-mail servers, and so forth.

The potential value of J2EE to the customer is tremendous. Among the benefits of J2EE are the following:

- ▶ Promotion of an architecture-driven approach to application development helps reduce maintenance costs and allows for construction of an Information Technology (IT) infrastructure that can grow to accommodate new services.
- ▶ Application development is focused on unique business requirements and rules, rather than common application aspects, such as security and transaction support. This improves productivity and shortens development cycles.
- ▶ Industry standard technologies allow customers to choose among platforms, development tools, and middleware to power their applications.

Embedded support for Internet and Web technologies allows for a new breed of applications that can bring services and content to a wider range of customers, suppliers, and others, without creating the need for proprietary integration.

WebSphere Application Servers are a suite of servers that implement the J2EE specification. This simply means that any Web applications that are written to the J2EE specification can be installed and deployed on any of the servers in the WebSphere Application Server family.

WebSphere Application Servers are available in multiple configurations to meet specific business needs. They also serve as the base for other WebSphere products, such as WebSphere Commerce, by providing the application server required for running these specialized applications.

WebSphere Application Servers are available on a wide range of platforms, including UNIX-based platforms, Microsoft operating systems, IBM z/OS®, and iSeries™. Although branded for iSeries, the WebSphere Application Server products for iSeries are functionally equivalent to those for the UNIX and Microsoft platforms.

### **Highlights and benefits**

WebSphere Application Server provides the environment to run your Web-enabled e-business applications. You may think of an application server as *Web middleware*, or a middle tier in a three-tier e-business environment. The first tier is the HTTP server that handles requests from the browser client. The third tier is the business database and the business logic (for example, traditional business applications such as order processing). The middle tier is IBM WebSphere Application Server, which provides a framework for consistent, architected linkage between the HTTP requests and the business data and logic.

IBM WebSphere Application Server is intended for organizations that want to take advantage of the productivity, performance advantages, and portability that Java provides for dynamic Web sites. It includes:

- ▶ Java runtime support for server-side Java servlets
- ▶ Support for the Enterprise JavaBeans specification
- ▶ High-performance connectors to many common back-end systems to reduce the coding effort required to link dynamic Web pages to real line-of-business data
- ▶ Application services for session and state management
- ▶ Web Services enable businesses to connect applications to other business applications, to deliver business functions to a broader set of customers and partners, to interact with marketplaces more efficiently, and to create new business models dynamically.

For more information about WebSphere Application Server visit

<http://www.ibm.com/websphere>

In addition there are two IBM Redpapers that provide very complete overviews:

- ▶ *WebSphere Application Server V5 Architecture* (REDP-3721-00)
- ▶ *WebSphere Product Overview* (REDP-3740-00)

Archived





## Linux systems and advanced technologies

This chapter discusses some advanced concepts and technologies related to high availability and scalability and some technologies that provide these features to a Linux solution.

## 2.1 Scalability versus high availability

Scalability and high availability are related to each other, but are different. In this section we describe these two concepts.

### 2.1.1 Scalability

*Scalability* means the ability to add power and expand the capabilities and the complexity of a computing solution without making major changes to the systems or application software.

#### Vertical scalability

*Vertical* scalability is the ability to increase existing capacity of the hardware or software by adding resources or upgrading the existing resources to newer technologies. Examples of vertical scaling include adding RAM, CPUs, networking bandwidth and I/O devices to make the server faster.

#### Application vertical scaling

Applications can also scale vertically if broken into components, each for specific types of requests. E-mail servers are good example of a solution that can be implemented easily on a vertical configuration. Instead of having one server to take care of incoming mail, outgoing mail and POP/IMAP, you configure three separate servers to each serve one of these tasks, increasing the throughput of your e-mail system. See Figure 2-1.

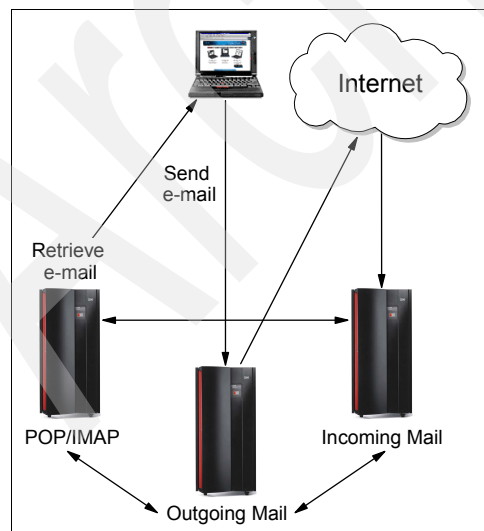


Figure 2-1 Application vertical scaling

## Horizontal scalability

Horizontal scalability is the ability to connect multiple hardware or software entities, so that they work as a single, logical unit. With servers, for example, you could increase the speed or availability of your solution by adding more servers and using clustering and load balancing technologies. On a zSeries this could be achieved simply by adding Linux virtual machines. See Figure 2-2.

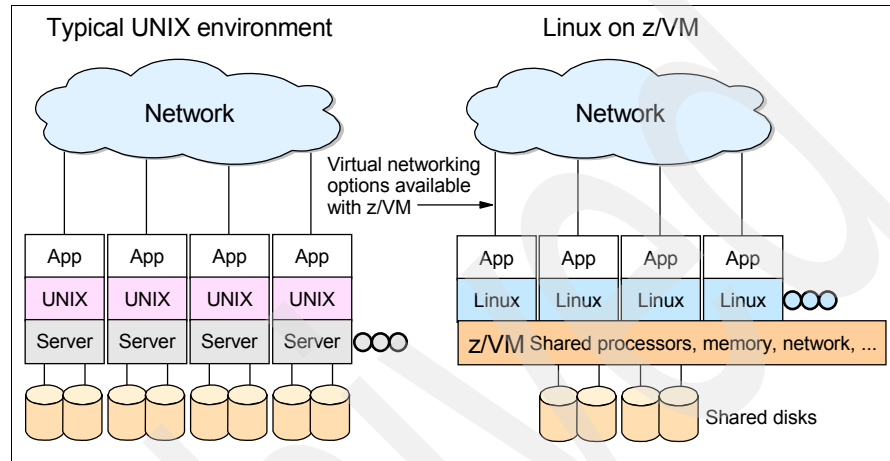


Figure 2-2 Horizontal Server growth on mainframe

### 2.1.2 High availability

*High availability*, a resource's ability to withstand a hardware or software failure, depends on the resource's reliability. It can be achieved by using resource duplication to remove *single points of failure* and adding automatic error detection and correction features. In today's IT world, businesses aim for at least 99.9% application availability. The availability of the node running the application is a combination of several things, which all have to be available:

- ▶ Availability of the hardware
- ▶ Availability of the operating system
- ▶ Availability of the application
- ▶ Availability of the network

To calculate the overall availability of the solution, we need to find the availability of each of the components and multiply them all together. For information about how to calculate the availability of your solution, read the article "Planning for Availability in the Enterprise", on the *IBM WebSphere Developer Technical Journal* Web site:

[http://www-128.ibm.com/developerworks/websphere/techjournal/0312\\_polozoff/polozoff.html](http://www-128.ibm.com/developerworks/websphere/techjournal/0312_polozoff/polozoff.html)

## **Hardware high availability**

While mainframes deliver high availability with characteristics such as error detection and reliable redundant hardware components, Intel and Power based servers do not have all these redundancies built in. In these systems, high availability has to be achieved with installation of, for example:

- ▶ Redundant network/host adapters
- ▶ Redundant network hardware
- ▶ Redundant power supplies and UPS
- ▶ Hardware RAID (Redundant Array of Inexpensive Disks)
- ▶ Redundant servers or server farms

## **Application and operating system high availability**

However, even with all the redundancy built into our hardware, we might still experience a server outage. In this case we want to be able to keep our services and application available for the user. There are several ways to ensure application high availability for example:

- ▶ Application and service monitoring and automated responses
- ▶ Application clustering with load balancer
- ▶ Redundant servers or server farms

In a typical set-up, a combination of these methods is used in order to achieve an acceptable level of high availability for the hardware, operating system and application. Figure 2-3 on page 37 is an example of a simplified redundant hardware setup.

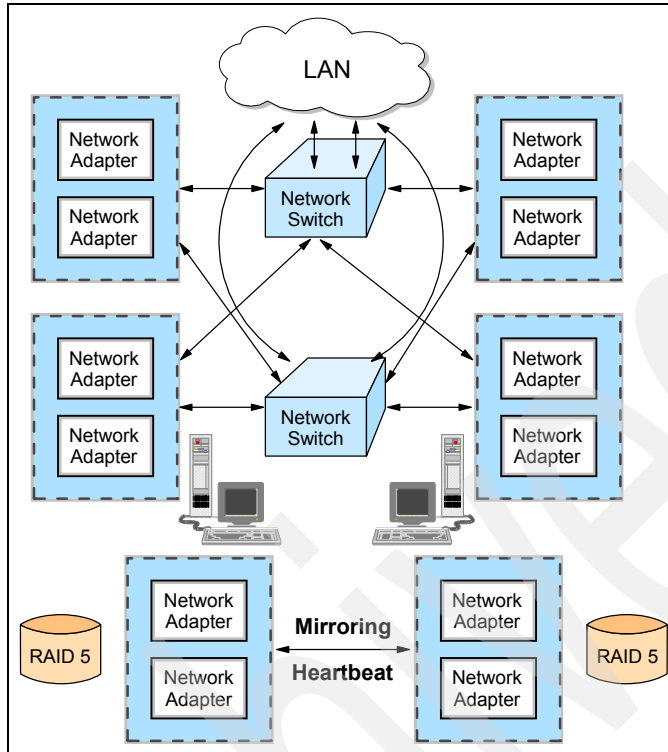


Figure 2-3 Redundant hardware setup

## 2.2 Linux technologies for scalability

This section discusses technologies that are relevant for making a Linux solution scalable.

### 2.2.1 Storage area network

A *Storage Area Network* (SAN) is a high-speed subnetwork of shared storage devices connected through peripheral channels such as SCSI, SSA, ESCON® and Fibre Channel instead of the network. SAN ties multiple hosts into a single storage system, a RAID device with large amounts of cache and redundant power supplies. Fiber Channel protocol makes local as well as campus- and metropolitan-wide hookups and remote copying of the data for disaster recovery possible. It is commonly employed to tie a server cluster together for failover. Because stored data does not reside directly on any of a network's servers, server power is used for business applications, and network capacity is released

to the end user. A SAN also offers improvements to application availability because storage is independent of the application and accessible through multiple data paths. SANs often consist of several types of servers running different operating systems. This enables users from a wide variety of platforms to access common storage information. The result is a single and highly scalable storage system which simplifies management and increases overall system performance.

For more a more thorough introduction into SAN systems, refer to the IBM Redbook *Introduction to Storage Area Networks*, SG24-5470.

### **Network attached storage (NAS)**

A *network-attached storage* device is a server that is dedicated to file sharing. It does not provide any of the services that a server in a server-centric system typically provides, such as e-mail, authentication or file management.

NAS allows more hard disk storage space to be added to a network without the need to shut down servers for maintenance and upgrades as storage is not an integral part of the server, providing simplified management and scalability.

## **2.2.2 IBM hardware offerings for Linux scalability**

This section discusses a few hardware offerings from IBM that provide scalability for a Linux solution.

### **IBM eServer clusters**

Cluster solutions 1300 and 1350 offer a combination of IBM @server xSeries® rack-optimized servers, storage solutions including BladeCenters and FastTx and Myrinet-2000 cluster interconnect. They are made for high performance, scientific and technical workloads, as well as commercial computing workloads. Configurations support specific versions of Red Hat and SUSE Linux operating systems.

Cluster solutions 1600 are made for large-scale computational modeling, multiterabyte databases and cost-effective data center, server and workload consolidations. They support a wide range of IBM @server pSeries®, Myrinet-2000 cluster interconnect and specific versions of SUSE Enterprise Linux and Turbo Linux.

### **IBM eServer BladeCenters**

Slim, hot-swappable IBM @serverblade servers each consist of processors, memory, storage and network controllers. They plug into a single chassis and share power, fans, floppy drives, CD drivers, switches and ports.

HS20 and HS40 are 2-way and 4-way Intel-based solutions. JS20 is a 2-way POWER™-based solution.

### **IBM's SAN and NAS offerings**

IBM offers a full range of SAN devices including: Fibre Array Storage Technology (FAStT) servers, fiber channel hubs, switches, tape gateways and routers, NAS and iSCSI products.

For more information, visit the IBM TotalStorage® support Web site:

<http://www-1.ibm.com/servers/storage/support/>

## **2.2.3 IBM software offerings for Linux scalability**

This section discusses a few software offerings from IBM that provide scalability for a Linux solution.

### **Extreme Cluster Administration Toolkit**

*xCAT* is a collection of scripts written in languages like Korn shell and Perl combined with tables and relevant commands, making it easy to modify the functions if needed. It was created for administration of Linux HPC and horizontal scaling, but the utility set grew and it became a proving ground for the new Linux clustering concepts and best practices. It supports various versions of Red Hat Linux and SUSE. Some of its main features are:

- ▶ *Automated installation* allows you to automate the installation of several machines in parallel.
- ▶ *Hardware management and monitoring* allows you to perform remote power control and network BIOS and firmware updates, configurations and access to POST/BIOS console on most IBM hardware. Remote monitoring for vitals such as fan speed and temperature, as well as remote access to the hardware event logs is available.
- ▶ *Software administration* allows you to run automated setup and installation for Myrinet. Command line utilities for all cluster management functions, parallel remote shell, ping, rsync and copy are also included.

*xCAT* is a free tool for IBM Linux cluster customers. For more information and download, visit:

<http://www.alphaworks.ibm.com/tech/xCAT/>

## Cluster Systems Management

CSM provides a robust, powerful and centralized way to manage large numbers of IBM's cluster hardware solutions from a single point-of-control. Its features are combined from PSSP for AIX®, xCAT and other best practices for clustering and include:

- ▶ Manage node and node group information  
With this feature you can add and delete nodes from a cluster and keep track of their status.
- ▶ Install, maintain and setup software  
You can perform remote network installations of the cluster nodes as well as run software updates.
- ▶ Control hardware  
This feature gives you the capability to power the system on and off, and reboot and query the node status in the cluster.
- ▶ Distributed command execution  
enables to run commands parallel across nodes or node groups in the cluster and gather their output.
- ▶ Configuration file management  
This feature synchronizes and maintains the consistency in files across the cluster nodes.
- ▶ Event monitoring and automated responses  
With this feature you can monitor for various conditions such as network, power status and CPU, disk and file system utilization, across the nodes in the cluster and perform actions in response to these events.
- ▶ Security  
This feature consists of a security authentication mechanism host-based public key/private key setup.
- ▶ Interoperability with CSM for AIX 5L™  
This feature allows mixed clusters of xSeries running Linux and pSeries running either Linux or AIX.
- ▶ Integration with IBM Director  
Both products can be used together to manage machines from a single graphical console sharing information and events.

For more information, visit:

<http://www-1.ibm.com/servers/eserver/clusters/software/>



### **Enhanced Cluster Tools**

ECT is package providing extra functionality and utilities to enhance CSM. Some of the tools include:

- ▶ Tools to setup HPC stack
- ▶ Scripts to configure terminal servers and RSAs
- ▶ Tools for turning SNMP events into CSM events
- ▶ Web interface to CSM
- ▶ xCAT-to-CSM migration tool

For more information, visit:

<http://www.alphaworks.ibm.com/tech/ect4linux>

## **2.2.4 Open source offerings for Linux scalability**

This section discusses a few open source software offerings that provide scalability for a Linux solution.

### **OSCAR**

OSCAR is a collection of open source tools that can be used to build a moderate sized, high performance cluster, better known as a *Beowulf* cluster. For more information and download, visit:

<http://oscar.openclustergroup.org>

### **Linux Virtual Server Project**

From the Web site, the basic goal of the Linux Virtual Server Project (LVS) is: *Produce a high-performance and highly available server for Linux based on clustering, which provides a good scalability, reliability and serviceability.*

LVS provides load balancing for TCP and UDP sessions running on multiple, real servers by implementing Layer-4 switching in the Linux Kernel. Scaling HTTP and HTTPS services for the World Wide Web is beyond doubt its most common usage, but it can be used for more or less any service from e-mail to X windows. The solution is very high in performance and can be used to handle thousands of simultaneous connections. It has virtually unlimited scalability and can be used easily together with high availability tools such as heartbeat.

Real servers, hosts running Linux daemons such as Apache, can be connected by highspeed LAN or they can reside in geographically different locations. The front-end acts as a load balancer, distributing requests to the other servers and makes parallel services of the cluster appear as served by a single IP-address. As a result, adding or removing servers is transparent for the users. See Figure 2-4 on page 42.

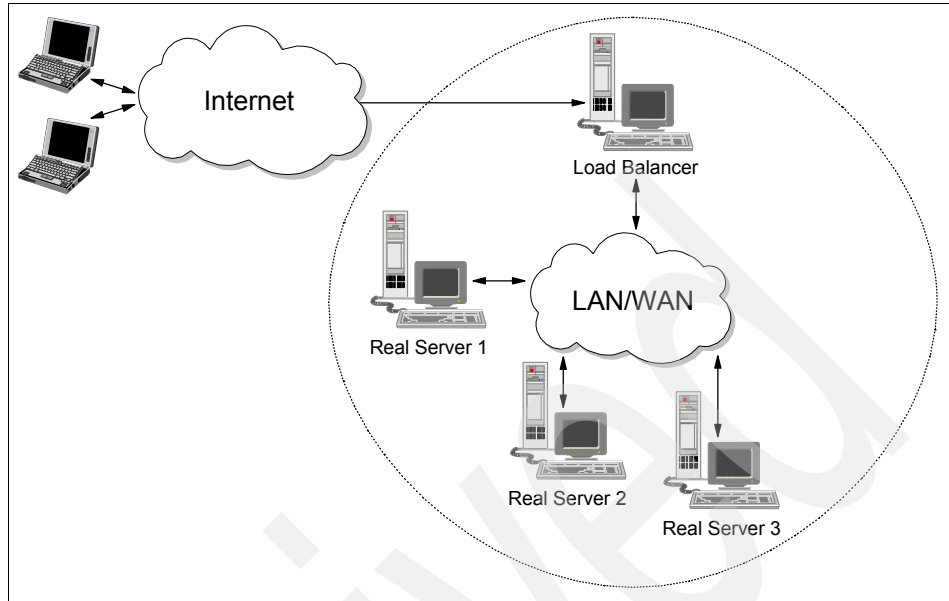


Figure 2-4 Linux Virtual Server

For more information about the project and download, please visit:

<http://www.linuxvirtualserver.org/>

## 2.3 Linux technologies for high availability

This section discusses some technologies that are relevant for making a Linux solution scalable.

### 2.3.1 Journalled file systems

Journaling file systems offer much faster system recovery in the event of system failure than nonjournalled file systems such as HPFS and ext2. Instead of relying on boot time utilities like `fsck` to examine the file system's metadata in order to restore the file system to a consistent state, they simply read and apply a transaction log.

#### The Journalled File System

The Journalled File System (JFS) for Linux is an IBM open source project licensed under the GNU GPL. It has been developed with transaction-oriented, high performance systems in mind. It is scalable and robust, providing a

log-based, byte-level file system that was designed to include fully integrated journaling from the beginning rather than adding the journaling feature to an existing file system. Its journaling techniques were originally developed for databases, which log operations performed on the data records, although JFS only implements logging of metadata. Some of its other features include:

- ▶ Dynamic disk inode allocation

JFS dynamically allocates space for disk inodes as required and frees the space when it is no longer required. This approach eliminates the need for reserving a fixed amount of space for disk inodes at the file system creation time, hence it will not be necessary for users to estimate the maximum number of files and directories on a file system.

- ▶ Variable block size

Block sizes of 512, 1024, 2048 and 4096 are supported on an individual file system basis, allowing a user to select the appropriate size for his application set. A smaller block size reduces file fragmentation and is more space-efficient. On the other hand, space allocation occurs more often than if a large block size were to be used, reducing performance.

JFS is included in most of the major distributions: Red Hat, SUSE, Debian and Turbo Linux. For free download and more information about the JFS file system and its features, visit:

<http://www-124.ibm.com/developerworks/opensource/jfs/>

## **General Parallel File System for Linux**

GPFS is IBM's high performance cluster file system. Its main features are:

- ▶ Scalable file system

The system is designed for clusters where the number of nodes and the file system size can be large. It allows expansion of file system throughput and capacity as service demands increase.

- ▶ Parallel file system

Employs multiple nodes of a cluster to store blocks of a single file. Exploits parallel I/O to circumvent bottlenecks that might be present if a file resided on a single node.

- ▶ Journaling file system

Logs information about operations performed on the file system metadata as atomic transactions that can be replayed. Provides rapid restoration of file system to consistent state after system crash.

- ▶ Shared file system  
Implemented so that multiple clients can concurrently access a file and includes a sophisticated token management system to ensure data consistency. Allows nodes running parallel applications to concurrently access a file and thereby reduces processing time.
- ▶ High availability file system  
Provides for automated recovery from events that would normally interrupt data availability. Applications and users can continue without interruption.
- ▶ Client side data caching  
Keeps records of recent I/Os in client memory. Improves performance by reducing need for repetitive processing of requests for frequently accessed files and also hides write latency by using "write behind".
- ▶ Large Blocksize options  
Blocksizes of 16K, 64K, 256K, 512K or 1024K can be defined. Provides for efficient use of disk bandwidth and disk space, particularly important for RAID devices.
- ▶ Access pattern recognition and deep prefetch  
Detects access pattern as sequential, random, fuzzy sequential or strided, and prefetches striped data in parallel. Enables high throughput to be sustained, by overlapping disk latencies.
- ▶ Data protection  
Both metadata and user data can be replicated in the GPFS installation. Provides protection from loss of data and metadata.

GPFS supports two separate models of disk attachment and a combination of the two is also allowed. This is very useful if, for example, you wanted to create a disaster recovery site by duplicating your data across two physical sites.

### ***Directly attached model***

All the nodes are directly connected to the storage with a SAN switch, for example. See Figure 2-5 on page 45.

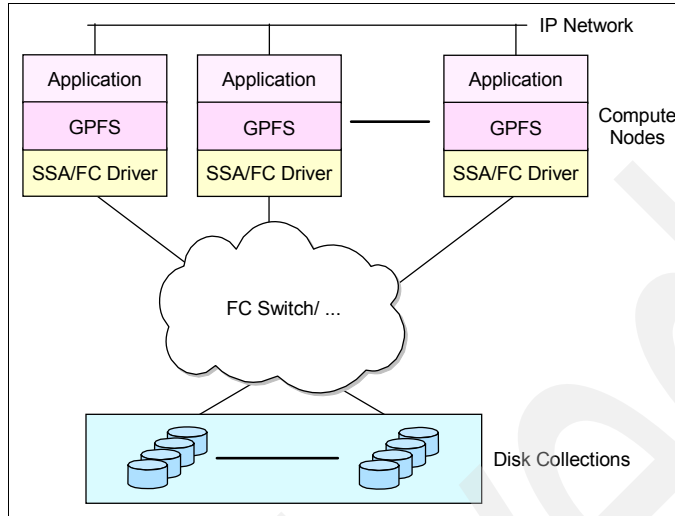


Figure 2-5 GPFS in a direct attached SAN environment

### Network shared disk model

When not every system in a cluster can have a SAN connection to the disks, a network shared disk (NSD) is used to simulate SAN in software. NSD is responsible for the abstraction of the disk data blocks across either a Myrinet or IP-based network. See Figure 2-6.

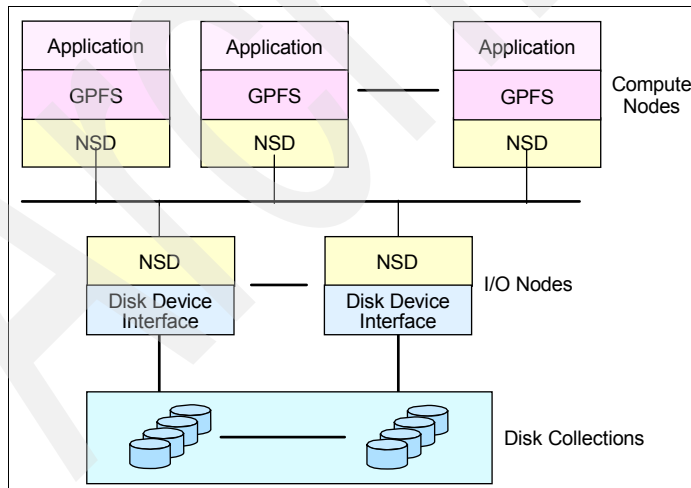


Figure 2-6 GPFS in a Network Shared Disk environment

For more information, visit:

<http://www-1.ibm.com/servers/eserver/clusters/software/gpfs.html>

## Open Global File System

OpenGFS or OGFS is a Journalled file system that supports simultaneous sharing of common storage devices by multiple computer nodes. In addition to supporting SANs it also supports cheaper shared SCSI, HyperSCSI and firewire storage devices. Its main features are:

- ▶ *Concurrent access* to storage from all nodes.
- ▶ *Locking services* prohibiting two nodes from writing on the same storage location simultaneously or one node to read information that is being changed.
- ▶ *Individual journals* for each node preventing one from corrupting the journal of an other. Journaling is provided for metadata only.
- ▶ *Fencing* can isolate a rogue node from storage devices and power down or reboot the node in order to protect the file system from corruption.
- ▶ *Resizing of the file system* feature allows you to expand the file system data storage space and add new journals.

For more information, visit:

<http://opengfs.sourceforge.net>

## ext3

The ext3 file system is a journaling extension to the standard ext2 file system on Linux and provides complete forward and backward compatibility with ext2 without formatting. Journaling modes are configurable to suit for individual needs:

- ▶ *data=writeback* provides limited data integrity guarantee and merely avoids the file system check during a boot.
- ▶ *data=ordered* guarantees that data is consistent with the file system.
- ▶ *data=journal* is used mostly for database operations and requires a large journal. As a result, it takes longer to recover in the event of an unclean shutdown.

For more information, visit:

<http://olstrans.sourceforge.net/release/OLS2000-ext3/>

## 2.3.2 IBM software offerings for Linux high availability

This section discusses a few IBM software offerings that provide high availability for a Linux solution.

### Tivoli® System Automation for Linux

This is a policy-based, self-healing product that offers mainframe-like high availability to Linux applications running on single servers or clusters alike, exploiting existing z/OS, AIX and IBM research technologies. Its main features include:

- ▶ *Resource monitoring* is highly customizable detecting outages effectively and fast.
- ▶ *Application recovery* is a feature that allows cluster-wide restart of components or applications.
- ▶ *Automatic application start/stop/move* capability enables you to take care of cluster-wide relationships, correct start/stop order and required pre and post start/stop actions.
- ▶ *Resource grouping* is used to start, stop and monitor all resources related to an application or group or applications.
- ▶ *Resource relationships* allows you to stipulate things like start order within a group of resources and location relationships.

For more information please visit:

<http://www-306.ibm.com/software/tivoli/products/sys-auto-linux/>

## 2.3.3 Open source offerings for Linux high availability

This section discusses a few open source software offerings that provide high availability for a Linux solution.

### Linux-HA Project

The basic goal of the Linux high availability project is: *Provide a high-availability (clustering) solution for Linux which promotes reliability, availability, and serviceability (RAS) through a community development effort.*

The project consists of two distinct pieces of code:

- |                  |  |
|------------------|--|
| <b>Heartbeat</b> | Implements serial, UDP, and PPP/UDP heartbeats. This code also ships as part of many Linux distributions like SUSE and Debian GNU/Linux. |
| <b>Fake</b>      | Provides IP-address take-over.   |

The code is highly portable and runs also on, for example, Solaris and BSD. For more details, visit:

<http://www.linux-ha.org>  
<http://www.vergenet.net/linux/fake/>

## Monit

*Monit* is a utility that allows you to manage, monitor and take meaningful action such as a process restart or send an e-mail alert in error situations. It can monitor the following:

- ▶ Process for status
- ▶ Files, directories and devices for changes like timestamp, checksum and size
- ▶ Remote host by ping and check port connections and protocols

For more information. visit:

<http://www.tildeslash.com/monit/>

## DRBD

DRBD provides disk mirroring over a dedicated network. Separate software is needed to provide node monitoring for fail-over. In this Active-Passive setup each device has a state, either *primary* or *secondary*. An application runs on the node owning the primary device and accesses it for reads and writes. Every write is written to the secondary device and reads are always carried out locally. In the event the node owning the primary device fails, the secondary device is switched into primary state and the application is started on the other node. When the failed node comes back up, a failback occurs after the synchronization with the primary device. The speed of the failover and failback is determined by the choice of file systems because a nonjournaling file system has to run a check on the device. Additionally the size of the partition on the devices has an effect on the synchronization time. For more information please visit:

<http://www.drbd.org/>

## 2.4 Creating a highly available and scalable solution

Various hardware and software solutions provide the building blocks for scalability, availability, security and *single point of management* control needed in today's infrastructure. See Figure 2-7 on page 49 which shows the software components we use in later chapters to create a highly available WebSphere MQ solution. For many components, multiple technologies offer the desired features such as a journaling file system.

In this chapter, we have divided technologies in two pigeon holes: high availability and scalability. In reality, most of the technologies described cannot



be solely placed in one group because their features can be used to achieve both. As an example of this, we can examine SAN technology:

SAN provides you with the capability to connect multiple hosts to a single storage system and is therefore ideally suited for failover solutions. However, it also gives you almost unlimited scalability of disk storage, transcending the limitations imposed by SCSI.

GPFS is another technology which truly belongs to both categories. It provides you with a distributed file system for scalability and, at the same time, possesses features such as journaling, making it a good choice for high availability solutions.

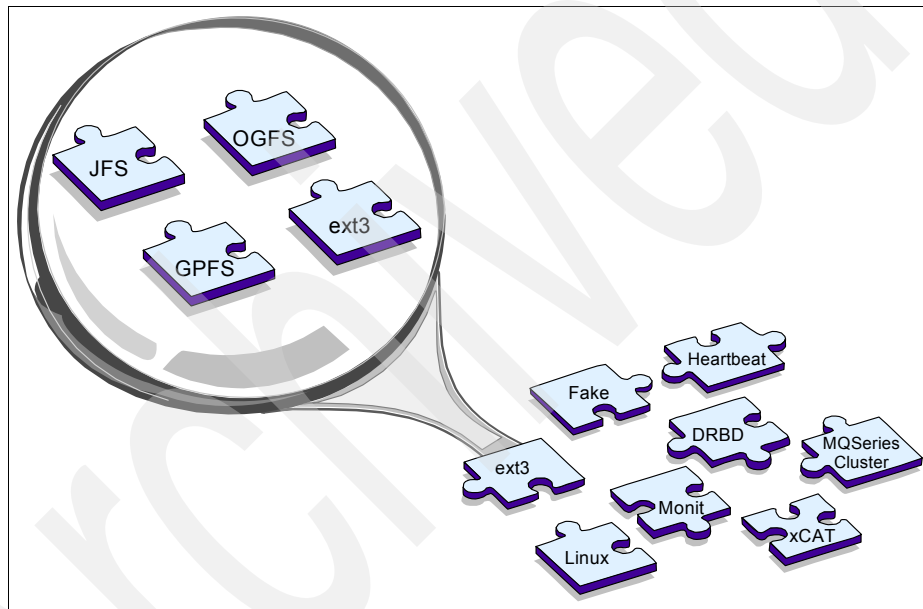


Figure 2-7 Clustering components for high availability

To achieve true redundancy with the ability to scale to any level of demand, you have to break an application down into pieces and serve multiple copies of those pieces. As requests increase, simply add a server in a specific area. Even if growth increases exponentially, you can scale your servers any way that you need to accommodate this growth.

Archived



## Implementing HA queue managers: Part 1

This chapter discusses a WebSphere MQ configuration and two applications that demonstrate implementing a simple queue manager recovery solution with Linux by using disk-mirroring over a network.

## 3.1 Scenario one overview

In this scenario, two variations of a low cost, high-availability solution are implemented. The first variation uses MQ bindings in the client, or receiving, application. The second variation uses MQ client connections to receive messages.

### 3.1.1 Using WebSphere MQ in bindings mode

Messages are put from two queue managers, *qmgr1* and *qmgr2*, using WebSphere MQ *Sender Channels* to *qmgr3* where messages are received from a local queue by a WebSphere Application Server application and displayed as an HTML table. In this version in Figure 3-1, the message-receiving application uses bindings mode to connect to *qmgr3*.

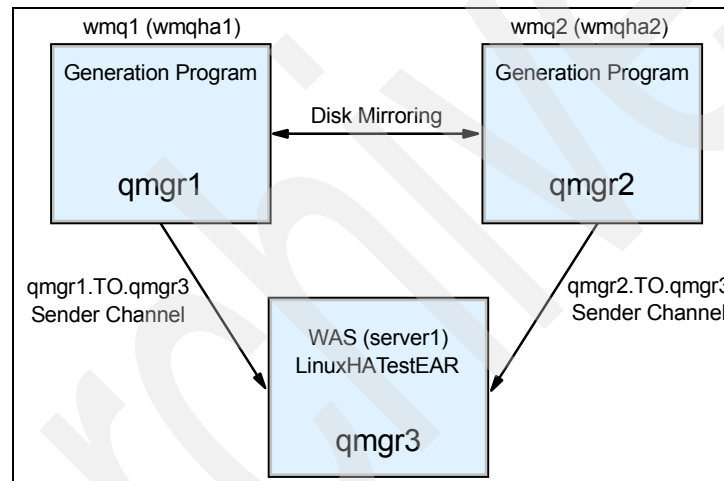


Figure 3-1 High-Availability through disk-mirroring with bindings mode

### 3.1.2 Using WebSphere MQ in client mode

Messages are put to a local queue on *qmgr2*. The same WebSphere Application Server application is used but in client mode to connect to *qmgr2* through a *Server Connection* channel. The messages are again displayed as an HTML table.

See Figure 3-2 on page 53.

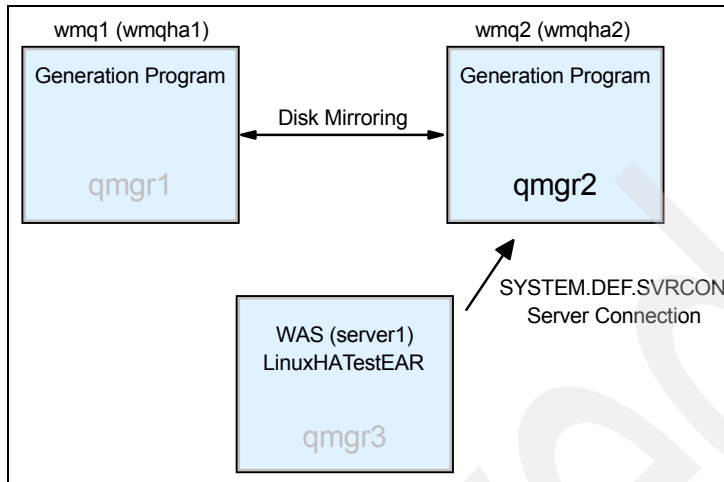


Figure 3-2 High-Availability through disk-mirroring with client mode

## 3.2 Using Linux-HA

This section discusses the implementation of technologies of Linux-HA.

### 3.2.1 Features of Linux-HA

The Linux-HA project provides us with *heartbeat*. Heartbeat, as its name suggests, implements a heartbeat between two nodes and provides a good means of organizing HA resources. HA resources are the services that you want to be highly available, an Apache server or WebSphere MQ for example, but they can also be custom scripts that will be launched during takeover or release. Heartbeat also supports multiple virtual IP addresses and includes the ability to ping other network nodes in addition to the heartbeat partner. This provides the ability to detect upstream network failures and perform a failover even though the heartbeat link may be fine. Virtual IP addresses are not permanently assigned to any real interface, but instead are assigned dynamically to the current master node as an IP alias. Users seeking more information about IP aliases are encouraged to read the IP Alias mini-how to at:

<http://tldp.org/HOWTO/IP-Alias>

For more information about Linux-HA, visit the Linux-HA homepage:

<http://www.linux-ha.org>

At this site you will find many resources to help you on your way and a clear and concise getting started guide which we found to be very useful.

## 3.2.2 Planning an implementation

Our goal for this scenario was to have queue managers running on two separate machines in a high availability cluster, where one machine can take over the job of its downed partner. Furthermore, we wanted to use inexpensive, off-the-shelf hardware to provide this function.

The HA solution we chose consists of two modestly-sized servers with two network cards and two hard drives each. We chose Red Hat Enterprise Linux AS 3.0, but we did not use the custom HA features that are included with this release. We wanted our method to be portable to any distribution, so instead we used heartbeat from Linux-HA.org, and drbd (<http://www.drbd.org>) for the network mirrors. These components should be usable on most Linux distributions.

Here are the hardware planning tips we learned:

- ▶ Dedicate one pair of network cards in each node for every pair of network mirrored drives. The configuration we chose worked well for a lab environment. However, without too much extra effort or cost, its performance could have been greatly improved with the addition of another network card. The disk synchronizing occurs simultaneously for all mirrored pairs. By dedicating a network card to each pair, the available bandwidth for the synchronization is increased.
- ▶ Dedicate an entire drive spindle to a single, mirrored device. For example, on node1 you would not want to mirror `/dev/sdb1` and `/dev/sdb2` to the corresponding partitions on node2. Instead, add another drive so that you can mirror `/dev/sdb1` and `/dev/sdc1` to the corresponding partitions on node2.

Figure 3-3 on page 55 shows the base operating environment of the cluster. The shared resources denoted in boldface are the primary for that node and the grayed resources are secondary.

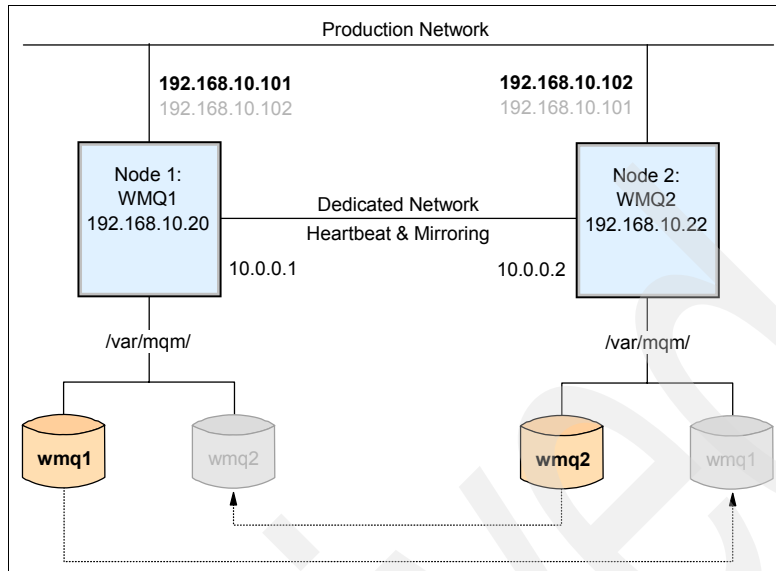


Figure 3-3 Two nodes in an HA cluster with shared resources

### 3.2.3 Perform an implementation

Install Red Hat on each machine. Make sure to select **Development Tools** and **Kernel Development** from the metapackage list. These will be needed for the installation of the drive mirroring software. The rest of the default package selections are fine.

Once the nodes are built, install the heartbeat package. The Linux-HA home page has the heartbeat source and binary packages for SUSE, but in the download section there is a link to a repository that contains binary packages for many different flavors of Red Hat and other distributions as well. Conveniently, you will also find on that same site all the dependencies in RPM package format needed to install heartbeat (except for the ones included with the Red Hat installation media). Install heartbeat with this command:

```
# rpm -Uvh heartbeat*rpm
```

There is excellent documentation for installing and configuring heartbeat and drbd from their respective Web sites and packages. The information in this redbook is intended to supplement those sources by showing key elements specific to our installation. To fully configure the basic high availability scenario, the following files need to be created or modified:

- ▶ Heartbeat configuration: /etc/ha.d/ha.cf
- ▶ Heartbeat authentication: /etc/ha.d/authkeys

- ▶ High availability resource configuration: `/etc/ha.d/haresources`
- ▶ Network mirror configuration: `/etc/drbd.conf`

For more examples of heartbeat configuration files, use this `rpm` query to see the list of documentation files included in the heartbeat package:

```
# rpm -qd heartbeat
```

### 3.2.4 Heartbeat configuration

The file shown in Example 3-1 sets the desired behavior for heartbeat and also establishes upstream hosts to ping for ipfail. This is important because we want failover to occur if one node cannot reach the network even though its heartbeat is normal. Typically you would put the IP addresses of a select group of nodes that are always supposed to be up, such as routers and switches.

*Example 3-1 /etc/ha.d/ha.cf*

---

```
# ha.cf - a configuration file for heartbeat
#
bcast eth1
keepalive 2
warntime 10
deadtime 30
initdead 120
udpport 694
auto_failback on
node   wmq1
node   wmq2

# for ipfail (detects network failures)
respawn hacluster /usr/lib/heartbeat/ipfail
ping   host1 host2 host3
```

---

### 3.2.5 Heartbeat authentication

This can be an important decision depending on your configuration. For our setup, we chose the simplest option because our heartbeat travels over an Ethernet crossover cable which is not susceptible to network intrusion. Use stronger authentication if your heartbeat travels over a corporate network. When you are done, remember to set the mode to 0600.

```
# chmod 0600 /etc/ha.d/ha.cf/authkeys
```

Example 3-2 on page 57 shows the result of this command.



*Example 3-2 /etc/ha.d/ha.cf*

---

```
auth 1
1 crc
```

---

### 3.2.6 High availability resource configuration

We need to be able to control which node is primary for which resources and in what order to start and stop those resources. For instance, we do not want to start an application server before its shared drive has been mounted. The `/etc/ha.d/haresources` file controls all of these elements.

*Example 3-3 /etc/ha.d/haresources*

---

```
wmq1 IPaddr::192.168.10.101/24
wmq2 IPaddr::192.168.10.102/24
```

---

In the first line of Example 3-3, we tell heartbeat that node `wmq1` is the primary node for IP address `192.168.10.101`. See important note below. When `wmq1` is starting or coming back from failover, it will bring up an IP alias with the specified address. When `wmq1` is stopping or failing, it will release the IP address. The second line is almost the same, except it makes node `wmq2` primary for IP address `192.168.10.102`. The resource items after the node names are names of scripts located either in `/etc/ha.d/resource.d` or `/etc/rc.d/init.d/` with arguments separated by a double colon (`::`). The heartbeat calls these scripts with the specified arguments and adds a start or stop as the final argument.

**Important:** If you do not have `auto_failback on` in the `/etc/ha.d/ha.cf` file, then the concept of primary/secondary node is lost. The resources will be owned by whichever node requested them last. This option is equivalent to `nice_failback off` in older versions of heartbeat.

This is the resource assigned to `wmq1`, a script located in `/etc/ha.d/resource.d`:

```
IPaddr::192.168.10.101/24
```

**HA resource script:** `IPaddr <ip_address> <start/stop>`

This script takes the supplied `ip_address`, determines which physical network interface has the shortest route to this address, then creates an IP alias on that interface.

**Tip:** In order to use a DNS host name with the highly available virtual IP addresses, new names will have to be added to DNS. Do not use the host names of the nodes themselves (wmq1 and wmq2, in our case), because those names do not resolve to addresses that are highly available.

Example DNS entry:

```
#ORIGIN some.net
wmq1 IN A 192.168.10.20
wmq2 IN A 192.168.10.22
wmqha1 IN A 192.168.10.101
wmqha2 IN A 192.168.10.102
```

### 3.2.7 Network mirror configuration

Because we want to have two queue managers up at the same time with each node failing over to the other, we will need shared storage. There are various technologies available for shared storage, but because our goal in this scenario was to use inexpensive, off-the-shelf components, we chose to implement network mirroring using drbd. Drbd does not allow simultaneous access to the shared disk. Simultaneous access would require more expensive hardware and a cluster file system such as IBM's GPFS. With drbd, we can mirror a local block device that works with RAID and LVM with another node over the network. The source and documentation for drbd is available from its homepage:

<http://www.drbd.org>

There are two high-level steps to installing the drbd software. The first step is to recompile the Linux kernel. The second step is to compile the drbd source.

#### Recompile the Linux kernel

The drbd drive mirroring code comes in the form of a kernel module that needs to be built using the source code of the running kernel. Therefore the kernel will have to be rebuilt so that we will have kernel source that matches the running kernel.

1. Change to the kernel source tree directory and prepare it for recompiling.

```
# cd /usr/src/linux-2.4
# make mrproper
```

2. In recent distributions of Red Hat Linux, a copy of the configuration file for the installed kernel is stored in /boot. By using it, we can rebuild the kernel with the same options as the one that came with the Red Hat installation CDs. The drbd documentation has instructions for other distributions, like SuSE.

```
# cp /boot/config-2.4.21-9.EL .config
```

3. Run one of the kernel configurators. Choices include `config`, `menuconfig`, or `xconfig`.

```
# make menuconfig
```

4. It is not necessary to make any changes, but if you know what you are doing, then now is the time to make them. When you are done, exit from the configuration tool and save your settings.
5. Build dependencies and then the kernel and the modules.

```
# make dep && make bzImage modules
```

This will take some time to complete. The default kernels come with almost every option and module turned on, so there are a lot of pieces that need to be built.

6. Install the new modules. By default modules are stored in `/lib/modules/$(uname -r)`. For Red Hat Enterprise Server that equates to `/lib/modules/2.4.21-9.EL`. The new modules will end up in `/lib/modules/2.4.21-9.ELcustom`.

```
# make modules_install
```

This command installs the new kernel and support files into `/boot` and updates the bootloader with a new boot option entitled *Red Hat Enterprise Linux AS (2.4.21-9.ELcustom)*:

```
# make install
```

7. It is usually wise to take a moment to verify that `make install` did everything as it was supposed to. When you are satisfied, reboot the system.
8. Repeat this entire procedure on the other node.

### Compile the drbd source

To compile the drbd source code, perform the following tasks:

1. Unpack the drbd tarball to a sensible location, such as `/usr/local/src`.

```
# tar xvfz drbd-0.6.12.tar.gz -C /usr/local/src
# cd /usr/local/src/drbd-0.6.12
```

2. Take a careful look at the `drbd_config.h` file for modifications that must be made in order to compile on certain systems. For Red Hat AS 3.0, we had to uncomment the second line shown in Example 3-4 on page 60. If you are using Red Hat Enterprise Linux 2.1, then make the change shown in Example 3-5 on page 60 instead.

*Example 3-4 drbd\_config.h for Red Hat Enterprise Linux 3.0*

---

```
// If your vendor kernel happens to have struct sighand (e.g. RH 2.4.20 and \
  later)
#define SIGHAND_HACK
```

---

*Example 3-5 drbd\_config for Red Hat Enterprise Linux 2.1*

---

```
// If your vendor kernel happens to have O(1) sched in place,
// but autodetection of that fact does not work (e.g. RH 2.4.9 and some other)
// simply add "|| 1" to the below line
#if defined(MAX_RT_PRIORITY) || defined(CONFIG_MAX_RT_PRIORITY) || 1
```

---

3. The rest of the build is straight forward.

```
# make all
# make install
```

4. Test the new drbd kernel module to see if it loads correctly, then remove it.

```
# modprobe drbd
# dmesg | grep drbd
# rmmod drbd
```

5. Add drbd to the list of Red Hat's managed services and set it to start automatically at the default run-levels.

```
# chkconfig --add drbd
# chkconfig drbd on
```

6. Recompile the kernel and build drbd for the second node also.

7. With drbd successfully built, the next step is to configure it. Example 3-6 is the first half of our drbd configuration file, defining one block device on each node to mirror. We used the same drbd.conf for both nodes.

*Example 3-6 Part one of /etc/drbd.conf*

---

```
resource drbd1
{
  protocol= C
  fsckcmd= /bin/true
  disk
  {
    disk-size= 2000000k
  }
  net
  {
    sync-min= 4M
    sync-max= 40M
    tl-size= 5000
    timeout= 60
    connect-int= 10
```

```

        ping-int= 10
        ko-count= 4
    }
    on wmq1
    {
        device= /dev/nb0
        disk= /dev/sdb1
        address= 10.0.0.1
        port= 7789
    }
    on wmq2
    {
        device= /dev/nb0
        disk= /dev/sdb1
        address= 10.0.0.2
        port= 7789
    }
}

```

---

This configuration establishes an HA resource called `drbd1`, which is associated with a new block device called `/dev/nb0`, which in turn is actually `/dev/sdb1` on each node. You do not have to use the same block devices on each node, but they should be the same size. Note that we have not established which node is to be primary or secondary for this resource; that comes later.

8. Recall that we want both machines to have their own queue managers running at the same time. Because `drbd` does not allow simultaneous access the shared file system, we need to create another shared disk resource for the other node to use. Example 3-7 is the second half of our `drbd.conf` file.

*Example 3-7 Part two of /etc/drbd.conf*

---

```

resource drbd2
{
    protocol= C
    fsckcmd= /bin/true
    disk
    {
        disk-size= 2000000k
    }
    net
    {
        sync-min= 4M
        sync-max= 80M
        tl-size= 5000
        timeout= 60
        connect-int= 10
        ping-int= 10
    }
}

```

```
        ko-count= 4
    }
    on wmq1
    {
        device= /dev/nb1
        disk= /dev/sdc1
        address= 10.0.0.1
        port= 7790
    }
    on wmq2
    {
        device= /dev/nb1
        disk= /dev/sdc1
        address= 10.0.0.2
        port= 7790
    }
}
```

---

Here we have defined a new resource called `drbd2` that comes from different block devices on each node. We also had to change the port number to use, as `drbd` binds to that address and port thus preventing us from reusing that address/port pair for a second resource.

Refer to the `drbd` documentation for detailed explanations of configuration options available in the `drbd.conf` file.

**Note:** Whenever a node is down or the `drbd` kernel module is unloaded, the mirrors will effectively be broken. When brought back online, `drbd` will synchronize the drives to reestablish the mirror. In our tests, we found that it took just over a minute to synchronize two mirrors with one gigabyte each over a single dedicated Fast Ethernet connection.

For this reason, we kept our shared drives relatively small so we could rapidly move from one test to another in our laboratory environment.

9. Start `drbd` on node1 first. Because the other node is not started yet, `drbd` will ask if it should assume the primary role for each disk. Answer *yes*. Look at “Monitoring the disk synchronization” on page 67 for ways to monitor `drbd`’s status.

```
node1# service drbd start
```

10. At this point, the two drives are available on node1, but there is no file system on them yet. Create them now, making sure to use journaling file system. In Linux, the choices are `ext3`, `reiserfs`, `jfs`, and `xf`s. For this example, we used `ext3`.

```
node1# mkfs.ext3 /dev/nb0
node1# mkfs.ext3 /dev/nb1
```

11. Start drbd on node2 now.

```
node2# service drbd start
```

This process will appear to hang while starting. In reality, the mirrors are being synchronized across the network. The file system that node1 put on each disk is being replicated to node2. There is no need to format the disks on node2. Refer again to “Monitoring the disk synchronization” on page 67 to watch the synchronization progress.

12. On each node, modify `/etc/fstab` by adding the lines in Example 3-8 to the end:

*Example 3-8 Lines added to `/etc/fstab`*

---

```
/dev/nb0 /var/mqm/wmq1 ext3 noauto 0 0
/dev/nb1 /var/mqm/wmq2 ext3 noauto 0 0
```

---

## Adding the shared drives to the HA resources

To add shared drives, perform the following tasks:

1. With the drive mirrors installed and configured, it is time to add them to the `haresources` file.

*Example 3-9 `/etc/ha.d/haresources` with disks added*

---

```
wmq1 IPaddr::192.168.10.101/24 datadisk::drbd1
wmq2 IPaddr::192.168.10.102/24 datadisk::drbd2
```

---

In the first line in Example 3-9, we tell heartbeat that node `wmq1` is the primary node for IP address `192.168.10.101` and shared disk `drbd1`. When `wmq1` is starting or coming back from failover, it will first grab the IP address, then the disk `drbd1`. When `wmq1` is stopping or failing, it will first stop disk `drbd1` then release the IP address. The same process holds for line two, except that node `wmq1` will not start the resources on the second line until heartbeat determines that node `wmq2` is down.

Normally, it is extremely important that this file be identical on each node. So far, we have kept true. However, we will come back to this file later on and purposefully break this rule in order to control the starting and stopping of queue managers in an active-active role.

This is the new resource added to `wmq1`, also located in `/etc/ha.d/resource.d`:

```
datadisk::drbd1
```

### **HA resource script: `datadisk::<drbd_resource><start/stop>`**

This script will not be installed on the system until the drbd package has been built and installed. The arguments are simple. It takes a drbd resource, which is defined in the drbd.conf file, as shown in Example 3-6 on page 60 and Example 3-7 on page 61.

1. So far, the drives have not been mounted yet. This is a good time to test the datadisk script.

```
node1# cd /etc/ha.d/resource.d
node1# ./datadisk drbd1 start
node1# ./datadisk drbd2 stop
```

```
node2# cd /etc/ha.d/resource.d
node2# ./datadisk drbd2 start
```

Each node should now have its primary drive and actively mirroring both pairs.

## **3.2.8 Validating heartbeat failover and failback**

This section discusses ways to validate the configuration.

### **Causing failovers and failbacks to occur**

There are multiple ways to test failover and failback. Here are a some of the ways we used in our laboratory. One thing that you should avoid is pulling the heartbeat link. This will cause both nodes to try to take over all resources.

- ▶ Power-off one of the nodes. If you can stomach it, this is a good test to perform. It lets you see how things work when the failed machine did not have a chance to do any cleanup work. If you picked journaling filesystems for all of your drives, then bringing the downed node back online should be painless. If not, then your power ups can take a long time while the system runs **fsck** against the dirty drives.
- ▶ Simulate a failure in the production network. Recall that `/etc/ha.d/ha.cf` has the following lines in Example 3-10:

*Example 3-10 ipfail snippet from `/etc/ha.d/ha.cf`*

---

```
respawn hacluster /usr/lib/heartbeat/ipfail
ping host1 host2 host3
```

---

- a. Change the bottom line so that you are pinging a linux host that you have root on, as in Example 3-11 on page 65:



*Example 3-11 Configuring the ipfail host in /etc/ha.d/ha.cf*

---

```
respawn hacluster /usr/lib/heartbeat/ipfail
ping linuxbox
```

---

- b. Login to linuxbox and run try the following command. If you have a previous iptables configuration that you want to keep, make sure you save it first.

```
linuxbox# iptables -A INPUT -s wmq1 -p icmp -j DROP
```

This will make linuxbox drop any ICMP (ping) packets coming from node wmq1. Node wmq1 will notice that its pings are not being answered and it will assume that its production network link is no longer working correctly, thus triggering a failover of all resources to the partner node.

- c. To resume ping responses and cause the failed resources to failback to their primary node, delete the iptables rule just added:

```
linuxbox# iptables -D INPUT -s wmq1 -p icmp -j DROP
```

Example 3-12 shows a simple script to help automate the simulated network failures:

*Example 3-12 droptnode.sh -- simulates network failures*

---

```
#!/bin/bash
#
# droptnode.sh for dropping icmp packets

if [ "${1}" == "" ]
then
    echo You must supply a hostname or IP address!
    exit 1
fi

iptables -A INPUT -s "${1}" -p icmp -j DROP
echo ICMP packets from "${1}" have been dropped.
echo Hit ENTER to resume ICMP traffic.
read
iptables -D INPUT -s "${1}" -p icmp -j DROP
```

---

- The last method we found useful is to call `hb_standby`, located in `/usr/lib/heartbeat`. The downside to this method is that it requires `auto_failback` to be set to `off` in `/etc/ha.d/ha.cf`. If you use this method, remember to fix your `ha.cf` when you are done testing or your resources will not failback to their primary nodes.

- a. On one node, run this command to cause failover:

```
node1# /usr/lib/heartbeat/hb_standby all
```

- b. On the other node, run this command when you are ready to failback:

```
node2# /usr/lib/heartbeat/hb_standby foreign
```

## Monitoring the HA resources

This section discusses how to set up monitoring resources.

### Monitoring heartbeat service

The heartbeat log files are an excellent source of information for your running HA cluster. The logs can be sent to syslogd if desired, but by default the logs files are here:

```
/var/log/ha-log  
/var/log/ha-debug
```

- ▶ Open up a few terminals and watch the HA messages go by:

```
# tail -f /var/log/ha-log  
# tail -f /var/log/ha-debug
```

### Monitoring network resources

There are many ways to verify the network resources are working the way they are supposed to, such as ping or an application layer test to name a few.

- ▶ One method is to use `ifconfig` to watch the IP aliases.

```
# ifconfig
```

The results are shown in Example 3-13:

#### Example 3-13 `ifconfig` output

---

```
#---some lines not shown---#  
eth0:0  Link encap:Ethernet  HWaddr 00:02:55:91:15:D7  
        inet addr:192.168.10.101  Bcast:192.168.10.255  Mask:255.255.255.0  
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
        RX packets:21526388  errors:0  dropped:0  overruns:0  frame:0  
        TX packets:39509513  errors:0  dropped:0  overruns:0  carrier:0  
        collisions:0  txqueuelen:1000  
        RX bytes:2910011309 (2775.2 Mb)  TX bytes:899150380 (857.4 Mb)  
#---some lines not shown---#
```

---

- ▶ If you are in failover mode, the available node will show something similar to Example 3-14:

#### Example 3-14 `ifconfig` output in failover mode

---

```
#---some lines not shown---#  
eth0:0  Link encap:Ethernet  HWaddr 00:02:55:91:15:D7  
        inet addr:192.168.10.101  Bcast:192.168.10.255  Mask:255.255.255.0  
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

```

RX packets:21526388 errors:0 dropped:0 overruns:0 frame:0
TX packets:39509513 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:2910011309 (2775.2 Mb) TX bytes:899150380 (857.4 Mb)

eth0:1 Link encap:Ethernet HWaddr 00:02:55:91:15:D7
inet addr:192.168.10.101 Bcast:192.168.10.255 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:4342 errors:0 dropped:0 overruns:0 frame:0
TX packets:456 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:417466 (407.6 Kb) TX bytes:45187 (44.1 Kb)

#---some lines not shown---#

```

---

### ***Monitoring the disk synchronization***

As with the network, there are many ways to monitor how the drives are behaving. The classics `mount` and `df` commands will certainly come in handy.

```
# df -h
```

The output is shown in Example 3-15.

#### *Example 3-15 Output from df -h in failover*

---

```

Filesystem      Size  Used Avail Use% Mounted on
#---some lines not shown---#
/dev/nb0        3.7G  46M  3.5G   2% /var/mqm/wmq1
/dev/nb1        3.7G   54M  3.5G   2% /var/mqm/wmq2

```

---

The `proc` filesystem also has very good information about the status of `drbd`. Here is a simple method in Example 3-16 for monitoring `drbd`'s status:

#### *Example 3-16 showdrbd.sh*

---

```

#!/bin/sh
while /bin/true
do
    cat /proc/drbd
    sleep 2
done

```

---

Example 3-17 on page 68 shows a typical output from concatenating `/proc/drbd`. This represent the normal case: both drives are connect (and therefore actively mirroring) and one drive is primary on the local node while the other is primary on the other node.

#### Example 3-17 /proc/drbd output

---

```
version: 0.6.12 (api:64/proto:62)
```

```
0: cs:Connected st:Primary/Secondary ns:40 nr:0 dw:16 dr:39 pe:0 ua:0
1: cs:Connected st:Secondary/Primary ns:484 nr:10 dw:21 dr:511 pe:0 ua:0
```

---

Example 3-18 shows the drbd system in a failover situation with the other node down. See the drbd documentation for more information about the counters and possible status codes available here.

#### Example 3-18 /proc/drbd output: failover

---

```
version: 0.6.12 (api:64/proto:62)
```

```
0: cs:WFConnection st:Primary/Unknown ns:4061 nr:0 dw:166 dr:3930 pe:0 ua:0
1: cs:WFConnection st:Primary/Unknown ns:484 nr:10 dw:23 dr:511 pe:0 ua:0
```

---

### 3.2.9 HA configuration summary

In summary, our high availability configuration looks like Table 3-1:

Table 3-1 Overview of IP address and filesystems in different situations

Node(s) Up:	IP Addresses:	Mounted Filesystems:
wmq1 and wmq2	wmq1 - 192.168.10.101 wmq2 - 192.168.10.102	wmq1 - /var/mqm/wmq1 wmq2 - /var/mqm/wmq2
wmq1 only	wmq1 - 192.168.10.101 wmq1 - 192.168.10.102	wmq1 - /var/mqm/wmq1 wmq1 - /var/mqm/wmq2
wmq2 only	wmq2 - 192.168.10.102 wmq2 - 192.168.10.101	wmq2 - /var/mqm/wmq2 wmq2 - /var/mqm/wmq1

## 3.3 Installing and configuring WebSphere MQ

This section provides an outline of a WebSphere MQ installation on Linux.

**Tip:** The complete installation guide should be followed. Download the latest available guide, *WebSphere MQ for Linux V5.3 Quick Beginnings*, from:

<http://publibfp.boulder.ibm.com/epubs/pdf/amq1ac03.pdf>

In addition, it is strongly recommended that you review the readme file included with the software delivery for late changes.

### 3.3.1 Preinstallation steps

Before installing WebSphere MQ, a few steps have to be performed.

#### Creating filesystems

For our scenario, the `/var/mqm` and `/opt/mqm` filesystems were created before the software was installed.

**Tip:** For assistance with calculating your file system size requirements, see Chapter 5: MQSeries Tuning Recommendations of the publication *SupportPac MPL1 WebSphere MQ for Linux (Intel) V5.3 - Performance Evaluations*. SupportPacs can be downloaded from:

<http://www-306.ibm.com/software/integration/support/supportpacs/>

#### Setting up user ID and group

For our scenario, the `mqm` user ID and `mqm` group were created prior to installing the software. In addition, `root` was added to the group `mqm` for this test so that all other configuration tasks could be completed by one user. Your security requirements need to be reviewed before deciding what user IDs should be added the `mqm` group.

**Tip:** Users do not need `mqm` group authority to run applications that use the queue manager; it is needed only for the administration commands. The queue manager object access is given through the Object Authority Manager (OAM). For help determining your security requirements, please review the IBM publication *WebSphere MQ Security* that can be downloaded from:

<http://www-306.ibm.com/software/integration/wmq/library/>

**Important:** An additional step is required with kernel versions 2.6, including RedHat Enterprise Edition 3, in order to successfully create a queue manager with the `crtmqm` command.

- Change to the directory where WebSphere MQ installables packages are located. And issue the following command:

```
export LD_ASSUME_KERNEL = 2.4.19
```

For a full explanation, see “New threading model introduced in kernel release 2.6” on page 5

### 3.3.2 Installation steps

Install WebSphere MQ with the *Red Hat Package Manager* (RPM):

- ▶ `rpm -i MQSeriesRuntime-5.3.0-2.rpm`
- ▶ `rpm -i MQSeriesSDK-5.3.0-2.rpm`
- ▶ `rpm -i MQSeriesServer5.3.0-2.rpm`
- ▶ `rpm -i MQSeriesClient5.3.0-2.rpm`

The above commands need to be executed for Corrective Service Disk (CSD) RPMs also. The latest CSD can be downloaded from:

<http://www-306.ibm.com/software/integration/mqfamily/support/summary/>

### 3.3.3 Postinstallation steps and install verification

After WebSphere MQ is installed, a few steps can be performed to verify the installation.

#### Setting capacity

Execute the command `setmqcap n` where *n* is the number of processors currently available on the system. This should correspond to the number of licenses that have been purchased.

#### Verifying installation

Execute the command `dspmqr` to show the version of WebSphere MQ installed along with the CSD level. For all of the tests in this book, CSD09 was the latest maintenance applied.

Login to a new terminal session as the user *mqm*.

Execute the following commands:

1. `crtmqm test`
2. `strmqm test`
3. `runmqsc test`

Within the interactive session, type and run the following commands:

- a. `def q1('q1')`
- b. `end`
4. `./opt/mqm/samp/bin/amqsput q1 test`
  - a. `Hello World!`
  - b. Press enter two times.

5. `./opt/mqm/samp/bin/amqsget q1 test`
  - a. Messages from the queue q1 will be displayed.
  - b. **Control-C** will stop the amqsget program.
6. `endmqm -i test`
7. `dltmqm test`

## 3.4 Message generation application

This section discusses a message generation application that is used in this redbook to demonstrate various aspect of WebSphere MQ in a Linux environment.

### 3.4.1 Design of the message generation application

The purpose of this application is to simulate a resource monitoring scenario that generates messages with information of available resources on a server. For this test, the application generates a random sequence of number that will simulate the usage of one of three possible resources specified as a parameter keywords of: memory, cpu and harddisk.

The program was developed using C++ to show the compatibility and libraries that should be used with Linux.

### 3.4.2 Developing the message generation application

In the application, there needs to be a reference for the queue manager and queue. See Example 3-19.

*Example 3-19 Assigning queue manager and queue references*

---

```
ImqQueueManager mgr; // Queue manager
ImqQueue queueA;    // Queue
```

---

These objects names are populated by parameters from the command line at runtime. The parameters are:

- ▶ Delay
- ▶ Keyword
- ▶ Queue name
- ▶ Queue manager name.

Once the variables are populated, the queue is associated with the queue manager. See Example 3-20 on page 72.

---

*Example 3-20 Associating a queue with a queue manager*

---

```
queueA.setConnectionReference( mgr );
queueA.setQueueManagerName( (char *)strManagerName );
```

---

Now the program can create random messages with a prefix as an identifier. See Example 3-21.

---

*Example 3-21 Generating random messages*

---

```
for(;;)
{
    time(&lt;time);
    curr=*localtime(&lt;time);
    r=-((double)rand()/(RAND_MAX+1));
    y=(unsigned long)(r*M);

    sprintf(buffer,"%s,%i-%02i-%02i-%02i.%02i.%02i,%s,%d",keyword,
            curr.tm_year+1900,curr.tm_mon,curr.tm_mday,curr.tm_hour,
            curr.tm_min,curr.tm_sec,(char*)mgr.name(),y);

    msg.useEmptyBuffer( buffer, sizeof( buffer ) );
    msg.setFormat( MQFMT_STRING ); /* character string format */
    buflen = (int)strlen( buffer ) ; /* length w/o line-end */
    buffer[ buflen ] = '\0' ;
    printf("%s\n",buffer);

    if ( buflen > 0 ) {
        msg.setMessageLength( buflen );
        if(!queueA.put(msg)){
            /* report reason, if any */
            printf( "ImqQueue::put ended with reason code %1d\n",
                    (long)queueA.reasonCode( ) );
        }
    }
    fflush(stdout);
    sleep(delay);
}
```

---

This will allow the program to create random messages and insert them in the queue, until the application is terminated.

### 3.4.3 Compiling and running the message generation application

To compile the message generation application, there are a few tasks to perform.

1. For instance, the application is developed using C++ libraries. When using GNU C++ compiler, you must make sure that any libraries containing C++



functions are built using the same version of the C++ compiler as your application.

The supported versions of the compiler are:

- 2.95
- 3.03

To check the version of your compiler, enter the command

```
g++ --version
```

2. If your chosen compiler version does not match the default version selected by WebSphere MQ, you must reconfigure the compiler to use the correct libraries by linking the C++ libraries in the WebSphere MQ library directory to those that match your compiler. To do this, issue the following commands replacing <version> with the version of your compiler.

We are using G++ version 2.96 so the commands are as in Example 3-22:

*Example 3-22 Matching the compiler versions*

---

```
ln -s -f /opt/mqm/lib/2.96/libimqb23g1.so /opt/mqm/lib/libimqb23g1.so
ln -s -f /opt/mqm/lib/2.96/libimqs23g1.so /opt/mqm/lib/libimqs23g1.so
ln -s -f /opt/mqm/lib/2.96/libimqc23g1.so /opt/mqm/lib/libimqc23g1.so
```

---

Always remember to reference the newly linked libraries from the LD\_LIBRARY\_PATH environment variable.

```
export LD_LIBRARY_PATH=/opt/mqm/lib/2.96/:$LD_LIBRARY_PATH
```

3. To compile the application enter the command below:

```
g++ -o msggen msggen.cpp
```

4. To run the application use the name of the application with parameters:

```
./msggen 5 cpu Q1 QM1
```

## 3.5 Message retrieval application

For this scenario, WebSphere Application Server 5.1 was configured to support a Web-based test application. Parts of the application code are reviewed in this section. The complete Enterprise Application Resource (EAR) including the source code is called LinuxHATestEAR.ear as is available for download. Refer to Appendix D, “Additional material” on page 333 for information about downloading the additional materials

### 3.5.1 Design of the message retrieval application

The purpose of this test application is twofold:

- ▶ First, it demonstrates how JMS connection management creates and releases connections to a queue manager in the event of a failure.
- ▶ Secondly, it also demonstrates how information can be retrieved from a database instead of a queue without impacting the presentation logic. Message retrieval from a database is discussed in 7.12, “Message retrieval application” on page 294.

**Note:** All of the Web application components were built with WebSphere Studio, but no IBM extensions were used because this is a pure J2EE application. Other J2EE-compliant application servers should be able to deploy this same application. For a discussion of installing WebSphere Studio and using it to build JMS applications, see the IBM Redbook *WebSphere Application Server V5 and WebSphere MQ Family Integration* (SG24-6878-00).

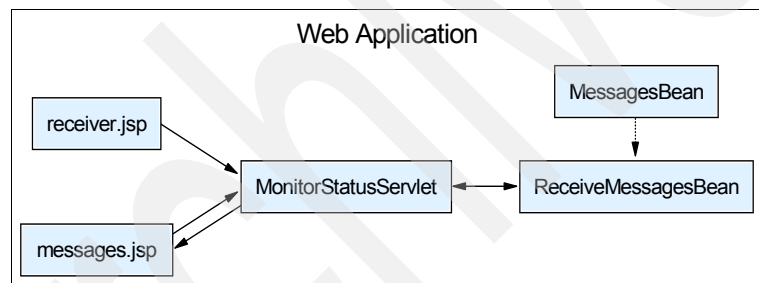


Figure 3-4 Web application topology for scenario one

### 3.5.2 Developing the message retrieval application

This section discusses some of the classes that are used to implement the message retrieval application.

#### Class `MonitorStatusServlet.java`

The class `MonitorStatusServlet.java` is responsible for accepting requests and calling a service to fulfill the request. For this scenario we will be calling the class `ReceiveMessagesBean.java` to retrieve messages. The servlet then forwards the results to the JavaServer Page `receiver.jsp` for display to the user. The same servlet is used for retrieving messages from all messaging tests because it contains no JMS or connection management code.

While the details of writing servlets are beyond the scope of this book, this is a standard servlet with only a few special methods the first being `doReceiver()`. Example 3-23 shows the relevant portion of the `doReceiver` call.

*Example 3-23 doReceiver() for calling ReceiveMessagesBean*

---

```
private ArrayList doReceiver() {
    if (messagesBean == null) {
        messagesBean = new ReceiveMessagesBean();
    }
    return (ArrayList) messagesBean.getMsgs();
}
```

---

Notice that the method only returns an `ArrayList` that will contain a list of `String` objects and knows nothing about JMS.

### **Class `ReceiveMessagesBean.java`**

The class `ReceiveMessageBean.java` is responsible for creating several objects including a JMS `QueueConnectionFactory`, `QueueConnection`, `QueueSession`, and `QueueReceiver`. It then retrieves the messages from the queue and returns the results as a `List` of `String` objects.

#### ***Initializing JMS Resources***

The `init()` method, shown in Example 3-24, is called only the first time the `getMsgs()` method is called or after an exception is caught, so a retry is attempted. This limits the number of times the JMS connection preparation is required. If no errors are encountered, the connection to the queue manager is maintained by the application with the container, the application server, for the life of the `ReceiveMessagesBean` instance.

*Example 3-24 init() for initializing the RecieveMessagesBean*

---

```
public void init() {
    try {
        jndiContext = new InitialContext();
        factory = (QueueConnectionFactory) jndiContext.lookup(QCF);
        connection = factory.createQueueConnection();
        connection.start();
        receiveQueue = (Queue) jndiContext.lookup(Q);
        initialized = true;
    } catch (JMSEException e) {
        close();
    } catch (NamingException e) {
        close();
    }
}
```

---

1. First create a context where the JNDI name of the resource will be located, in this case the default location of the application server's JNDI repository.
2. Create a queue connection from the queue connection factory using the JNDI reference of QCF (defined earlier in the code as `jms/mq/linuxHATestReceiverQCF`). At runtime this reference will point to the the WebSphere MQ queue manager.
3. From the factory, create a connection to the queue manager and start it.
4. Now that we have a running connection to the queue manager, create a JMS Queue Destination. To do this, use the same JNDI context and look for the reference called Q (defined earlier in the code as `jms/mq/linuxHATestReceiver1Q`). At runtime, this reference will point to the WebSphere MQ queue to be tested.
5. If an exception was encountered, attempt to close any opened resources through the `close()` method so that they are not orphaned.

### **Getting JMS messages**

Now that initialization is complete, the `getMsgs()` method needs only to do a few preparation steps.

First, create a new `QueueSession` from the connection. For this example, the session is created using automatic acknowledgement and is not transacted as seen in Example 3-25.

#### *Example 3-25 Creating a QueueSession from the QueueConnection*

---

```
session = connection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
```

---

From the session create a `QueueReceiver` that uses the `receiveQueue` that was previously looked-up via JNDI. Using a `QueueReceiver` ensures it is used for retrieving messages only as seen in Example 3-26.

#### *Example 3-26 Creating a QueueReceiver from the QueueSession*

---

```
queueReceiver = session.createReceiver(receiveQueue);
```

---

Now messages can be received with the `queueReceiver`'s `receive()` method. For this example, the messages are received until the queue is empty or an exception is thrown with a `do/while` loop. If a message is received and is a `TextMessage`, it is appended to a list. The relevant portions can be seen in Example 3-27.

#### *Example 3-27 Using the QueueReceiver to receive TextMessages*

---

```
do {  
    inMessage = queueReceiver.receiveNoWait();
```

```
if (inMessage instanceof TextMessage) {
    payload = ((TextMessage) inMessage).getText();
} else {
    payload = "Message was not a text message";
}
messages.add(payload);
} while (inMessage != null);
```

---

### ***Closing JMS resources***

At the end of the `getMsgs()` method, we also close the `QueueSession` and `QueueReceiver` if they were previously initialized. This ensures the entire session is isolated to the receive portion of the bean. The relevant portions of the code can be seen in Example 3-28.

#### *Example 3-28 Closing the QueueSession and QueueReceiver*

---

```
try {
    queueReceiver.close();
} catch (JMSEException e) {
    //
}
try {
    session.close();
} catch (JMSEException e) {
    //
}
```

---

When a message bean itself is no longer needed, the `close()` method should always be called because connections to the queue manager will normally not be released by the *Java Virtual Machine* (JVM) garbage collection. To keep the number of orphaned connections to the queue manager to a minimum and to return the unused connection back to container connection pooling, it is also recommended that the `close()` be attempted when a `JMSEException` is thrown.

In Example 3-29 it is done for every type of `JMSEException`. The relevant portions can be seen in Example 3-29. Your implementation might be different.

#### *Example 3-29 Closing the QueueConnection*

---

```
public void close() {
    try {
        connection.close();
    } catch (JMSEException e) {
        //
    }
}
```

---

## JavaServer Page Messages.jsp

So far the controller servlet as requested data from the message bean and returned the message payloads in the form of a list. To make this test application complete, we created a JSP Messages.jsp to present the list as a simple HTML table, allowing allows the results to be refreshed at a user defined interval. The use of *Java Server Pages* (JSP) is beyond the scope of this book, but the relevant portions of the presentation logic are presented here briefly.

The ArrayList that the servlet created is retrieved from the users session as seen in Example 3-30.

---

### Example 3-30 Retrieving the list of messages from the users session

---

```
ArrayList msg = (ArrayList) session.getAttribute("messages");
```

---

Then the ArrayList can be cycled through surrounding the entries with HTML as seen in Example 3-31.

---

### Example 3-31 Presenting the list of messages with HTML

---

```
<TABLE BORDER="1">
  <TBODY>
    <TR>
      <TD><B>Interval</B></TD>
      <TD><B>Message Payload</B></TD>
    </TR>
    <%for (int i = msg.size() - 1; i >= 0; i--) {%>
      <TR>
        <TD><B><%=i%></B></TD>
        <TD><%= (String) msg.get(i)%></TD>
      </TR>
    <%}%>
  </TBODY>
</TABLE>
```

---

A screen shot of this JSP at runtime is shown later in this redbook when the results of this test are displayed. See, for example, Figure 3-25 on page 106.

## JMS coding hints and tips

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

- ▶ The JMS specification contains some areas of loose definition. Provider implementation can, and is allowed to, vary in these areas. There are also some counter-intuitive areas in the API. These area must be treated with care. Use the JMS specification as a guide when writing JMS code and do not rely solely on a provider's documentation.

- ▶ JMS Sessions are single-threaded objects. They must not be accessed from multiple threads simultaneously. This rule extends to child objects, such as Producers and Consumers. Access to these objects must always be serialized.
- ▶ Always close JMS objects when shutting down an application. JMS objects usually contain or own resources not under the control of the JVM and therefore cannot be garbage-collected automatically. Typically, an object must be closed from within the class that created or instantiated it. This rule extends to the often overlooked JNDI context.
- ▶ The JMSEException, thrown on error, usually contains useful information from the JMS provider. This is held in the embedded LinkedException, and taking note of this data can make tracking down problems much easier.
- ▶ JMS Connections have the facility for registering an ExceptionListener. This is an application-provided piece of code that implements the onException method. Exceptions that cannot be thrown to application code will be sent here, and corrective action can be triggered. If an ExceptionListener is not registered, then the Exception information is lost. This is especially important when using asynchronous delivery.

## 3.6 WebSphere MQ HA configuration and scripts

This section discusses the configuration of MQ within an HA environment. We also discuss a number of scripts that assist in such an environment.

### 3.6.1 Normal scenario

Under normal operations, qmgr1 will be running on host wmq1, while qmgr2 will be running on host wmq2. MQ specific file systems will be mirrored as illustrated in the Figure 3-5 on page 80.

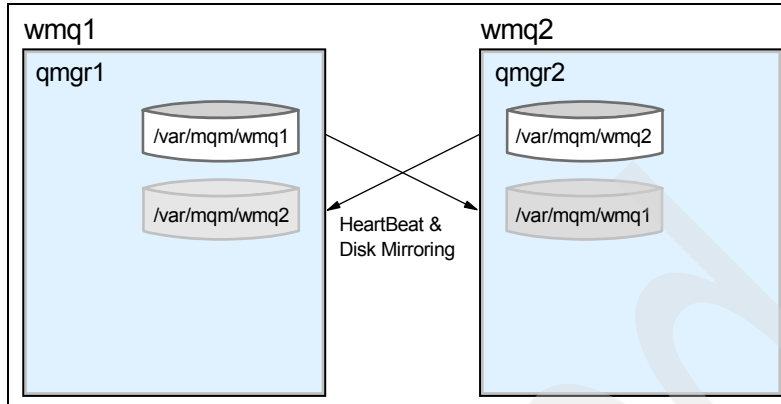


Figure 3-5 Scenario one under normal operation

### 3.6.2 Configuring wmq1 and wmq2 for WebSphere MQ HA

For WebSphere MQ HA to work in this scenario, the following configurations are required on both wmq1 and wmq2.

#### Shell scripts to manage MQ resources

To assist in the failover and fallback configuration for WebSphere MQ, a number of shell and perl scripts are used. These scripts are available as additional materials for this redbook. For information to download these scripts, refer to Appendix D, “Additional material” on page 333.

The script `mqstart` is used to start queue managers and associated components on a Linux machine. You can either name the queue manager as a parameter of the script or you can provide no parameters at all and then the script will start all defined queue managers on the system. Example 3-32 shows the usage information for this script.

#### Example 3-32 Using the `mqstart` script

---

```
USAGE: mqstart
      -or- mqstart [qmgr name]
      -or- mqstart [qmgr1 qmgr2 qmgr3...]
```

---

The script `mqstop` is similar to the script `mqstart`. It can be used to stop one or more queue managers on a system. But it can also be used to stop all queue managers defined on a system. Example 3-33 on page 81 shows the usage information for this script.



### Example 3-33 Using the mqstop script

---

```
USAGE: mqstop
      -or- mqstop [qmgr name]
      -or- mqstop [qmgr1 qmgr2 qmgr3 ...]
```

---

A third script called `mqstatus` is used to show the status of a queue manager. The status is determined by parsing the list of active processes. Alternatively, this script could have used the standard MQ command `dspmqs`.

These three scripts are used in configuration scripts that run during a failover or failback and should be copied in places accessible to root and/or mqm users, for example `/opt/mqm/bin`.

The last script is `mqsMod.pl`, a perl program that modifies the `mqs.ini` file during failover and failback to ensure that the queue managers can be started. The script `mqsMod.pl` also calls `mqstart` and `mqstop` utilities to start and stop queue managers,

The details of using `mqsMod.pl` are shown in Example 3-34. The same information can be obtained by executing `mqsMod.pl -h`. This script should be made available in a directory such as `/etc/ha.d/resource.d` directory on hosts `wmq1` and `wmq2`.

### Example 3-34 Using the mqsMod.pl perl script

---

```
Usage: mqsMod.pl PATH STATE COMMAND
       mqsMod.pl PATH STATE COMMAND [debug]
```

Where PATH should be something like `/var/mqm/node1`  
STATE must be either "primary" or "secondary"  
COMMAND must be either "start" or "stop"  
debug can be any non-whitespace characters

---

## High availability resource configuration

The file `/etc/ha.d/haresources` on both hosts `wmq1` and `wmq2` needs to be modified to include the start and stop of the queue managers.

### File haresources on wmq1

Figure 3-6 on page 82 shows the contents of the file `haresources` on host `wmq1`. In the first line in Figure 3-6, we tell heartbeat that node `wmq1` is the primary node for IP address `192.168.10.101` and shared disk `drbd1`. When `wmq1` is starting or coming back from failover, it will first grab the IP address, then disk `drbd1` and then it starts `mqsMod.pl /var/mqm/wmq1 primary start`, which starts all the queue managers present on `wmq1`.

In the second line, we tell heartbeat that node wmq2 is the primary node for IP address 192.168.10.102 and shared disk drbd2. when wmq2 fails, wmq1 takesover the IP address 192.168.10.102 ,starts drbd2 and starts **mqsMod.pl /var/mqm/wmq2 secondary start** , which modifies the mqs.ini present in wmq1 to include qmgrs present on /var/mqm/wmq2 and starts them.

```
wmq1 IPAddr::192.168.10.101/24 datadisk::drbd1 \ mqsMod.pl::/var/mqm/wmq1::primary
wmq2 IPAddr::192.168.10.102/24 datadisk::drbd2 \ mqsMod.pl::/var/mqm/wmq2::secondary
```

Figure 3-6 File /etc/ha.d/haresources on host wmq1

### File haresources on wmq2

The sequence of activities on wmq2 is the same as on mq1, except for the changes in the IPaddress, disk and the filesystem names. Figure 3-7 shows the contents of the file haresources on host wmq2.

```
wmq2 IPAddr::192.168.10.102/24 datadisk::drbd2 \ mqsMod.pl::/var/mqm/wmq2::primagry
wmq1 IPAddr::192.168.10.101/24 datadisk::drbd1 \ mqsMod.pl::/var/mqm/wmq1::Secondary
```

Figure 3-7 File /etc/ha.d/haresources on host wmq2

## 3.6.3 Failover

Either wmq1 or wmq2 can face various failures, for example hardware failures, software crashes and so forth. When one of the machines fails, the heartbeat software detects that the other machine is down and does the following. In Figure 3-8, we assume wmq2 has crashed.

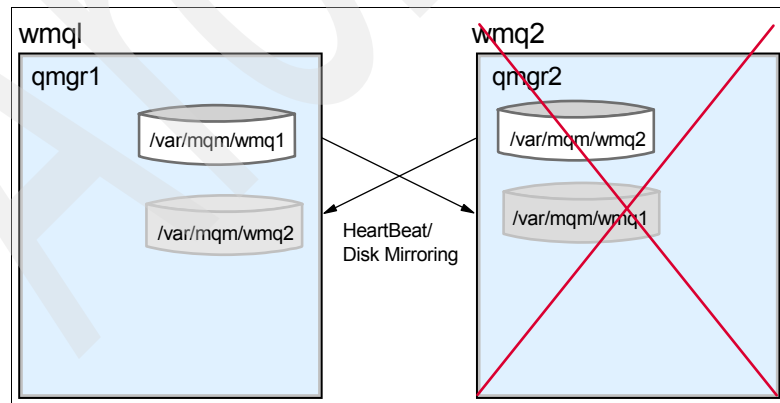


Figure 3-8 Failover

1. Host wmq1 takes over the virtual ip of wmq2.
2. Host wmq1 starts drbd2.
3. File system /var/mqm/wmq2 is mounted on wmq1.
4. Script mqsMod.pl is called with the following parameters.

```
/etc/ha.d/resource.d/mqsMod.pl /var/mqm/wmq2 secondary start
```

- a. This adds entries to /var/mqm/mqs.ini file present on wmq1 to reflect the queue managers that were present on wmq2 machine. File mqs.ini after modification is shown in Example 3-35.

*Example 3-35 File mqs.ini after updates*

---

```
AllQueueManagers:
#*****#
#* The path to the qmgrs directory, below which queue manager data  *#
#* is stored                                                         *#
#*****#
DefaultPrefix=/var/mqm/wmq1

ClientExitPath:
  ExitsDefaultPath=/var/mqm/exits

LogDefaults:
  LogPrimaryFiles=3
  LogSecondaryFiles=2
  LogFilePages=1024
  LogType=CIRCULAR
  LogBufferPages=0
  LogDefaultPath=/var/mqm/wmq1/log

QueueManager:
  Name=qmgr1
  Prefix=/var/mqm/wmq1
  Directory=qmgr1
###HA-BEGIN###

QueueManager:
  Name=qmgr2
  Prefix=/var/mqm/wmq2
  Directory=qmgr2
###HA-END###
```

---

- b. All the queue managers that were running on qmgr2 are started.

As a result, though wmq2 is down, qmgr2 is available on wmq1 as illustrated in Figure 3-9 on page 84. Any application using qmgr2 can continue its operation.

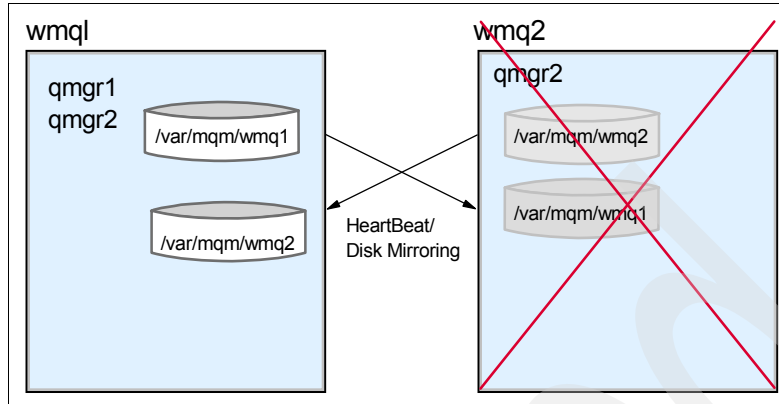


Figure 3-9 Failover

### 3.6.4 Failback

When the problem on `wmq2` is rectified, `qmgr2` stops running on `wmq1` and starts running again on `wmq2`. When `wmq2` is up, the heartbeat software detects this and does the following:

1. On host `wmq1`:
  - a. Heartbeat calls `mqsMod.pl secondary stop`
    - Stops all the queue managers present on `/var/mqm/wmq2`
    - Removes the entries of these queue managers in `/var/mqm/qmgr2` from the `mqs.ini` file of `wmq1`. The file `mqs.ini` after modification is shown in Figure 3-10 on page 85.

```
AllQueueManagers:
#*****#
#* The path to the qmgrs directory, below which queue manager data *#
#* is stored *#
#*****#
DefaultPrefix=/var/mqm/wmq1

ClientExitPath:
  ExitsDefaultPath=/var/mqm/exits

LogDefaults:
  LogPrimaryFiles=3
  LogSecondaryFiles=2
  LogFilePages=1024
  LogType=CIRCULAR
  LogBufferPages=0
  LogDefaultPath=/var/mqm/wmq1/log

QueueManager:
  Name=qmgr1
  Prefix=/var/mqm/wmq1
  Directory=qmgr1
```

Figure 3-10 File `mqs.ini` after removing entries of `qmgr2`

- b. The filesystem `/var/mqm/wmq2` is unmounted on `wmq1`
  - c. Heartbeat software stops `drbd2`
  - d. Heartbeat releases the IP address of `wmq2`
- On host `wmq2`:
- a. It takes the IP address
  - b. Heartbeat software starts `drbd2`
  - c. It calls `mqsMod.pl /var/mqm/wmq2 primary start`
    - This starts all the queue managers present on `wmq2` machine.

After executing these steps, all queue managers are running on `wmq1` and `wmq2`.

### 3.7 Persistent messages on WebSphere MQ queues

If reliable message delivery is required for applications, then using the WebSphere MQ queue-based transport and persistent messages is a good solution. Based on one of the cornerstones of the WebSphere MQ product design, message persistence involves the writing of message data to disk and, when combined with the underlying queue manager's check-pointing process,

provides complete recovery of data in the case of an unexpected program termination.

All of the tests conducted in the scenarios used in this book used persistent messages.

## 3.8 Bindings versus client tests

Messaging architectures for high availability have many options. The *Bindings* test shows how messages can be continuously received from a queue manager local to the receiving application even if one of the queue managers used by sending applications fails.

The *Client* test shows how messages can be received from a single highly-available queue manager by one or many receiving applications because they are connected remotely. This reduces the chance for orphaned messages because any receiving application can process the messages. The major drawback is that if the single high-availability queue manager fails, all the receiving applications will stop processing while the failover is done.

Enhancements can be made to both designs used in these tests. Evaluations should be conducted based on your own requirements.

## 3.9 Running the Bindings mode test

This section discusses the configuration details of WebSphere MQ when the receiving application uses Bindings mode to receive messages.

### 3.9.1 Queue Manager configuration

Figure 3-11 on page 87 illustrates the queue manager setup, where messages are sent from qmg1 and qmgr2 to qmgr3.

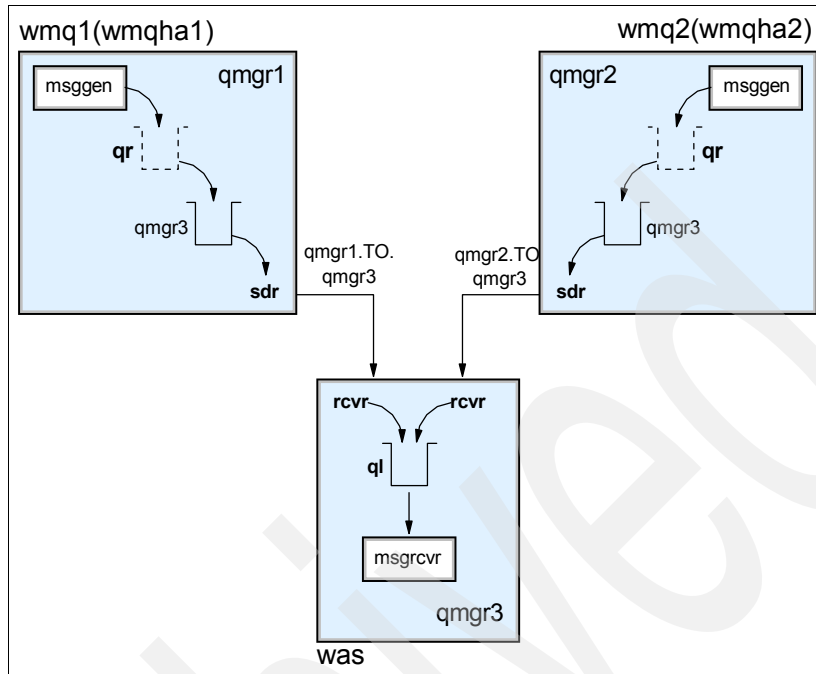


Figure 3-11 Queue Manager setup: Bindings mode

## Setup on qmgr1

To set up on `qmgr1`, perform the following tasks:

1. Ensure that `/var/mqm/wmq1` filesystem is mounted and copy the entire directory structure from `/var/mqm/` to `/var/mqm/wmq1`.

**Restriction:** Ensure that you copy the directory structure and do not perform a move.

2. Modify the contents of `mqs.ini` to reflect the new `DefaultPrefix` and `LogDefaultPath` as shown in Example 3-36.

### Example 3-36 Modified `mqs.ini`

```
AllQueueManagers:
*****#
#* The path to the qmgrs directory, below which queue manager data is stored *#
*****#
DefaultPrefix=/var/mqm/wmq1

ClientExitPath:
```

```
ExitsDefaultPath=/var/mqm/exits
```

```
LogDefaults:
```

```
LogPrimaryFiles=3  
LogSecondaryFiles=2  
LogFilePages=1024  
LogType=CIRCULAR  
LogBufferPages=0  
LogDefaultPath=/var/mqm/wmq1/log
```

---

Save and close the `mqm.ini` file.

3. Use the following commands to create and start the queue manager `qmgr1`:
  - a. **`crtmqm qmgr1`**
  - b. **`strmqm qmgr1`**
4. Now that the queue manager is created, execute the following commands using the `qmgr1.mqsc` file. The contents of this file is shown in Example 3-37.

```
runmqsc qmgr1 < qmgr1.mqsc
```

*Example 3-37 WebSphere MQ command file `qmgr1.mqsc`*

---

```
*****/  
* Define a remote queue  
*****/  
DEFINE QREMOTE('qr') +  
  RNAME('q1') +  
  RQMNAME('qmgr3') +  
  XMITQ('qmgr3') +  
  DEFPSIST(YES)  
  
*****/  
* Define the xmitq and trigger to start the channel automatically  
*****/  
DEFINE QLOCAL('qmgr3') +  
  TRIGGER +  
  INITQ(SYSTEM.CHANNEL.INITQ) +  
  TRIGTYPE(FIRST) +  
  USAGE(xmitq) +  
  MAXDEPTH(1000000) +  
  TRIGDATA('qmgr1.TO.qmgr3') +  
  DEFPSIST(YES)  
  
*****/  
* define sender channel  
*****/  
DEFINE CHANNEL('qmgr1.TO.qmgr3') +  
  CHLTYPE(SDR) +  
  TRPTYPE(TCP) +
```



```
CONNAME('was(3600)') +
XMITQ('qmgr3')
```

---

## Setup on qmgr2

To set up on qmgr2, perform the following tasks:

1. Ensure that /var/mqm/wmq2 filesystem is mounted and copy the entire directory structure from /var/mqm/ to /var/mqm/wmq2.

**Restriction:** Ensure that you copy the directory structure and do not perform a move.

2. Modify the contents of mqs.ini to reflect the new DefaultPrefix and LogDefaultPath as shown in Example 3-36.

### Example 3-38 Modified mqs.ini

---

```
AllQueueManagers:
*****#
#* The path to the qmgrs directory, below which queue manager data is stored *#
*****#
DefaultPrefix=/var/mqm/wmq2

ClientExitPath:
  ExitsDefaultPath=/var/mqm/exits

LogDefaults:
  LogPrimaryFiles=3
  LogSecondaryFiles=2
  LogFilePages=1024
  LogType=CIRCULAR
  LogBufferPages=0
  LogDefaultPath=/var/mqm/wmq2/log
```

---

Save and close the mqs.ini file.

3. Use the following commands to create and start the queue manager:
  - a. **crtmqm qmgr2**
  - b. **strmqm qmgr2**
4. Now that the queue manager is created, execute the following commands using the qmgr2.mqsc file. See Example 3-39 on page 90.

```
runmqsc qmgr2 < qmgr2.mqsc
```

*Example 3-39 WebSphere MQ command file qmgr2.mqsc*

---

```
*****/
* Define a remote queue
*****/
DEFINE QREMOTE('qr')  +
      RNAME('q1')    +
      RQMNAME('qmgr3') +
      XMITQ('qmgr3') +
      DEFPSIST(YES)

*****/
* Define the xmitq and trigger to start the channel automatically
*****/
DEFINE QLOCAL('qmgr3') +
      TRIGGER          +
      INITQ(SYSTEM.CHANNEL.INITQ) +
      TRIGTYPE(FIRST) +
      USAGE(xmitq)    +
      MAXDEPTH(1000000) +
      TRIGDATA('qmgr2.TO.qmgr3') +
      DEFPSIST(YES)

*****/
* define sender channel
*****/
DEFINE CHANNEL('qmgr2.TO.qmgr3') +
      CHLTYPE(SDR)          +
      TRPTYPE(TCP)         +
      CONNAME('was(3600)') +
      XMITQ('qmgr3')
```

---

### Setup on qmgr3

To set up on qmgr3, perform the following tasks:

1. Use the following commands to create and start the queue manager:
  - a. **crtmqm qmgr3**
  - b. **strmqm qmgr3**
2. Now that the queue manager is created, execute the following commands using the qmgr3.mqsc file. See Example 3-40 on page 91.  

```
runmqsc qmgr3 < qmgr3.mqsc
```

### Example 3-40 WebSphere MQ commands in qmgr3.mqsc

---

```
*****/
* Define a local queue
*****/
DEFINE QLOCAL('q1')

*****/
* define receiver channel
*****/
DEFINE CHANNEL('qmgr1.TO.qmgr3') +
        CHLTYPE(RCVR)      +
        TRPTYPE(TCP)

*****/
* define receiver channel
*****/
DEFINE CHANNEL('qmgr2.TO.qmgr3') +
        CHLTYPE(RCVR)      +
        TRPTYPE(TCP)
```

---

3. Execute the following command to start the queue manager listener:

```
runmq1sr -t tcp -p 3600 -m qmgr3 &
```

Note that the port number 3600 is not the standard port number for WebSphere MQ. The use of this non-standard port is reflected in the commands to create sender channels for queue managers qmgr1 and qmgr2. See Example 3-37 on page 88 and Example 3-39 on page 90.

## 3.9.2 Configuration verification

To perform an initial validation of the configuration, we can use some of the sample programs that are provided by WebSphere MQ.

1. Execute the following command on wmq1 to put messages on the remote queue:

```
./opt/mqm/samp/bin/amqspout qr qmgr1
Hello World!
```

Press the enter key two times.

2. Execute the following command on wmq1 server to ensure that the channels are in running state. The output should be similar to Example 3-41.

### Example 3-41 Channel status on qmgr1

---

```
[root@wmq1 root]# runmqsc qmgr1
5724-B41 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.
Starting MQSC for queue manager qmgr1.
```

```

dis chs(*)
  1 : dis chs(*)
AMQ8417: Display Channel Status details.
CHANNEL(qmgr1.TO.qmgr3)          XMITQ(qmgr3)
CONNAME(was(3600))              CURRENT
CHLTYPE(SDR)                   STATUS(RUNNING)
RQMNAME(qmgr3)

```

---

- Execute the following command on `wmq2` to put messages on the remote queue:

```

./opt/mqm/samp/bin/amqsput qr qmgr2
Hello World!

```

Press the enter key two times.

- Execute the following command on `wmq2` to ensure that the channels are in running state. The output should be similar to Example 3-42.

*Example 3-42 Channel status on qmgr2*

---

```

[root@wmq1 root]# runmqsc qmgr2
5724-B41 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.
Starting MQSC for queue manager qmgr2.

```

```

dis chs(*)
  1 : dis chs(*)
AMQ8417: Display Channel Status details.
CHANNEL(qmgr2.TO.qmgr3)          XMITQ(qmgr3)
CONNAME(was(3600))              CURRENT
CHLTYPE(SDR)                   STATUS(RUNNING)
RQMNAME(qmgr3)

```

---

- Execute the following command on `host was` to retrieve the messages from the local queue:

```

./opt/mqm/samp/bin/amqsget q1 test

```

- Two messages from the queue `q1` should be displayed.
- Control-C** will stop the `amqsget` program.
- Execute the following command on the `was` server to ensure that the channels are in running state. The output should be similar to Example 3-43.

*Example 3-43 Channel status on qmgr3*

---

```

[root@was errors]# runmqsc qmgr3
5724-B41 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.

```

Starting MQSC for queue manager qmgr3.

```
dis chs(*)
  1 : dis chs(*)
AMQ8417: Display Channel Status details.
  CHANNEL(qmgr2.TO.qmgr3)          XMITQ( )
  CONNAME(192.168.10.22)           CURRENT
  CHLTYPE(RCVR)                    STATUS(RUNNING)
  RQMNAME(qmgr2)
AMQ8417: Display Channel Status details.
  CHANNEL(qmgr1.TO.qmgr3)          XMITQ( )
  CONNAME(192.168.10.20)           CURRENT
  CHLTYPE(RCVR)                    STATUS(RUNNING)
  RQMNAME(qmgr1)
```

---

### 3.9.3 WebSphere Application Server configuration

In 3.5, “Message retrieval application” on page 73, we reviewed the `LinuxHATestEAR.ear` application. For the application to work at runtime, several WebSphere Application Server resources need to be defined.

**Attention:** The WebSphere Application Server installation and application server instance creation is beyond the scope of this book. However, there is a multitude of references to assist with these tasks. For a complete description of installing WebSphere Application Server and configuring it to interoperate with messaging applications, see the IBM Redbook *WebSphere Application Server V5 and WebSphere MQ Family Integration* (SG24-6878-00).

#### Reviewing the application server instance

For this test, a single application server called `server1` was created and started. From the *WebSphere Application Server Administrative Console* (shown in Figure 3-12 on page 94), notice that there is only one entry for Application Servers created.

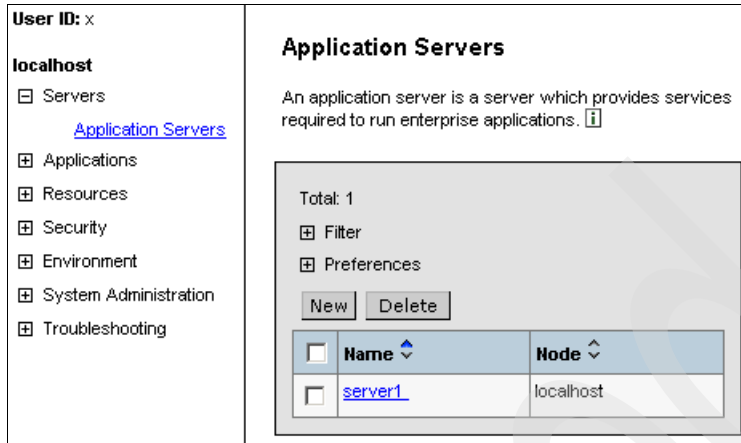


Figure 3-12 Application Server: server1

Because an application server has been created, the LinuxHATestEAR can be deployed. From **Install New Applications**, install the EAR and accept all the default deployment settings. Once the application is deployed and the new configuration is saved, you can see the application from the Enterprise Applications panel, shown in Figure 3-13.

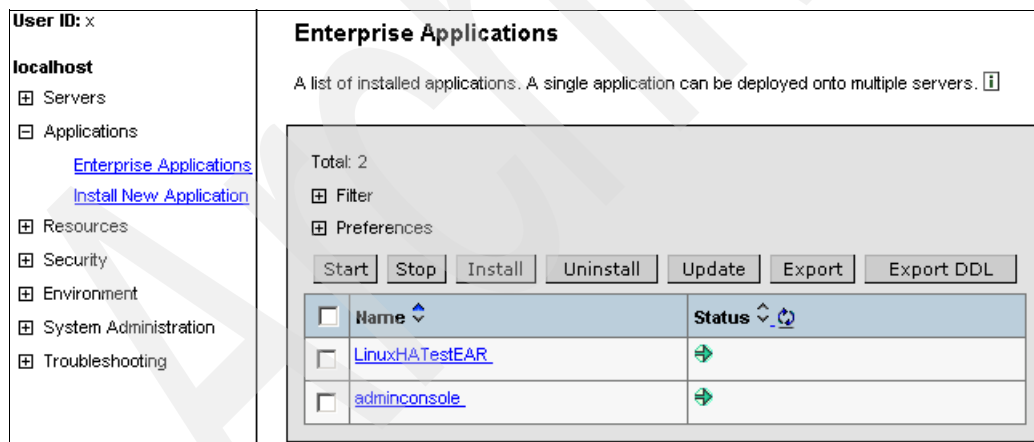


Figure 3-13 Enterprise Application: LinuxHATestEAR

For this test, the only applications that were left installed are the adminconsole and the test application LinuxHATestEAR.

**Tip:** For testing purposes, it is much easier to view the tracing, logging and configuration if there are only these two applications installed. For a real deployment, this application server could contain many more applications.

Now that the application is deployed, the *WebSphere MQ JMS Provider* needs to be configured. The default scope for the JMS Provider is set to **Node**, shown in Figure 3-14. For this test we set the scope for the provider settings to be at the Server level. Select **Server** and click **Apply**. For this test, the choice of scope was an arbitrary one because there is only one application server and application. For your environment the requirements might vary.

**User ID:** x

**localhost**

- Servers
- Applications
- Resources
  - [JDBC Providers](#)
  - [Generic JMS Providers](#)
  - [WebSphere JMS Provider](#)
  - [WebSphere MQ JMS Provider](#)
  - [Mail Providers](#)
  - [Resource Environment Providers](#)
  - [URL Providers](#)
  - [Resource Adapters](#)
- Security
- Environment
- System Administration
- Troubleshooting

### WebSphere MQ JMS Provider

A JMS provider enables asynchronous messaging based on the Java Messaging Service (JMS). It provides J2EE connection factories to create connections for specific JMS queue or topic destinations. WebSphere MQ JMS provider administrative objects are used to manage JMS resources for WebSphere MQ as the JMS provider. [i](#)

#### Configuration

Scope: Cell=localhost, Node=localhost

<input type="radio"/>	Cell	localhost	Use scope settings to limit the availability of resources to a particular cell, node, or server.
<input checked="" type="radio"/>	Node	localhost	When new items are created in this view, they will be created within the current scope.
<input type="radio"/>	Server	server1	

#### General Properties

Scope	cells:localhost:nodes:localhost
Name	WebSphere MQ JMS Provider

Figure 3-14 WebSphere MQ JMS Provider - Setting Scope to Node

Now that the scope is set, scroll to the bottom of the panel and select **WebSphere MQ Queue Connection Factories**. Create a new instance and populate the fields as shown in Figure 3-15 on page 96.

[WebSphere MQ JMS Provider](#) > [WebSphere MQ Queue Connection Factories](#) >  
**linuxHATestReceiverQCF**

A queue connection factory is used to create connections to the associated JMS provider of JMS queue destinations, for point-to-point messaging. Use WebSphere MQ Queue Connection Factory administrative objects to manage queue connection factories for the WebSphere MQ JMS provider. [i](#)

**Configuration**

General Properties	
Scope	* cells:localhost:nodes:localhost
Name	* linuxHATestReceiverQCF
JNDI Name	* jms/mq/linuxHATestReceiverQCF
Description	A QCF to be used by receiver applications (non-listener based) to test Linux HA.
Category	RECEIVING_ONLY
Component-managed Authentication Alias	(none)
Container-managed Authentication Alias	(none)
Mapping-Configuration Alias	DefaultPrincipalMapping
Queue Manager	qmgr3
Host	
Port	0
Channel	
Transport Type	BINDINGS
Model Queue Definition	

Figure 3-15 Defining linuxHATestReceiverQCF in Bindings mode

Apply the changes and save the configuration. Now that the queue connection factory linuxHATestReceiverQCF has been defined, the list of WebSphere MQ Queue Connection Factories should display the new configuration, shown in Figure 3-16 on page 97.



[WebSphere MQ JMS Provider](#) >

### WebSphere MQ Queue Connection Factories

A queue connection factory is used to create connections to the associated JMS provider of JMS queue destinations, for point-to-point messaging. Use WebSphere MQ Queue Connection Factory administrative objects to manage queue connection factories for the WebSphere MQ JMS provider. [i](#)

Total: 1

Filter

Preferences

<input type="checkbox"/>	Name ▾	JNDI Name ▾	Description ▾	Category ▾
<input type="checkbox"/>	<a href="#">linuxHATestReceiverQCF</a>	jms/mq/linuxHATestReceiverQCF	A QCF to be used by receiver applications (non-listener based) to test Linux HA.	RECEIVING_ONLY

Figure 3-16 Completed linuxHATestReceiverQCF

Return to the WebSphere MQ JMS Provider panel and select **WebSphere MQ Queue Destinations**. Create a new instance and populate the fields as shown in Figure 3-17 on page 98.

[WebSphere MQ JMS Provider](#) > [WebSphere MQ Queue Destinations](#) >

### linuxHATestReceiver1Q

Queue destinations provided for point-to-point messaging by the WebSphere MQ JMS provider. Use WebSphere MQ Queue Destination administrative objects to manage queue destinations for the WebSphere MQ JMS provider.

**Configuration**

General Properties		
Scope	* cells:localhost:nodes:localhost	The scope of the configured resource. This value indicates the configuration location for the configuration file.
Name	* linuxHATestReceiver1Q	The required display name for the resource.
JNDI Name	* jms/mq/linuxHATestReceiver1Q	The JNDI name for the resource.
Description	A queue for receiving messages during the Linux HA test (no listener, receiver only).	An optional description for the resource.
Category	RECEIVER_ONLY	An optional category string which can be used to classify or group the resource.
Persistence	APPLICATION DEFINED	Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application.
Priority	APPLICATION DEFINED	Whether the message priority for this destination is defined by the application or the Specified priority property.
Specified Priority	0	If the Priority property is set to Specified, type here the message priority for this queue, in the range 0 through 9.
Expiry	APPLICATION DEFINED	Whether the expiry timeout for this queue is defined by the application or the Specified expiry property, or messages on the queue never expire (have an unlimited expiry timeout).
Specified Expiry	0 milliseconds	If the Expiry timeout property is set to Specified, type here the number of milliseconds (greater than 0) after which messages on this queue expire.
Base Queue Name	* q1	The name of the queue to which messages are sent, as the queue manager specified by

Figure 3-17 Defining linuxHATestReceiver1Q

Now that the queue destination linuxHATestReceiver1Q has been defined, the list of WebSphere MQ Queue Destinations should show the new configuration (shown in Figure 3-18 on page 99).

[WebSphere MQ JMS Provider >](#)

### WebSphere MQ Queue Destinations

Queue destinations provided for point-to-point messaging by the WebSphere MQ JMS provider. Use WebSphere MQ Queue Destination administrative objects to manage queue destinations for the WebSphere MQ JMS provider. [i](#)

Total: 1

Filter

Preferences

<input type="checkbox"/>	Name <input type="text"/>	JNDI Name <input type="text"/>	Description <input type="text"/>	Category <input type="text"/>
<input type="checkbox"/>	<a href="#">linuxHATestReceiver1Q</a>	jms/mq/linuxHATestReceiver1Q	A queue for receiving messages during the Linux HA test (no listener, receiver only).	RECEIVER_ONLY

Figure 3-18 Completed linuxHATestReceiver1Q

The linuxHATestReceiverQCF and linuxHATestReceiver1Q are the only two references required for this first scenario. The next time you start the application server, the references to these connections should be found automatically.

### 3.9.4 Queue Manager in action

In this section, the application is started under normal operation and then the failover test is executed.

#### Normal operation

To begin normal operation, perform the following tasks:

1. Execute **dspm** on host wmq1 to ensure qmgr1 is running. See Example 3-44.

*Example 3-44 dspm on wmq1*

---

```
[root@wmq1 mqm]# dspm
QMNAME(qmgr1) STATUS(Running)
```

---

2. Execute **dspm** on host wmq2 to ensure qmgr2 is running, as in Example 3-45 on page 100.

*Example 3-45 dspmq on wmq2*

```
[root@wmq2 mqm]# dspmq
QMNAME(qmgr2) STATUS(Running)
```

3. Execute **dspmq** on host was to ensure qmgr3 is running. See Example 3-46.

*Example 3-46 dspmq on was*

```
[root@was mqm]# dspmq
QMNAME(qmgr3) STATUS(Running)
```

4. Restart server1 and ensure the two JMS resources previously defined are discovered through JNDI. In server1's SystemOut.1log there should be entries similar to Example 3-47.

*Example 3-47 Checking for JMS resource storage*

```
WSVR0049I: Binding linuxHATestReceiverQCF as jms/mq/linuxHATestReceiverQCF
WSVR0049I: Binding linuxHATestReceiver1Q as jms/mq/linuxHATestReceiver1Q
```

At the end of the SystemOut.1log there should also be entries that ensure the deployed application and the application server are started similar to Example 3-48.

*Example 3-48 Checking for application and server startup*

```
WSVR0221I: Application started: LinuxHATestEAR
ADMU3000I: Server server1 open for e-business
```

5. If you used the default settings for the application installation, open a Web browser and point it to:

`http://was:9080/LinuxHATest/index.jsp`

The screen will look similar to Figure 3-19.



*Figure 3-19 Browser view of index.jsp*

6. Select the **Receiver Test**. A new JSP will be shown, as in Figure 3-20 on page 101.



Figure 3-20 Browser view of receiver.jsp

- For this test, the refresh interval was set to one second to witness the closest failover time interval possible. Click **Start Monitor** to initiate the receiver.

The receiver starts, but because there are no messages available on the queue, the table displays No messages received.

### Problem determination

If there is an error accessing the queue manager, the top of the error will be displayed, for example Error receiving messages: MQJMS2008: failed to open MQ queue. If an error is encountered, check the server1's SystemOut.log and SystemErr.log to get the problem details.

If the cause of the problem cannot be determined from the log files alone, you can easily turned on high-level tracing for server1. This tracing should show the root cause of the problem if it is a connectivity or JMS API to queue manager issue. To start the trace, open the WebSphere Application Server Administrative Console

- Select **Logs and Traces** from the **Troubleshooting** menu.

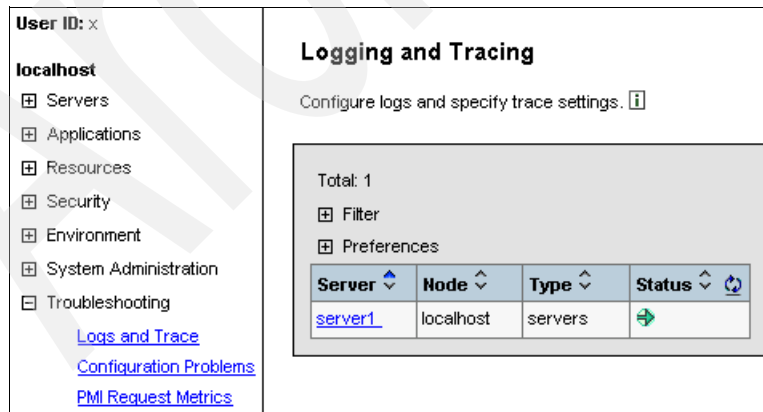


Figure 3-21 Troubleshooting: Logging and Tracing

2. Select **server1** and then **Diagnostic Trace**. See Figure 3-22.

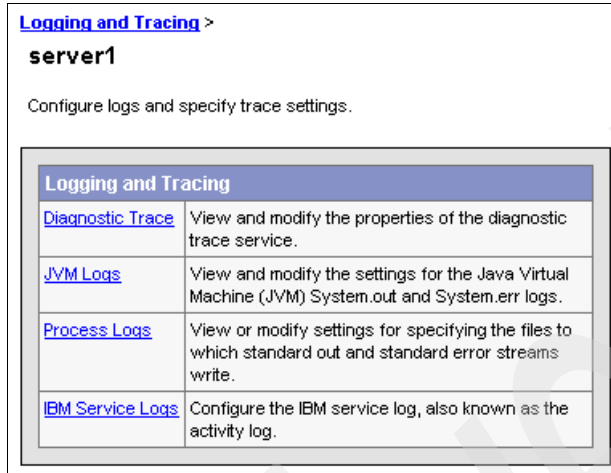


Figure 3-22 Logging and Tracing; Diagnostic Trace

3. Select the **Runtime** tab and fill the Trace Specification field as shown in Figure 3-23 on page 103. For the trace to be started every time server1 starts, also select the **Save Trace** checkbox.

[Logging and Tracing](#) > [server1](#) >  
**Diagnostic Trace Service**

Use this page to view and modify the properties of the diagnostic trace service. [i](#)

**Configuration** | **Runtime**

**General Properties**

Save Trace  Save runtime changes to configuration as well

Trace Specification  
 JMSApi=all-enabled: Messaging=all-enabled

Trace Output

Memory Buffer  
 Maximum Buffer Size  thousand entries  
 Dump File Name

File  
 Maximum File Size  MB  
 Maximum Number of Historical Files   
 File Name

Figure 3-23 Diagnostic Trace Service: Runtime General Properties

4. Apply and save the changes. From the server1 logs directory, you should be able to see a trace file already created.

This trace file should show every JMS based interaction with server1 and the queue manager with which it is attempting to connect. The details of the problem should be found here, as in Example 3-49.

*Example 3-49 Example of JMS API trace output*

```
5b627db0 JMS_WASTraceA d com.ibm.mq.MQInternalCommunications Waiting for data
on input stream.
6fda3db0 JMS_WASTraceA d com.ibm.mq.MQInternalCommunications
java.net.SocketTimeoutException: Read timed out
```

```
6fda3db0 JMS_WASTraceA > com.ibm.mqservices.MQInternalException MQException
constructor(cc, rc, source, msgid) Entry
6fda3db0 JMS_WASTraceA d com.ibm.mqservices.MQInternalException
javabase/com/ibm/mq/MQException.java, java, j5306, j5306-L031211 03/12/11
10:35:52 @(#) 1.50.1.1
6fda3db0 JMS_WASTraceA d com.ibm.mqservices.MQInternalException cc = 2
6fda3db0 JMS_WASTraceA d com.ibm.mqservices.MQInternalException rc = 2195
6fda3db0 JMS_WASTraceA d com.ibm.mqservices.MQInternalException source = null
6fda3db0 JMS_WASTraceA d com.ibm.mqservices.MQInternalException msgId = 48
6fda3db0 JMS_WASTraceA d com.ibm.mqservices.MQInternalException Explanation is
'MQJE001: An MQException occurred: Completion Code 2, Reason 2195
```

---

5. Once the problem is resolved, complete the steps in this section 3.9.4, “Queue Manager in action” on page 99 again.
6. Continue with running the application.
  - a. Run the application on host wmq1.

Change to the directory where the message generating program is present and start the application to put messages on the remote queue definition qr on qmgr1 using the following command.

```
./msggen 1 cpu qr qmgr1
```
  - b. Run the application on host wmq2.

Change to the directory where the message generating program is present and start the application to put messages on the remote queue definition qr on qmgr2 using the following command.

```
./msggen 1 mem qr qmgr2
```
  - c. Run the application on host was.

The messages from qmgr1 and qmgr2 will be sent to the queue q1 on qmgr3. The message receiver program should immediately retrieve the messages and display the messages as illustrated in Figure 3-24 on page 105.



## Monitoring Resources

Message Number	Message Payload
13	cpu,2004-03-28-23.47.04,qmgr1,18
12	mem,2004-03-28-23.47.04,qmgr2,0
11	cpu,2004-03-28-23.47.03,qmgr1,21
10	mem,2004-03-28-23.47.03,qmgr2,0
9	cpu,2004-03-28-23.47.02,qmgr1,76
8	mem,2004-03-28-23.47.02,qmgr2,0
7	cpu,2004-03-28-23.47.01,qmgr1,55
6	mem,2004-03-28-23.47.01,qmgr2,0
5	cpu,2004-03-28-23.47.00,qmgr1,45
4	mem,2004-03-28-23.46.59,qmgr2,0
3	cpu,2004-03-28-23.46.59,qmgr1,23
2	cpu,2004-03-28-23.46.58,qmgr1,27
1	No messages received
0	No messages received

Figure 3-24 Displaying results with messages.jsp before failover

### Failover scenario

Now that we are successfully sending and receiving messages, it is time to test the failover.

1. Power off the `wmq2` server.

The message receiving application on the was server will stop receiving messages from `wmq2` temporarily while the queue manager is recovered on `wmq1`. Heartbeat software on `wmq1` will detect that `wmq2` is not responding. To view the tasks performed on `wmq1` see the log file at `/var/mqm/ha-log`. The output from the `ha-log` should be similar to Example 3-50.

#### Example 3-50 `ha-log` during failover

```
heartbeat: 2004/04/28_23:47:14 WARN: node wmq2rt: is dead
heartbeat: 2004/04/28_23:47:14 info: Dead node wmq2 held no resources.
heartbeat: 2004/04/28_23:47:14 ERROR: No one owns foreign resources!
heartbeat: 2004/04/28_23:47:14 info: Resources being acquired from wmq2.
heartbeat: 2004/04/28_23:47:14 info: Link wmq2:eth1 dead.
heartbeat: 2004/04/28_23:47:14 info: Running /etc/ha.d/rc.d/status status
```

```

heartbeat: 2004/04/28_23:47:14 info: Taking over resource group IPAddr::192.168.10.102/24
heartbeat: 2004/04/28_23:47:14 info: Acquiring resource group: wmq2 IPAddr::192.168.10.102/24
datadisk::drbd2 mqsMod.pl::/var/mqm/wmq2::secondary
heartbeat: 2004/04/28_23:47:14 info: Local Resource acquisition completed.
heartbeat: 2004/04/28_23:47:14 info: Running /etc/ha.d/resource.d/IPAddr 192.168.10.102/24
start
heartbeat: 2004/04/28_23:47:14 info: /sbin/ifconfig eth0:1 192.168.10.102 netmask 255.255.255.0
broadcast 192.168.10.255
heartbeat: 2004/04/28_23:47:14 info: Sending Gratuitous Arp for 192.168.10.102 on eth0:1 [eth0]
heartbeat: 2004/04/28_23:47:14 /usr/lib/heartbeat/send_arp -i 500 -r 10 -p
/var/lib/heartbeat/rsctmp/send_arp/send_arp-192.168.10.102 eth0 192.168.10.102 auto
192.168.10.102 ffffffff
heartbeat: 2004/04/28_23:47:15 info: Running /etc/ha.d/resource.d/datadisk drbd2 start
heartbeat: 2004/04/28_23:47:21 info: Running /etc/ha.d/resource.d/mqsMod.pl /var/mqm/wmq2
secondary start
heartbeat: 2004/04/28_23:47:26 info: /usr/lib/heartbeat/mach_down: nice_failback: foreign
resources acquired
heartbeat: 2004/04/28_23:47:26 info: mach_down takeover complete.
heartbeat: 2004/04/28_23:47:26 info: mach_down takeover complete for node wmq2.

```

The output from messages.jsp should show something similar to Figure 3-25.

Monitoring Resources	
Message Number	Message Payload
23	cpu,2004-03-28-23.47.15,qmgr1,43
22	cpu,2004-03-28-23.47.05,qmgr1,71
21	No messages received
20	No messages received
19	No messages received
18	No messages received
17	No messages received
16	No messages received
15	No messages received
14	No messages received
13	cpu,2004-03-28-23.47.04,qmgr1,18
12	mem,2004-03-28-23.47.04,qmgr2,0
11	cpu,2004-03-28-23.47.03,qmgr1,21

Figure 3-25 Displaying results of messages.jsp during failover

**Note:** Notice that temporarily no messages are received from qmgr1 or qmgr2. This is because the message generating application using qmgr1 is being blocked for IO while the disk-mirroring detects that wmq2 is unavailable. Scenario two will demonstrate this is only a problem when using disk-mirroring.

The actual time it takes for wmq1 to notice wmq2 is offline showed less than ten seconds during these tests. The blocked MQPut call from WebSphere MQ will wait for up to 10 seconds for the message to be committed to disk. Therefore, it should appear to the message generating application that there is only a slow response on the first put after the failover. There is no data loss or an application aware failure. If it does take longer than ten seconds, the message generating application would have to retry the mqput operation.

2. Now execute **dspm** on wmq1. It should show that qmgr1 and qmgr2 are both running as shown in Example 3-51.

*Example 3-51 dspm on wmq1 after failover*

---

```
[root@wmq1 c++]# dspm
QMNAME(qmgr1)                STATUS(Running)
QMNAME(qmgr2)                STATUS(Running)
```

---

3. Start the message generating programs to put messages on the remote queue definition q1 on qmgr2 again by executing the command **msggen 1 mem qr qmgr2**. The program instance using qmgr1 should still be running.

Now that the second message generating program is started, the results on the Web page messages.jsp should show messages arriving from both queue managers.

**Note:** If there had been messages queued on qmgr2 while the failure took place, they would have immediately flowed to qmgr3 when the failover completed because the message were persistent. Failover to qmgr3 is not illustrated in this scenario, but you can test it by generating large numbers of messages on qmgr2 prior to powering off. In addition, you could also test it by stopping the qmgr2.TO.qmgr3 sender channel, allowing messages to wait on the transmission queue.

## Monitoring Resources

Message Number	Message Payload
39	cpu,2004-03-28-23.47.28,qmgr1,88
38	mem,2004-03-28-23.47.27,qmgr2,0
37	cpu,2004-03-28-23.47.27,qmgr1,26
36	mem,2004-03-28-23.47.26,qmgr2,0
35	cpu,2004-03-28-23.47.26,qmgr1,77
34	mem,2004-03-28-23.47.25,qmgr2,0
33	cpu,2004-03-28-23.47.25,qmgr1,37
32	mem,2004-03-28-23.47.24,qmgr2,0
31	cpu,2004-03-28-23.47.24,qmgr1,22
30	cpu,2004-03-28-23.47.23,qmgr1,96
29	cpu,2004-03-28-23.47.21,qmgr1,68
28	cpu,2004-03-28-23.47.20,qmgr1,27
27	cpu,2004-03-28-23.47.19,qmgr1,12
26	cpu,2004-03-28-23.47.18,qmgr1,4
25	cpu,2004-03-28-23.47.17,qmgr1,15
24	cpu,2004-03-28-23.47.16,qmgr1,21
23	cpu,2004-03-28-23.47.15,qmgr1,43

Figure 3-26 Messages.jsp results after both programs are started

### Failback scenario

Now that both qmgr1 and qmgr2 are running on wmq1, it is time to test the failback.

**Note:** *Failback* in this context means moving queue manager qmgr2 back to wmq2.

1. Power on the wmq2 server.

During wmq2 server startup, the drbd service will synchronize the contents of the mirrored drive present on wmq1 to the drive present on wmq2 . The status of the synchronization can be viewed by executing the command `cat /proc/drbd`. The output should be similar to Example 3-52 on page 109.

### Example 3-52 drbd output during failback

---

```
0: cs:SyncingQuick st:Primary/Secondary ns:27096 nr:3911796 dw:3953368 dr:1821 pe:0 ua:0
1: cs:SyncingAll st:Primary/Secondary ns:3918220 nr:3935332 dw:3969864 dr:3919582 pe:74 ua:0
    [>.....] sync'ed: 0.1% (3819/3820)M
    finish: 2:43:58h speed: 356 (356) K/sec
version: 0.6.12 (api:64/proto:62)

0: cs:Connected st:Primary/Secondary ns:31160 nr:3911796 dw:3953368 dr:5885 pe:0 ua:0
1: cs:SyncingAll st:Primary/Secondary ns:7826140 nr:3935332 dw:3969864 dr:7827498 pe:14 ua:0
    [=====>] sync'ed:100.0% (3/3820)M
    finish: 0:00:00h speed: 5,414 (5,231) K/sec
version: 0.6.12 (api:64/proto:62)
```

---

**Important:** The time taken by the drbd service to synchronize the drives is proportional to the size of the hard disk mirrored. In our setup, to synchronize a 2GB drive the synchronization took five minutes. For a 4GB drive, it took 11 minutes. This also implies that the wmq2 server took five minutes to complete the restart process.

When wmq2 server has completed the restart, the heartbeat software on wmq2 should immediately takeover qmgr2. To view the tasks performed on wmq1 see the log file at `/var/mqm/ha-log on wmq1`. The file contents should be similar to Example 3-53.

### Example 3-53 ha-log on wmq1 on failback

---

```
heartbeat: 2004/04/29_12:28:48 info: Heartbeat restart on node wmq2
heartbeat: 2004/04/29_12:28:48 info: Link wmq2:eth1 up.
heartbeat: 2004/04/29_12:28:48 info: Status update for node wmq2: status up
heartbeat: 2004/04/29_12:28:48 info: Running /etc/ha.d/rc.d/status status
heartbeat: 2004/04/29_12:28:48 info: Status update for node wmq2: status active
heartbeat: 2004/04/29_12:28:48 info: remote resource transition completed.
heartbeat: 2004/04/29_12:28:48 info: wmq1 wants to go standby [foreign]
heartbeat: 2004/04/29_12:28:48 info: standby: wmq2 can take our foreign resources
heartbeat: 2004/04/29_12:28:48 info: give up foreign HA resources (standby).
heartbeat: 2004/04/29_12:28:48 info: Releasing resource group: wmq2 IPAddr::192.168.10.102/24
datadisk::drbd2 mqsMod.pl::/var/mqm/wmq2::secondary
heartbeat: 2004/04/29_12:28:48 info: Running /etc/ha.d/resource.d/mqsMod.pl /var/mqm/wmq2
secondary stop
heartbeat: 2004/04/29_12:28:53 info: Running /etc/ha.d/resource.d/datadisk drbd2 stop
heartbeat: 2004/04/29_12:28:54 info: Running /etc/ha.d/resource.d/IPAddr 192.168.10.102/24 stop
heartbeat: 2004/04/29_12:28:54 info: /sbin/route -n del -host 192.168.10.102
heartbeat: 2004/04/29_12:28:54 info: /sbin/ifconfig eth0:1 down
heartbeat: 2004/04/29_12:28:54 info: IP Address 192.168.10.102 released
heartbeat: 2004/04/29_12:28:54 info: foreign HA resource release completed (standby).
```

---

To view the tasks performed on `wmq2` see the log file at `/var/mqm/ha-log` on `wmq2`. The content of the file should be similar to Example 3-54.

*Example 3-54 ha-log on wmq2 on failback*

---

```
heartbeat: 2004/04/29_12:28:46 info: Configuration validated. Starting heartbeat 1.2.0
heartbeat: 2004/04/29_12:28:46 info: heartbeat: version 1.2.0
heartbeat: 2004/04/29_12:28:47 info: Heartbeat generation: 23
heartbeat: 2004/04/29_12:28:47 info: UDP Broadcast heartbeat started on port 694 (694)
interface eth1
heartbeat: 2004/04/29_12:28:47 info: pid 2524 locked in memory.
heartbeat: 2004/04/29_12:28:47 info: Local status now set to: 'up'
heartbeat: 2004/04/29_12:28:48 info: pid 2537 locked in memory.
heartbeat: 2004/04/29_12:28:48 info: pid 2538 locked in memory.
heartbeat: 2004/04/29_12:28:48 info: pid 2539 locked in memory.
heartbeat: 2004/04/29_12:28:48 info: Link wmq1:eth1 up.
heartbeat: 2004/04/29_12:28:48 info: pid 2540 locked in memory.
heartbeat: 2004/04/29_12:28:48 info: pid 2541 locked in memory.
heartbeat: 2004/04/29_12:28:48 info: Status update for node wmq1: status active
heartbeat: 2004/04/29_12:28:48 info: Link was:was up.
heartbeat: 2004/04/29_12:28:48 info: Status update for node was: status ping
heartbeat: 2004/04/29_12:28:48 info: Local status now set to: 'active'
heartbeat: 2004/04/29_12:28:48 info: Starting child client "/usr/lib/heartbeat/ipfail" (511,90)
heartbeat: 2004/04/29_12:28:48 info: Starting "/usr/lib/heartbeat/ipfail" as uid 511 gid 90
(pid 2578)
heartbeat: 2004/04/29_12:28:48 info: Link wmq2:eth1 up.
heartbeat: 2004/04/29_12:28:48 info: remote resource transition completed.
heartbeat: 2004/04/29_12:28:48 info: remote resource transition completed.
heartbeat: 2004/04/29_12:28:48 info: Local Resource acquisition completed. (none)
heartbeat: 2004/04/29_12:28:48 info: wmq1 wants to go standby [foreign]
heartbeat: 2004/04/29_12:28:48 info: Running /etc/ha.d/rc.d/status status
heartbeat: 2004/04/29_12:28:55 info: standby: acquire [foreign] resources from wmq1
heartbeat: 2004/04/29_12:28:55 info: acquire local HA resources (standby).
heartbeat: 2004/04/29_12:28:55 info: Acquiring resource group: wmq2 IPAddr::192.168.10.102/24
datadisk::drbd2 mqsMod.pl::/var/mqm/wmq2::primary
heartbeat: 2004/04/29_12:28:55 info: Running /etc/ha.d/resource.d/IPAddr 192.168.10.102/24
start
heartbeat: 2004/04/29_12:28:55 info: /sbin/ifconfig eth0:0 192.168.10.102 netmask 255.255.255.0
broadcast 192.168.10.255
heartbeat: 2004/04/29_12:28:55 info: Sending Gratuitous Arp for 192.168.10.102 on eth0:0 [eth0]
heartbeat: 2004/04/29_12:28:55 /usr/lib/heartbeat/send_arp -i 500 -r 10 -p
/var/lib/heartbeat/rsctmp/send_arp/send_arp-192.168.10.102 eth0 192.168.10.102 auto
192.168.10.102 ffffffff
heartbeat: 2004/04/29_12:28:55 info: Running /etc/ha.d/resource.d/datadisk drbd2 start
heartbeat: 2004/04/29_12:28:56 info: Running /etc/ha.d/resource.d/mqsMod.pl /var/mqm/wmq2
primary start
```

---

The message generation program running against qmgr2 on wmq1 will fail and the message retrieving application will stop receiving messages from qmgr2 temporarily while the queue manager is being recovered on wmq2.

2. Now execute **dspmq** on wmq2 .

It should show that qmgr2 is running.

3. Execute **dspmq** on wmq1.

It should show that only qmgr1 is running.

4. Start the message generating program to put messages on the remote queue definition q1 on qmgr2 with the command **./msggen 1 mem qr qmgr2**. The program running against qmgr1 on host wmq1 should still be running.

Now that the message generating program is started, the results on the Web page messages.jsp should show messages arriving from both queue manager.

**Note:** If there had been messages queued on qmgr2 while the failback took place, they would have immediately flowed to qmgr3 when the failback completed because the message were persistent. This is not illustrated in this scenario but can be tested by generating large numbers of messages on qmgr2 prior to powering up wmq2. In addition it could be tested by stopping the qmgr2.T0.qmgr3 sender channel allowing messages to wait on the transmission queue.

### 3.9.5 Summary of results

The results our application start and failover test are:

1. The message receiver application was started.
2. The message generation application was started on qmgr1 and qmgr2.
3. The wmq2 server was powered off.
4. The failover took place.
5. The qmgr1 recovered from the IO wait.
6. The message generator was started on qmgr2 on host wmq1.
7. The wmq2 server was powered on.
8. The qmgr2 restarted on host wmq2.
9. The message generator was restarted using qmgr2 on wmq2.
10. Except for brief intervals, messages from both applications were received by the message receiver application.

## 3.10 Running the Client mode test

This section discusses the configuration details of WebSphere MQ when the receiving application uses Client mode to receive messages.

### 3.10.1 Queue manager configuration

Figure 3-27 illustrates the setup of queue manager `qmgr2` where messages are placed in a local queue and accessed by the receiving application by a *server connection channel*.

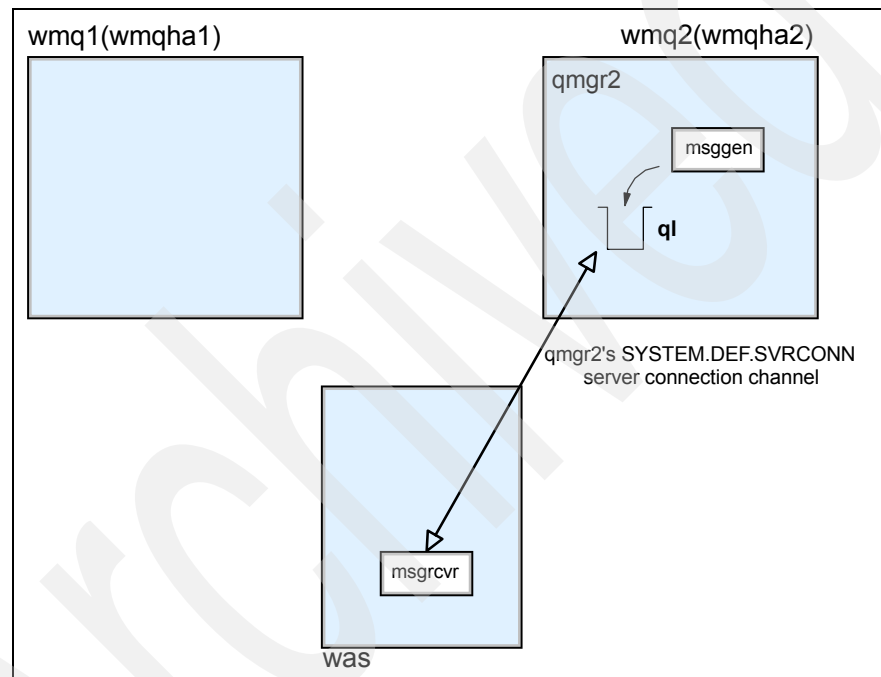


Figure 3-27 Queue manager setup for client mode test

#### Setup on qmgr1

No setup is required for `qmgr1` because it is not used in this test.

#### Setup on qmgr2

To set up on `qmgr2`, perform the following tasks:

1. Ensure that `/var/mqm/wmq2` filesystem is mounted and copy the entire directory structure from `/var/mqm/` to `/var/mqm/wmq2`.



**Restriction:** Ensure that you copy the directory structure and do not perform a move.

2. Modify the contents of `mq5.ini` to reflect the new `DefaultPrefix` and `LogDefaultPath` as shown in Example 3-55.

*Example 3-55 Modified mq5.ini*

---

```
AllQueueManagers:
*****#
#* The path to the qmgrs directory, below which queue manager data is stored *#
*****#
DefaultPrefix=/var/mqm/wmq2

ClientExitPath:
  ExitsDefaultPath=/var/mqm/exits

LogDefaults:
  LogPrimaryFiles=3
  LogSecondaryFiles=2
  LogFilePages=1024
  LogType=CIRCULAR
  LogBufferPages=0
  LogDefaultPath=/var/mqm/wmq2/log
```

---

3. Save and close the `mq5.ini` file. Use the following commands to create and start the queue manager:
  - a. `crtmqm qmgr2`
  - b. `strmqm qmgr2`
4. Now that the queue manager is created and the listener is running, execute the following commands using the `qmgr2_client.mqsc` file:

```
runmqsc qmgr2 < qmgr2_client.mqsc
```

The output will look similar to Example 3-56.

*Example 3-56 qmgr2\_client.mqsc*

---

```
*****/
* Define a remote queue
*****/
DEFINE QLOCAL('q1')
```

---

5. Execute the following command to start the queue manager listener:

```
runmq1sr -t tcp -p 3602 -m qmgr2 &
```

### Setup on qmgr3

No setup is required for qmgr3 because it is not used in this test.


### 3.10.2 Configuration of WebSphere Application Server

The only configuration change that was necessary to the server1 application server was to change the Transport Type of the queue connection factory `linuxHATestReceiverQCF`. Because the application itself has no knowledge of the runtime configuration, no changes were necessary to the `linuxHATestEAR`.

To make the change, open the WebSphere Application Server Administrative Console and return to the **WebSphere MQ JMS Provider** panel. Select the **linuxHATestReceiverQCF** and change the Transport Type from **BINDINGS** to **CLIENT**. Now the queue connection factory should look like Figure 3-28 on page 115.

[WebSphere MQ JMS Provider](#) > [WebSphere MQ Queue Connection Factories](#) >

### linuxHATestReceiverQCF

A queue connection factory is used to create connections to the associated JMS provider of JMS queue destinations, for point-to-point messaging. Use WebSphere MQ Queue Connection Factory administrative objects to manage queue connection factories for the WebSphere MQ JMS provider. 

**Configuration**

General Properties	
Scope	* cells:localhost:nodes:localhost:servers:server1
Name	* linuxHATestReceiverQCF
JNDI Name	* jms/mq/linuxHATestReceiverQCF
Description	<input type="text"/>
Category	<input type="text"/>
Component-managed Authentication Alias	(none)
Container-managed Authentication Alias	(none)
Mapping-Configuration Alias	DefaultPrincipalMapping
Queue Manager	qmgr2
Host	wmqha2
Port	3602
Channel	SYSTEM.DEF.SVRCONN
Transport Type	CLIENT

Figure 3-28 Changing Transport Type to CLIENT

6. Apply and save the changes.

### 3.10.3 WebSphere Application Server in action

In this section, the application is started under normal operation and then the failover test is executed.

#### Normal Operation

To begin operation, perform the following tasks:

1. Execute `dspmq` to ensure `qmgr2` is running on host `wmq2`, as in Example 3-57.

*Example 3-57 dspmq on wmq2*

---

```
[root@wmq2 mqm]# dspmq
QMNAME(qmgr2 STATUS(Running))
```

---

- Restart server1 on host was and ensure the same two JMS resources previously defined are discovered through JNDI again. In server1's SystemOut.1log there should be entries similar to Example 3-58.

*Example 3-58 Checking for JMS resource storage*

---

```
WSVR0049I: Binding linuxHATestReceiverQCF as jms/mq/linuxHATestReceiverQCF
WSVR0049I: Binding linuxHATestReceiver1Q as jms/mq/linuxHATestReceiver1Q
```

---

At the end of the SystemOut.1log there should also be entries that ensure the deployed application and the application server are started similar to Example 3-59.

*Example 3-59 Checking for application and server startup*

---

```
WSVR0221I: Application started: LinuxHATestEAR
ADMU3000I: Server server1 open for e-business
```

---

- If you used the default settings for the application installation, you should be able to open a web browser and point it to:

`http://was:9080/LinuxHATest/index.jsp`

The screen will look similar to Figure 3-29.



*Figure 3-29 Web page index.jsp*

- Select **Receiver Test**. See Figure 3-30 on page 117.



Figure 3-30 receiver.jsp

5. For this test, the refresh interval was set to one second to witness the closest failover time interval possible. Click **Start Monitor** to initiate the receiver. The receiver should start but because there are no messages available on the queue, the table should display *No messages received*.

### **Problem determination**

If there is a problem starting the receiver, review “Problem determination” on page 101 for some help.

### **Running the application on host qmgr2**

Change to the directory where the message generating program is present and start the program using the following command.

```
./msggen 1 mem q1 qmgr2
```

### **Running the application on WebSphere Application Server**

The messages from qmgr2 should be immediately available to the receiving application.

### **Failover scenario**

Now that we are successfully sending and receiving messages, it is time to test the failover.

1. Power off the wmq2 server. The message receiving application on the was server will stop receiving messages from wmq2 temporarily while the queue manager is recovered on wmq1.

Heartbeat software on wmq1 will detect that wmq2 is not responding. To view the tasks performed on wmq1 see the log file at /var/mqm/ha-log.

Because powering off wmq2 caused the server to immediately disappear from the local network and the WebSphere Application Server had an open TCP/IP connection to it with the JMS connection that was being used, the connection might need to wait for a client connection timeout to occur. However, if the virtual host name wmqha2 is immediately recognized, a JMS exception should be thrown immediately. Because the receiving application immediately

issues a disconnect/reconnect when an exception occurs, it should find the qmgr2 on wmq1 as soon as it is running, which might be only a few seconds.

2. Now execute **dspmq** on wmq1. It should show that qmgr2 is running. See Example 3-60.

*Example 3-60 dspmq on wmq1 after failover*

---

```
[root@wmq1 c++]# dspmq
QMNAME(qmgr2) STATUS(Running)
```

---

3. Execute the following command to start the queue manager listener on host wmq1. Note that this command could easily be added to the script that starts the queue manager itself.

```
runmq1sr -t tcp -p 3602 -m qmgr2 &
```

4. Start the message generating programs to put messages on the local queue definition q1 on qmgr2 only. Now that the message generating program is started, the results on Web page messages.jsp should show messages received from qmgr2 immediately.

**Note:** If there had been messages queued on qmgr2 while the failure took place, they would have immediately been received by the message retrieval program if they were defined as persistent.

### 3.10.4 Summary of results

The results of our test of the Client mode are:

1. The message receiver application was started.
2. The message generation application was started on qmgr2.
3. The wmq2 server was powered off.
4. The failover took place.
5. The message receiver caught a JMS exception and reconnected to qmgr2 now running on host wmq1.
6. The message generator was restarted on qmgr2 running on host wmq1.

## 3.11 Summary

This chapter demonstrated how a queue manager can be recovered on a secondary server by using a disk-mirroring configuration. Using the configuration was an enhancement from using no high-availability techniques. Implementing a high-availability solution allowed orphaned messages on the failing server's

queue manager to be recovered with only a few seconds of delay. The implementation was inexpensive and easily configured using standard hardware and no-cost high-availability software.

Archived

Archived





## Implementing HA queue managers: Part 2

This chapter discusses a WebSphere MQ configuration and two applications that demonstrate implementing a more robust queue manager recovery solution with Linux by using externally shared disks. This removes the need to use disk-mirroring, which was a feature of the first scenario and which slowed down the failover and failback.

## 4.1 Scenario two overview

This scenario expands on scenario one by showing the improvements that can be made by replacing the disk-mirroring implementation with external shared disks. Figure 4-1 illustrates an overview of the hardware topology used in this scenario.

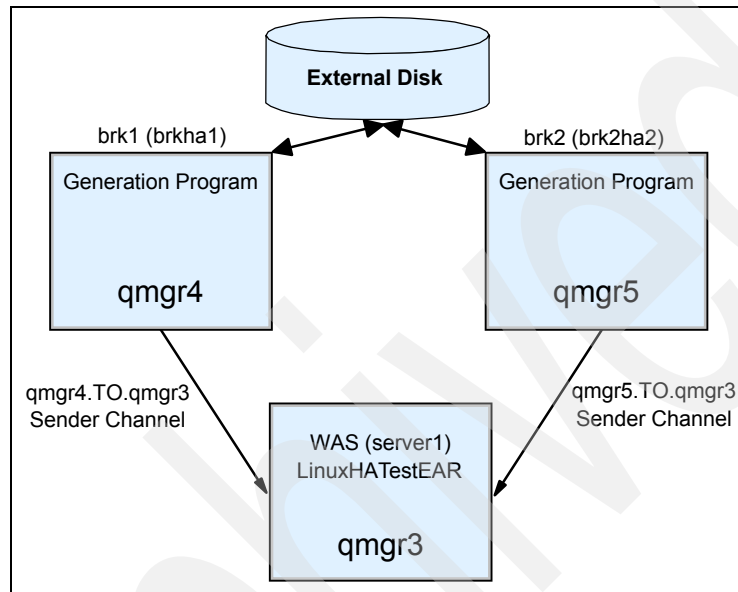


Figure 4-1 High-Availability through externally shared disks

Scenario one used only internal disks and was very inexpensive to implement. In this scenario, a shared, external disk is used but it would require the purchase of additional hardware. Therefore, this scenario would be more costly. However, this chapter will illustrate that the externally shared disk provides a more advanced failover and failback solution.

## 4.2 Implementing Linux-HA on SUSE

In this section, we discuss the implementation of Linux-HA software on servers running SUSE Linux Enterprise Server.

### 4.2.1 Planning an implementation

We begin our second highly available scenario with a plan similar to the first with inexpensive, off-the-shelf hardware. There are, however, two key changes:

1. We will not be using network disk-mirroring. Instead, we will use an external SCSI enclosure that can support up to two servers on a shared bus (IBM EXP200) and a SCSI RAID controller in each node (IBM ServeRAID™ 4M). We will see if this disk configuration will take care of the IO blocking issue discovered in Scenario one. See the related note on page 107).
2. The operating system used on the two new HA nodes is SUSE Linux Enterprise Server 8 powered by United Linux 1.0 with Service Pack 3. We will refer to it as SUSE Linux Enterprise Server. This version of SUSE Linux is based on the Linux implementation of United Linux, which was formed by Caldera, Connectiva, SuSE Linux, and Turbolinux.

As before, our goal is to have two or more queue managers running on two separate machines. When one machine fails, the other will take over the queue managers of the downed machine. Once again, we will use the heartbeat software from:

<http://www.linux-ha.org>

Figure 4-2 shows the layout of the cluster. The shared resources denoted in boldface are the primary for that node and the grayed resources are secondary.

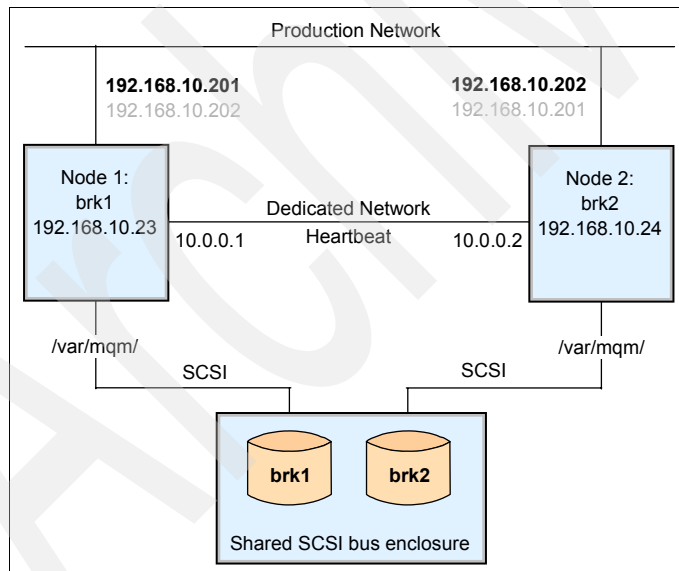


Figure 4-2 Two nodes in a HA cluster with shared resources

## 4.2.2 Installing Linux High-Availability

We installed United Linux and then applied Service Pack 3. This brought the heartbeat package up to version 1.04. We used 1.20 in our first scenario, so we decided to upgrade heartbeat to the same version.

We could find no RPM packages for heartbeat version 1.20 that would install properly under United Linux. Our only choice was to build 1.20 from source, as follows:

1. After downloading, extract `heartbeat-1.2.0.tar.gz` and change to the directory created.

```
# tar xvfz heartbeat-1.2.0.tar.gz
# cd heartbeat-1.2.0
```

2. Normally, the next step is to run the configuration script. But we ran into a problem during the compile. It appeared that on our system, the configure script fails to define the variable `SED`. So before configuring, we had to first patch the file `ltmain.sh` in the top level directory of the source code. See Example 4-1.

```
# patch -p0 < /tmp/ltmain.sh.patch
```

### Example 4-1 `ltmain.sh.patch`

---

```
*** ltmain.sh.old Tue Apr 20 09:34:13 2004
--- ltmain.sh Tue Apr 20 09:34:36 2004
*****
*** 1,3 ****
--- 1,4 ----
+ SED="/usr/bin/sed"
  # ltmain.sh - Provide generalized library-building support services.
  # NOTE: Changing this file will not affect anything until you rerun \
configure.
#
```

---

3. The source is now ready to be configured and compiled.

```
# ./ConfigureMe configure
# ./ConfigureMe package
```

4. When that completes, there should be a full set of heartbeat RPMs in your distribution's default package directory, which is usually `/usr/src/packages/RPMS/<arch>`:

```
heartbeat-1.2.0-1.i386.rpm
heartbeat-ldirectord-1.2.0-1.i386.rpm
heartbeat-pils-1.2.0-1.i386.rpm
heartbeat-stonith-1.2.0-1.i386.rpm
```

- ▶ There are perl module dependencies to take care of before the packages can be installed. Install as many of these as possible from RPM packages available for your distribution. When you have exhausted this source of perl module RPMs, you will have to install the rest from source. There are two ways to accomplish this:
  - a. For Internet-attached machines, use the CPAN module. The CPAN module will grab the necessary code and dependencies from the Internet and install them all for you.
 

```
# perl -MCPAN -e shell
cpan> install perl-ldap
```
  - b. Download the module tarballs from CPAN and build them each manually, starting with the lowest level dependencies first and working your way up.
 

```
# tar xvfz perl-ldap-0.31.tar.gz
# cd perl-ldap-0.31
# perl Makefile.PL
# make
# make test # optional
# make install
```
  - c. Repeat either step a) or b) until all dependencies are satisfied.
- ▶ It is now time to install the heartbeat RPM packages. Unless you were able to find all of the required perl modules in RPM form, we will have to use option `--nodeps` because the RPM database does not know about software installed by other means.
 

```
# rpm -Uvh heartbeat*rpm --nodeps
```

### 4.2.3 Configuring Linux-HA

The steps for configuring linux-HA in this chapter will be very similar to what was done in Chapter 3, “Implementing HA queue managers: Part 1” on page 51. Those steps are:

1. Heartbeat configuration: `/etc/ha.d/ha.cf`
2. Heartbeat authentication: `/etc/ha.d/authkeys`
3. High availability resource configuration: `/etc/ha.d/haresources`

#### Heartbeat configuration

Example 4-2 shows the configuration file `ha.cf` file that we used.

*Example 4-2* `/etc/ha.d/ha.cf`

---

```
# ha.cf - a configuration file for heartbeat
#
bcast eth1
keepalive 2
```

```
warntime 10
deadtime 30
initdead 120
udpport 694
auto_failback on
node brk1
node brk2

# for ipfail (detects network failures)
respawn hacluster /usr/lib/heartbeat/ipfail
ping host1 host2 host3
```

---

The only changes we made from Example 3-1 on page 56 were the member nodes of this cluster.

### Heartbeat authentication

The authentication file is identical to the one used in the previous chapter. Remember, you should not use the CRC method if your heartbeat is not on a dedicated network or crossover cable. When you are done, remember to set the mode to 0600.

```
# chmod 0600 /etc/ha.d/authkeys
```

Example 4-3 shows the result of this command.

*Example 4-3 /etc/ha.d/authkeys*

---

```
auth 1
1 crc
```

---

### High Availability resource configuration

In Example 4-4 we show the haresources file which, if you recall from the previous chapter, controls which node is primary for which resources and in what order those resource scripts are to be started and stopped. Starting happens left to right; stopping happens right to left.

*Example 4-4 /etc/ha.d/haresources*

---

```
brk1 IPaddr::192.168.10.201/24 ServeRAID::1::1 \
    Filesystem::-Lbrk1::/var/mqm/brk1::ext3
brk2 IPaddr::192.168.10.202/24 ServeRAID::1::2 \
    Filesystem::-Lbrk2::/var/mqm/brk2::ext3
```

---

The haresources is now quite a bit different than Example 3-3 on page 57, although each line starts out the same. The first line declares node brk1 to be

primary for the resources listed on the remainder of the line. See the following note.

**Important:** If you do not have `auto_failback on` in the `/etc/ha.d/ha.cf` file, then the concept of the primary and secondary node is lost. The resources will be owned by whichever node requested them last. This option is equivalent to `nice_failback off` in older versions of heartbeat.

**Tip:** In order to use DNS hostname with the highly available, virtual-IP addresses, new names will have to be added to DNS. Do not use the host names of the nodes themselves (`brk1` and `brk2` in our case), because those names do not resolve to addresses that are highly available.

Example DNS entry:

```
#ORIGIN some.net
brk1 IN A 192.168.10.23
brk2 IN A 192.168.10.24
brkha1 IN A 192.168.10.201
brkha2 IN A 192.168.10.202
```

These are the resource scripts for node `brk1`, as shown in Example 4-4:

1. `IPaddr::192.168.10.201/24`
2. `ServerRAID::1::1`
3. `Filesystem::-Lbrk1::/var/mqm/brk1::ext3`

The `IPaddr` script has already been covered in 3.2.6, “High availability resource configuration” on page 57. We use it here just as we did earlier, but with a different IP address.

The `ServerRAID` and `Filesystem` scripts are new to this scenario. The `ServerRAID` script is used to move the shared SCSI drives between nodes. The nodes cannot have simultaneous access to the drives in this configuration. The `Filesystem` script is used to mount and unmount the shared drives as needed.

► Use of `ServerRAID`:

```
# ServerRAID <controller#> <merge_group#> <start|stop>
```

► Use of `Filesystem`:

```
# Filesystem <dev> <mount_point> <fs_type> <start|stop>
```

We did not have to use the `Filesystem` script in our first scenario because the `datadisk` script, which is part of the `drbd` network mirror code, handled mounting and unmounting of filesystems for us.

## 4.2.4 Configuring and testing the shared SCSI drives

To perform a base configuration of shared SCSI drives for Linux, review and follow the instructions in Appendix B, “Using external SCSI storage enclosure with Linux” on page 321. Once that is complete, the nodes will be ready to test the heartbeat resource scripts that pertain to the ServeRAID setup in the `/etc/ha.d/haresources` file.

1. You might need to modify the ServeRAID script before using it. We found that in its default state, it is not compatible with the version of `ipssend` that came on the ServeRAID Support CD version 6.11. Example 4-5 is a simple patch to correctly configure the script for ServeRAID 6.11:

*Example 4-5 serveraid.patch*

---

```
*** ServerRAID.old Tue Apr 20 09:06:01 2004
--- ServerRAID Tue Apr 20 09:06:21 2004
*****
*** 286,295 ****
    : OK All is well!
    targetid=`MergeGroupToSCSI_ID`
    sr_logicaldrivenumber=`expr $targetid + 1`
!   #run $IPS SYNCH $sr_adapter $sr_logicaldrivenumber &
    # This version of the SYNCH command requires the 6.10 or later
    # ServeRAID support CD.
!   run $IPS SYNCH $sr_adapter GROUP $sr_mergegroup &
    AddSCSI
    else
    return $?
--- 286,295 ----
    : OK All is well!
    targetid=`MergeGroupToSCSI_ID`
    sr_logicaldrivenumber=`expr $targetid + 1`
!   run $IPS SYNCH $sr_adapter $sr_logicaldrivenumber &
    # This version of the SYNCH command requires the 6.10 or later
    # ServeRAID support CD.
!   #run $IPS SYNCH $sr_adapter GROUP $sr_mergegroup &
    AddSCSI
    else
    return $?
```

---

As you can see, it is a simple patch that comments one line and uncomments another. Pay no attention to the line regarding the `SYNCH` command and version 6.10 or later. We found that to be untrue for our 6.11 code.

2. Apply the patch:

```
# cd /etc/ha.d/resource.d
# patch -p0 < /tmp/serveraid.patch
```



**Tip:** If you installed the IBM ServeRAID Manager of the support CD, now is a good time to start it. It will graphically display which logical drives, if any, are currently connected to each ServeRAID controller. Run this command:

```
# /usr/RaidMan/RaidMan.sh &
```

3. If the shared drive in merge group 1 is not already present on the ServeRAID controller, make it so with this command:

```
# cd /etc/ha.d/resource.d  
# ./ServeRAID 1 1 start
```

The shared drive in merge group 1 should now be present on ServeRAID controller 1.

4. Bring over the shared drive in merge group 2 if it is not already present:

```
# ./ServeRAID 1 2 start
```

The shared drive in merge group 2 should now be present on controller 1.

ServeRAID synchronizes each logical drive when they are started by calling `ipssend sync`. This happens during every start. Each ServeRAID controller can only synchronize one drive at a time. Therefore, you might have received an error message when you started the second drive in the above tests. The error is not fatal and the filesystem should still be ready for use. However, care should be taken during testing when the two nodes are likely to be going up and down quite often. The synchronization occurs in the background and can take many hours to complete depending on drive size. During this time drive performance might be degraded.

**Important:** If the previous ServeRAID commands did not work, then verify that you are using the correct **SYNC** line in the ServeRAID script as shown in the patch file at Example 4-5. If you still encounter problems, then verify the hardware configuration. Use the information in Appendix B, “Using external SCSI storage enclosure with Linux” on page 321 as a guide.

Because of the way the ServeRAID cards merge and unmerge the logical drives, there is no guarantee in which order the drives will appear to the operating system. For example, sometimes the same shared drive might be located at `/dev/sdb1`, at other times it might be at `/dev/sdc1`. This requires that we use labels to make sure we are always using the correct drive on the correct mount point.

5. Add drive labels to each disk so that **Filesystem** can reference the drive by label, and not by the device name which is not static. Your devices and labels will probably be different, so be certain that you are operating on the correct

device in each case. For clarity, we label the drives with the same name as the node that will be its primary host.

- For ext3 filesystems:

```
# tune2fs -L brk1 /dev/sdb1
# tune2fs -L brk2 /dev/sdc1
# ### e2label also works for extX filesystems
```

- For reiserfs filesystems ):

```
# reiserfstune --label brk1 /dev/sdb1
# reiserfstune --label brk2 /dev/sdb2
```

For further information, see:

<http://www.namesys.com>

- For jfs filesystems:

```
# jfs_tune -l brk1 /dev/sdb1
# jfs_tune -l brk2 /dev/sdb2
```

For further information, see:

<http://oss.software.ibm.com/jfs>

- For xfs filesystems:

```
# xfs_admin -L brk1 /dev/sdb1
# xfs_admin -L brk2 /dev/sdc1
```

For further information, see:

<http://oss.sgi.com/projects/xfs>

6. Create mount points with which to test Filesystem.

```
# mkdir -p /var/mqm/brk1
# mkdir -p /var/mqm/brk2
```

7. There is no need to add entries to /etc/fstab because **Filesystem** handles that automatically. Test this script now and remember to change the values to ones that make sense for your configuration:

```
# cd /etc/ha.d/resource.d
# ./Filesystem -Lbrk1 /var/mqm/brk1 ext3 start
# ./Filesystem -Lbrk2 /var/mqm/brk2 ext3 start
```

8. Run **mount** or **df** to verify that the filesystems are mounted. Create a file to test write capability.

```
# df -h
# touch /var/mqm/brk1/brk1_test
# ls /var/mqm/brk1
```

Repeat this for the other drive if you wish.

9. We now need to test that we can use the HA resource scripts to reverse what we have done so far.

```
# cd /etc/ha.d/resource.d
# ./Filesystem -Lbrk2 /var/mqm/brk2 ext3 stop
# ./Filesystem -Lbrk1 /var/mqm/brk1 ext3 stop
# ./ServeRAID 1 2 stop
# ./ServeRAID 1 1 stop
```

10. Repeat the steps in this section on the other node, remembering to first patch the ServeRAID script. You should be able to see the test file you created on node 1. Once you are confident that the resource scripts are working correctly on each node, you can begin testing heartbeat's failover and failback mechanisms then move on to installing WebSphere MQ.

### 4.2.5 Validating heartbeat failover and failback

Refer to Section 3.2.8, “Validating heartbeat failover and failback” on page 64 for techniques to monitor and validate failback and failover situations. The only method that does not carry over from that section to this section is the material relating to the shared drive system.

The **mount** and **df** tools are still useful in this scenario. For more ServeRAID specific information, use the GUI **RaidMan.sh** tool or **ipssend**:

```
# /usr/RaidMan/RaidMan.sh &
```

Figure 4-3 on page 132 shows a ServeRAID Manager screen showing the system in a normal state, one drive available on each node.

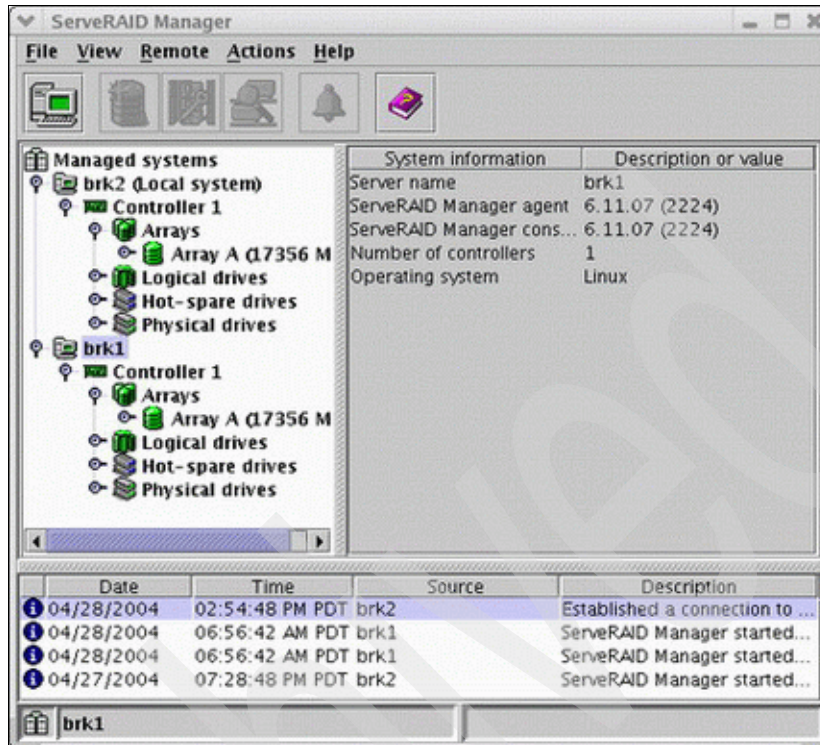


Figure 4-3 ServeRAID Manager: normal

In the next screen (Figure 4-4 on page 133), we see the system in a failover state. In the bottom, right-hand corner you can see the progress bar for the synchronization.

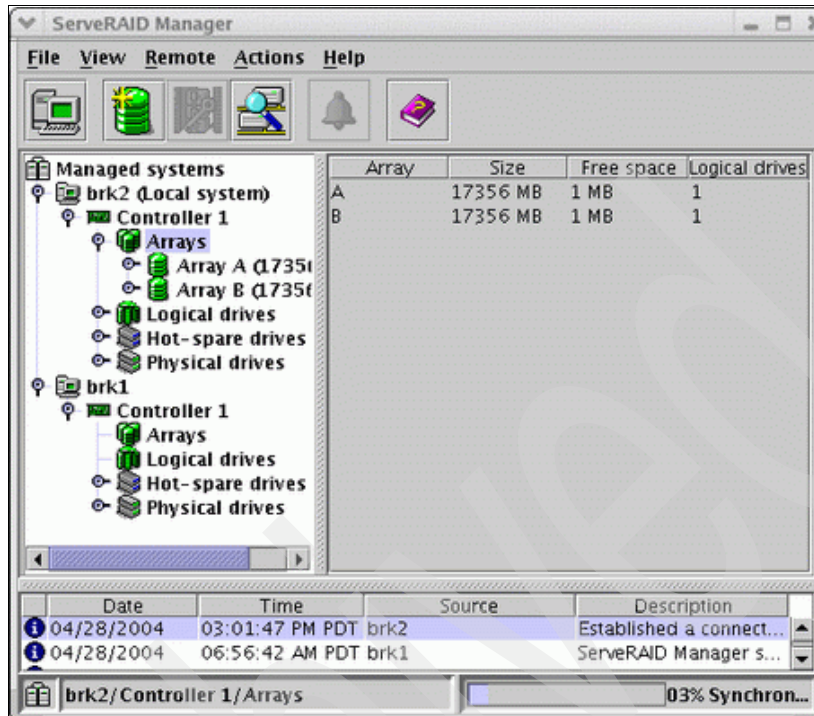


Figure 4-4 ServeRAID Manager: failover

1. For command line access to the ServeRAID controller, there is the **ipssend** command. This is the same command that the heartbeat resource script **ServerRAID** uses under the covers.

```
# ipssend getstatus 1
```

Example 4-6 shows the output from **ipssend**.

Example 4-6 *ipssend getstatus*

---

```
Found 1 IBM ServeRAID controller(s).
Background command progress status for controller 1...
Current/most recent operation : Synchronize
Logical drive                  : 1
Background operation active    : No
Status                          : Successfully completed
Logical drive size (in stripes): 1110784
Number of remaining stripes    : 0
Percentage complete            : 100.00%
Command completed successfully.
```

---

2. This command will show which arrays, if any, are on controller 1:

```
# ipssend getconfig 1 | grep Array
```

Example 4-7 shows this output from this command.

*Example 4-7 ipssend getconfig: failover condition*

---

```
Array A stripe order (Channel/SCSI ID) : 1,2 1,3  
Array B stripe order (Channel/SCSI ID) : 1,0 1,1
```

---

Running that command on the failed node will not return any lines.

## 4.2.6 HA configuration summary

In summary, Table 4-1 represents our high availability configuration:

*Table 4-1 Summary of HA configuration*

Node(s) Up:	IP Addresses:	Mounted Filesystems:
brk1 and brk2	brk1 - 192.168.10.101 brk2 - 192.168.10.102	brk1 - /var/mqm/brk1 brk2 - /var/mqm/brk2
brk1 only	brk1 - 192.168.10.101 brk1 - 192.168.10.102	brk1 - /var/mqm/brk1 brk1 - /var/mqm/brk2
brk2 only	brk2 - 192.168.10.102 brk2 - 192.168.10.101	brk2 - /var/mqm/brk2 brk2 - /var/mqm/brk1

## 4.3 Installing and configuring WebSphere MQ

Perform the same steps outlined in 3.3, “Installing and configuring WebSphere MQ” on page 68. There are no specific installation requirements for this scenario.

## 4.4 Message generation application

The message generating program used in this scenario is the same application used in the first scenario. See 3.4.1, “Design of the message generation application” on page 71 and 3.4.2, “Developing the message generation application” on page 71. No changes are required to the application.

### 4.4.1 Compiling and running the message generation application

Perform the following tasks to compile and run the application:

1. Copy the `msggen.cpp` program to any directory, for example `/tmp/mqcpp`.
2. Compile and link the program using the following command:

```
g++ -o msggen ./msggen.cpp -limqb23g1 -limqs23g1
```

The C++ libraries provided by WebSphere MQ will look for `libstdc++.so.3`. If this file is not present in `/usr/lib`, the loader should give the following error message similar to Example 4-8.

*Example 4-8 libstdc++.so.3 dependency error*

---

```
warning : libstdc++.so.3 needed by /usr/lib/libimqb23g1.so not found (try using
-rpath or -rpath-link)
```

---

The error in Example 4-8 can be solved in two ways:

- ▶ Locate where `libstdc++.so.3` is present and add a symbolic link to the same in `/usr/lib`.
- ▶ Create a symbolic link to the existing `libstdc++.*` as `libstdc++.so.3`.

## 4.5 Messaging retrieval application

The message retrieval application is same as the one used in the first scenario. Refer to 3.5, “Message retrieval application” on page 73.

## 4.6 WebSphere MQ HA scripts and configuration

This section discusses the configuration of MQ within an HA environment. We also discuss a number of scripts that assist in such an environment.

### 4.6.1 Normal scenario

Under normal operations, `qmgr4` will be running on `brk1`, `qmgr5` will be running on `brk2` and file systems will be mounted from a shared disk, as illustrated in Figure 4-5 on page 136.

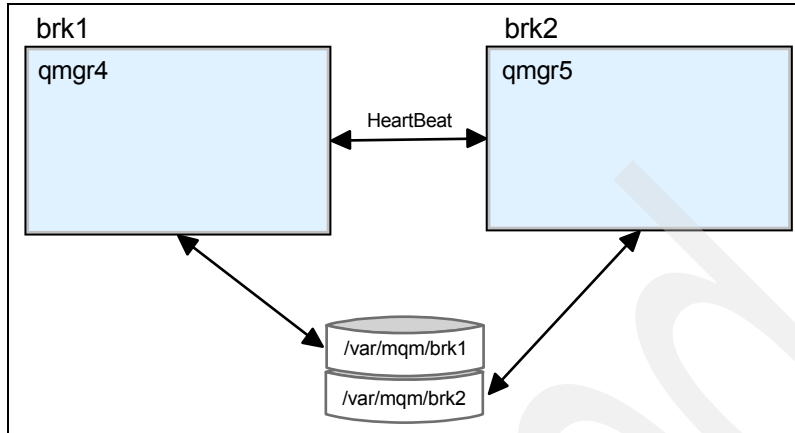


Figure 4-5 Scenario two under normal operation

## 4.6.2 Configuring brk1 and brk2 for WebSphere MQ HA

For WebSphere MQ HA to work in this scenario, the following configurations are required on both brk1 and brk2.

We will again be using the MQ scripts that were discussed earlier in “Shell scripts to manage MQ resources” on page 80. Make these scripts available on the hosts brk1 and brk2. Copy also the script `mqsMod.pl` to `/etc/ha.d/resource.d`.

### High Availability resource configuration

Modify `etc/ha.d/haresources` file on both brk1 and brk2 to include the start and stop of the queue managers. Example 4-9 shows the contents of this file on host brk1. Similar changes are needed to this file on host brk2.

*Example 4-9 Configuration file haresources on host brk1*

---

```
brk1    IPaddr::192.168.10.201/24 ServeRAID::1::1
Filesystem::-Lbrk1::/var/mqm/brk1::ext3 mqsMod.pl::/var/mqm/brk1::primary
brk2    IPaddr::192.168.10.202/24 ServeRAID::1::2
Filesystem::-Lbrk2::/var/mqm/brk2::ext3 mqsMod.pl::/var/mqm/brk2::secondary
```

---

## 4.7 Running the shared-disk test

This section discusses the configuration details of WebSphere MQ when the receiving application uses Bindings mode to receive messages.



## 4.7.1 Queue manager configuration

Figure 4-6 illustrates the queue manager setup, where messages are sent from qmgr4 and qmgr5 to qmgr3.

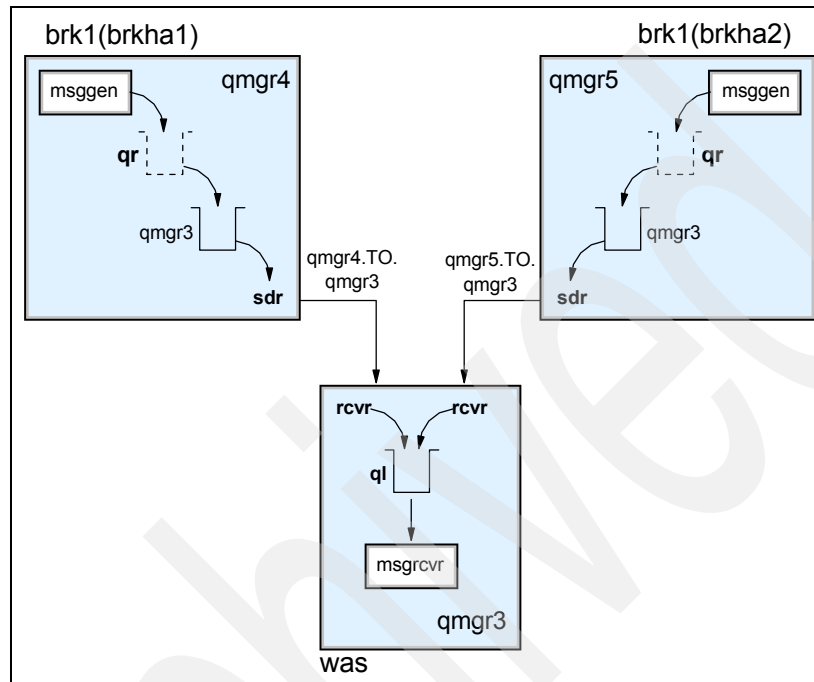


Figure 4-6 Queue Manager setup for a shared-disk configuration

### Setup on qmgr4

Use the following commands to create and start the queue manager qmgr4:

1. `crtmqm qmgr4`
2. `strmqm qmgr4`

Now that the queue manager is created, execute the following commands using the `qmgr4.mqsc` file. The contents of this file is shown in Example 4-10.

3. `runmqsc qmgr4 < qmgr4.mqsc`

Example 4-10 `qmgr4.mqsc`

```
*****/
* Define a remote queue
*****/
DEFINE QREMOTE('qr')  +
          RNAME('q1')  +
          RQMNAME('qmgr3') +
```

```

XMITQ('qmgr3') +
DEFPSIST(YES)

*****/
* Define the xmitq and trigger to start the channel automatically
*****/
DEFINE QLOCAL('qmgr3') +
    TRIGGER +
    INITQ(SYSTEM.CHANNEL.INITQ) +
    TRIGTYPE(FIRST) +
    USAGE(xmitq) +
    MAXDEPTH(1000000) +
    TRIGDATA('qmgr4.TO.qmgr3') +
    DEFPSIST(YES)

*****/
* define sender channel
*****/
DEFINE CHANNEL('qmgr4.TO.qmgr3') +
    CHLTYPE(SDR) +
    TRPTYPE(TCP) +
    CONNAME('was (3600)') +
    XMITQ('qmgr3')

```

---

## Setup on qmgr5

Use the following commands to create and start the queue manager:

1. **crtmqm qmgr5**
2. **strmqm qmgr5**

Now that the queue manager is created, execute the following commands using the qmgr5.mqsc file. The contents of this file is shown in Example 4-11.

3. **runmqsc qmgr5 < qmgr5.mqsc**

*Example 4-11 qmgr5.mqsc*

```

*****/
* Define a remote queue
*****/
DEFINE QREMOTE('qr') +
    RNAME('q1') +
    RQMNAME('qmgr3') +
    XMITQ('qmgr3') +
    DEFPSIST(YES)

*****/
* Define the xmitq and trigger to start the channel automatically
*****/

```

```

DEFINE QLOCAL('qmgr3') +
    TRIGGER +
    INITQ(SYSTEM.CHANNEL.INITQ) +
    TRIGTYPE(FIRST) +
    USAGE(xmitq) +
    TRIGDATA('qmgr5.TO.qmgr3') +
    DEFPSIST(YES)

*****/
* define sender channel
*****/
DEFINE CHANNEL('qmgr5.TO.qmgr3') +
    CHLTYPE(SDR) +
    TRPTYPE(TCP) +
    CONNAME('was (3600)') +
    XMITQ('qmgr3')

```

---

### Setup on qmgr3

Additional configuration is required for qmgr3. Execute the following commands using the qmgr3\_scenario2.mqsc file. These commands add two additional receiver channels to the configuration of qmgr3.

#### 1. runmqsc qmgr3 < qmgr3\_scenario2.mqsc

See Example 4-12.

#### Example 4-12 qmgr3\_scenario2.mqsc

```

*****/
* define receiver channel
*****/
DEFINE CHANNEL('qmgr4.TO.qmgr3') +
    CHLTYPE(RCVR) +
    TRPTYPE(TCP)

*****/
* define receiver channel
*****/
DEFINE CHANNEL('qmgr5.TO.qmgr3') +
    CHLTYPE(RCVR) +
    TRPTYPE(TCP)

```

---

#### 2. If the listener is not started, execute the following command to start the queue manager listener:

```
runmq1sr -t tcp -p 3600 -m qmgr3 &
```

## 4.7.2 Configuration verification

To perform an initial validation of the configuration, we can use some of the sample programs that are provided by WebSphere MQ.

1. Execute the following command on brk1 to put messages on the remote queue:

```
./opt/mqm/samp/bin/amqsput qr qmgr4  
Hello World!
```

Press the enter key enter two times.

2. Execute the following command on brk1 to ensure that the channels are in running state. The output should be similar to Example 4-13.

### *Example 4-13 Channel status on qmgr4*

---

```
brk1:/ # runmqsc qmgr4  
5724-B41 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.  
Starting MQSC for queue manager qmgr4.  
  
dis chs(*)  
1 : dis chs(*)  
AMQ8417: Display Channel Status details.  
CHANNEL(qmgr4.TO.qmgr3) XMITQ(qmgr3)  
CONNAME(was(3600)) CURRENT  
CHLTYPE(SDR) STATUS(RUNNING)  
RQMNAME(qmgr3)
```

---

3. Execute the following command on brk2 to put messages on the remote queue:

```
./opt/mqm/samp/bin/amqsput qr qmgr5  
Hello World!
```

Press the enter key enter two times.

4. Execute the following command on brk2 to ensure that the channels are in running state. The output should be similar to Example 4-14.

### *Example 4-14 Channel status on qmgr5*

---

```
brk2:/ # runmqsc qmgr5  
5724-B41 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.  
Starting MQSC for queue manager qmgr5.  
  
dis chs(*)  
1 : dis chs(*)  
AMQ8417: Display Channel Status details.  
CHANNEL(qmgr5.TO.qmgr3) XMITQ(qmgr3)  
CONNAME(was(3600)) CURRENT
```

```
CHLTYPE(SDR)                STATUS(RUNNING)
RQMNAME(qmgr3)
```

---

- Execute the following command on the WebSphere Application Server to retrieve the messages from the local queue:  

```
./opt/mqm/samp/bin/amqsget q1 test
```

Two messages from the queue q1 should be displayed.  
Use **Control-C** to stop the amqsget program.
- Execute the following command on was server to ensure that the channels are in running state. The output should be similar to Example 4-15.

*Example 4-15 Channel status on qmgr3*

---

```
[root@was mqm]# runmqsc qmgr3
5724-B41 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.
Starting MQSC for queue manager qmgr3.
```

```
dis chs(*)
  1 : dis chs(*)
AMQ8417: Display Channel Status details.
CHANNEL(qmgr4.TO.qmgr3)      XMITQ( )
CONNAME(192.168.10.23)      CURRENT
CHLTYPE(RCVR)                STATUS(RUNNING)
RQMNAME(qmgr4)
AMQ8417: Display Channel Status details.
CHANNEL(qmgr5.TO.qmgr3)      XMITQ( )
CONNAME(192.168.10.24)      CURRENT
CHLTYPE(RCVR)                STATUS(RUNNING)
RQMNAME(qmgr5)
```

---

### 4.7.3 WebSphere Application Server configuration

The only configuration change that was necessary to the server1 application server was to change the queue connection factory `linuxHATestReceiverQCF` Transport Type. Since the application itself has no knowledge of the runtime configuration, no changes were necessary to the `linuxHATestEAR`.

- To make the change, open the WebSphere Application Server Administrative Console and return to the WebSphere MQ JMS Provider panel. Select the `linuxHATestReceiverQCF` and change the Transport Type from `CLIENT` to `BINDINGS`. Now the QCF should look like Figure 3-15 on page 96.
- Apply the changes and save the configuration.

## 4.7.4 Application in action

In this section, we start the application under normal operation and then execute the failover test.

### Normal Operation

To begin normal operation, perform the following tasks:

1. Execute **dspm** on host brk1 to ensure qmgr4 is running. See Example 4-16.

*Example 4-16 dspm on brk1*

---

```
brk1:/geetha/2ndScenario/c++ # dspm  
QMNAME(qmgr4) STATUS(Running)
```

---

2. Execute **dspm** on host brk2 to ensure qmgr5 is running. See Example 4-17.

*Example 4-17 dspm on brk2*

---

```
brk2:/geetha/2ndScenario/c++ # dspm  
QMNAME(qmgr5) STATUS(Running)
```

---

3. Execute **dspm** on host was to ensure qmgr3 is running. See Example 4-18.

*Example 4-18 dspm on was*

---

```
[root@was mqm]# dspm  
QMNAME(qmgr3) STATUS(Running)
```

---

4. Restart `server1` and ensure the two JMS resources previously defined are discovered through JNDI. In `server1's SystemOut.log` there should be entries similar to Example 4-19.

*Example 4-19 Checking for JMS resource storage*

---

```
WSVR0049I: Binding linuxHATestReceiverQCF as jms/mq/linuxHATestReceiverQCF  
WSVR0049I: Binding linuxHATestReceiver1Q as jms/mq/linuxHATestReceiver1Q
```

---

At the end of the `SystemOut.log` there should also be entries that ensure the deployed application and the application server are started similar to Example 4-20.

*Example 4-20 Checking for application and server startup*

---

```
WSVR0221I: Application started: LinuxHATestEAR  
ADMU3000I: Server server1 open for e-business
```

---

5. If you used the default settings during the application installation, you should be able to open a web browser and point it to:

<http://was:9080/LinuxHATest/index.jsp>

The screen should look similar to Figure 4-7.



Figure 4-7 Web page index.jsp

6. Select **Receiver Test**.



Figure 4-8 Web page receiver.jsp

7. For this test, the refresh interval was set to one second to witness the closest failover time interval possible. Click **Start Monitor** (Figure 4-8) to initiate the receiver. The receiver should start but, because there are no messages available on the queue, the table should display No messages received.
8. If there is an error accessing the queue manager, the top of the error will be displayed, for example Error receiving messages: MQJMS2008: failed to open MQ queue.

If you do encounter an error, check server1's SystemOut.log and SystemErr.log to get the details of the problem. If there is a problem starting the receiver, review the "Problem determination" on page 101 for help.

9. On host brk1, change to the directory where the message generating program is present and start the same to put messages on the remote queue definition qr on qmgr1 using the following command.

```
./msggen 1 cpu qr qmgr4
```

10. On host brk2, change to the directory where the message generating program is present and start the same to put messages on the remote queue definition qr on qmgr1 using the following command.

```
./msggen 1 mem qr qmgr4
```

The messages from qmgr4 and qmgr5 will be sent to the queue q1 on qmgr3. The message receiver program should immediately retrieve the messages and display the messages as illustrated in Figure 4-9.

9	cpu,2004-03-29-13.55.18,qmgr4,76
8	mem,2004-03-29-13.55.18,qmgr5,0
7	cpu,2004-03-29-13.55.17,qmgr4,55
6	mem,2004-03-29-13.55.17,qmgr5,0
5	cpu,2004-03-29-13.55.16,qmgr4,45
4	mem,2004-03-29-13.55.16,qmgr5,0
3	cpu,2004-03-29-13.55.15,qmgr4,23
2	cpu,2004-03-29-13.55.14,qmgr4,27
1	No messages received
0	No messages received

Figure 4-9 Results with messages.jsp before failover

## Failover scenario

Now that we are successfully sending and receiving messages, it is time to test the failover.

1. Power off the brk2 server. The message receiving application on the was server will stop receiving messages from brk2 temporarily while the queue manager is recovered on brk1.

Heartbeat software on wmq1 will detect that brk2 is not responding. To view the tasks performed on brk1 see the log file at /var/mqm/ha-log. The output from the ha-log should be similar to Example 4-21.

### Example 4-21 ha-log during failover

---

```

heartbeat: 2004/04/29_13:55:59 WARN: node brk2: is dead
heartbeat: 2004/04/29_13:55:59 WARN: No STONITH device configured.
heartbeat: 2004/04/29_13:55:59 WARN: Shared disks are not protected.
heartbeat: 2004/04/29_13:55:59 info: Resources being acquired from brk2.
heartbeat: 2004/04/29_13:55:59 info: Link brk2:eth0 dead.
heartbeat: 2004/04/29_13:55:59 info: Running /etc/ha.d/rc.d/status status
heartbeat: 2004/04/29_13:55:59 info: Taking over resource group IPaddr::192.168.10.202/24
heartbeat: 2004/04/29_13:55:59 info: Acquiring resource group: brk2 IPaddr::192.168.10.202/24
ServerAID::1::2 Filesystem::-Lbrk2::/var/mqm/brk2::ext3 mqsMod.pl::/var/mqm/brk2::secondary
heartbeat: 2004/04/29_13:56:00 info: Local Resource acquisition completed.
heartbeat: 2004/04/29_13:56:00 info: Running /etc/ha.d/resource.d/IPaddr 192.168.10.202/24
start
heartbeat: 2004/04/29_13:56:00 info: /sbin/ifconfig eth0:1 192.168.10.202 netmask 255.255.255.0
broadcast 192.168.10.255

```



```

heartbeat: 2004/04/29_13:56:00 info: Sending Gratuitous Arp for 192.168.10.202 on eth0:1 [eth0]
heartbeat: 2004/04/29_13:56:00 /usr/lib/heartbeat/send_arp -i 500 -r 10 -p
/var/lib/heartbeat/rsctmp/send_arp/send_arp-192.168.10.202 eth0 192.168.10.202 auto
192.168.10.202 ffffffff
heartbeat: 2004/04/29_13:56:00 info: Running /etc/ha.d/resource.d/ServerRAID 1 2 start
heartbeat: 2004/04/29_13:56:00 info: Found 1 IBM ServerRAID controller(s). Configuration merge
has been initiated for controller 1... Command completed successfully.
heartbeat: 2004/04/29_13:56:13 info: Running /etc/ha.d/resource.d/Filesystem -Lbrk2
/var/mqm/brk2 ext3 start
heartbeat: 2004/04/29_13:56:19 info: Running /etc/ha.d/resource.d/mqsMod.pl /var/mqm/brk2
secondary start
heartbeat: 2004/04/29_13:56:26 info: /usr/lib/heartbeat/mach_down: nice_failback: foreign
resources acquired
heartbeat: 2004/04/29_13:56:26 info: mach_down takeover complete.
heartbeat: 2004/04/29_13:56:26 info: mach_down takeover complete for node brk2.

```

The output on Web page messages.jsp should be similar to Figure 4-10.

42	cpu,2004-03-29-13.55.40,qmgr4,42
41	cpu,2004-03-29-13.55.39,qmgr4,92
40	cpu,2004-03-29-13.55.38,qmgr4,3
39	cpu,2004-03-29-13.55.37,qmgr4,24
38	cpu,2004-03-29-13.55.36,qmgr4,34
37	cpu,2004-03-29-13.55.35,qmgr4,88
36	cpu,2004-03-29-13.55.34,qmgr4,26
35	cpu,2004-03-29-13.55.33,qmgr4,77
34	cpu,2004-03-29-13.55.32,qmgr4,37
33	cpu,2004-03-29-13.55.31,qmgr4,22
32	mem,2004-03-29-13.55.30,qmgr5,0

Figure 4-10 Displaying results of messages.jsp during failover

Notice that during the failover, messages continue to be received from qmgr4 in a timely manner. This demonstrates the improvement over scenario one. In this scenario, there was no IO wait time while the recovery for qmgr5 took place.

2. Now execute **dspmq** on brk1. It should show that qmgr1 and qmgr2 are both running as shown in Example 4-22.

Example 4-22 dspmq on brk1 after failover

```

brk1:/geetha/2ndScenario/c++ # dspmq
QMNAME(qmgr4)                                STATUS(Running)
QMNAME(qmgr5)                                STATUS(Running)

```

3. To ensure that the queue manager is working properly, start the message generating programs to put messages on the remote queue definition qr on qmgr5 by executing the command `msggen 1 mem qr qmgr5`. The program running against qmgr4 should still be running. Now that the message generating program is started, the results on Web page messages.jsp should show messages arriving from both queue managers.

**Note:** If messages had been queued on qmgr5 while the failure took place, they would have immediately flowed to qmgr3 when the failover completed because the messages were persistent. We did not illustrate this in this scenario, but you can test it by:

- ▶ Generating large numbers of messages on qmgr5 prior to powering off
- ▶ Stopping the qmgr5.T0.qmgr3 sender channel, allowing messages to wait on the transmission queue.

## Failback scenario

Now that both qmgr4 and qmgr5 are running on brk1, it's time to test the failback.

**Note:** Fail-back in this context means moving qmgr5 back to brk2.

Power on the brk2 server. When brk2 server has completed its restart, the heartbeat software on brk2 should immediately takeover qmgr5. To view the tasks performed on brk1 see the log file at `/var/mqm/ha-log` on brk1. The content of the file should be similar to Example 4-23.

### *Example 4-23 ha-log on brk1 during failback*

```
heartbeat: 2004/04/29_13:58:36 info: Heartbeat restart on node brk2
heartbeat: 2004/04/29_13:58:36 info: Link brk2:eth0 up.
heartbeat: 2004/04/29_13:58:36 info: Status update for node brk2: status up
heartbeat: 2004/04/29_13:58:36 info: Running /etc/ha.d/rc.d/status status
heartbeat: 2004/04/29_13:58:36 info: Status update for node brk2: status active
heartbeat: 2004/04/29_13:58:36 info: remote resource transition completed.
heartbeat: 2004/04/29_13:58:36 info: brk1 wants to go standby [foreign]
heartbeat: 2004/04/29_13:58:36 info: standby: brk2 can take our foreign resources
heartbeat: 2004/04/29_13:58:36 info: give up foreign HA resources (standby).
heartbeat: 2004/04/29_13:58:36 info: Running /etc/ha.d/rc.d/status status
heartbeat: 2004/04/29_13:58:36 info: Releasing resource group: brk2 IPAddr::192.168.10.202/24
ServerRAID::1::2 Filesystem::-Lbrk2::/var/mqm/brk2::ext3 mqsMod.pl::/var/mqm/brk2::secondary
heartbeat: 2004/04/29_13:58:36 info: Running /etc/ha.d/resource.d/mqsMod.pl /var/mqm/brk2
secondary stop
heartbeat: 2004/04/29_13:58:42 info: Running /etc/ha.d/resource.d/Filesystem -Lbrk2
/var/mqm/brk2 ext3 stop
heartbeat: 2004/04/29_13:58:43 info: Running /etc/ha.d/resource.d/ServeRAID 1 2 stop
```

```

heartbeat: 2004/04/29_13:58:44 info: Found 1 IBM ServeRAID controller(s). Configuration merge
has been initiated for controller 1... Command completed successfully.
heartbeat: 2004/04/29_13:58:44 info: Running /etc/ha.d/resource.d/IPAddr 192.168.10.202/24 stop
heartbeat: 2004/04/29_13:58:44 info: /sbin/route -n del -host 192.168.10.202
heartbeat: 2004/04/29_13:58:44 info: /sbin/ifconfig eth0:1 down
heartbeat: 2004/04/29_13:58:44 info: IP Address 192.168.10.202 released
heartbeat: 2004/04/29_13:58:44 info: foreign HA resource release completed (standby).
heartbeat: 2004/04/29_13:58:44 info: Local standby process completed [foreign].
heartbeat: 2004/04/29_13:58:45 info: Found 1 IBM ServeRAID controller(s). Synchronize has been
initiated for controller 1... Synchronizing logical drive #2:
.....
.....Done logical drive 2 Command completed successfully.
heartbeat: 2004/04/29_13:58:55 WARN: 1 lost packet(s) for [brk2] [17:19]
heartbeat: 2004/04/29_13:58:55 info: remote resource transition completed.
heartbeat: 2004/04/29_13:58:55 info: No pkts missing from brk2!
heartbeat: 2004/04/29_13:58:55 info: Other node completed standby takeover of foreign
resources.

```

- 
4. To view the tasks performed on brk2 see the log file at /var/mqm/ha-log on brk2. The content of the file should be similar to Example 4-24.

*Example 4-24 ha-log on brk2 on fail-back*

---

```

heartbeat: 2004/04/29_13:58:32 info: Configuration validated. Starting heartbeat 1.2.0
heartbeat: 2004/04/29_13:58:32 info: heartbeat: version 1.2.0
heartbeat: 2004/04/29_13:58:34 info: Heartbeat generation: 45
heartbeat: 2004/04/29_13:58:35 info: UDP Broadcast heartbeat started on port 694 (694)
interface eth0
heartbeat: 2004/04/29_13:58:35 info: pid 2011 locked in memory.
heartbeat: 2004/04/29_13:58:35 info: Local status now set to: 'up'
heartbeat: 2004/04/29_13:58:36 info: pid 2131 locked in memory.
heartbeat: 2004/04/29_13:58:36 info: pid 2132 locked in memory.
heartbeat: 2004/04/29_13:58:36 info: pid 2133 locked in memory.
heartbeat: 2004/04/29_13:58:36 info: Link brk2:eth0 up.
heartbeat: 2004/04/29_13:58:36 info: pid 2134 locked in memory.
heartbeat: 2004/04/29_13:58:36 info: pid 2135 locked in memory.
heartbeat: 2004/04/29_13:58:36 info: Link 192.168.10.21:192.168.10.21 up.
heartbeat: 2004/04/29_13:58:36 info: Status update for node 192.168.10.21: status ping
heartbeat: 2004/04/29_13:58:36 info: Link brk1:eth0 up.
heartbeat: 2004/04/29_13:58:36 info: Status update for node brk1: status active
heartbeat: 2004/04/29_13:58:36 info: Local status now set to: 'active'
heartbeat: 2004/04/29_13:58:36 info: Starting child client "/usr/lib/heartbeat/ipfail" (90,90)
heartbeat: 2004/04/29_13:58:36 info: Starting "/usr/lib/heartbeat/ipfail" as uid 90 gid 90
(pid 2151)
heartbeat: 2004/04/29_13:58:36 info: remote resource transition completed.
heartbeat: 2004/04/29_13:58:36 info: remote resource transition completed.
heartbeat: 2004/04/29_13:58:36 info: Local Resource acquisition completed. (none)
heartbeat: 2004/04/29_13:58:36 info: brk1 wants to go standby [foreign]
heartbeat: 2004/04/29_13:58:36 info: Running /etc/ha.d/rc.d/status status

```

```
heartbeat: 2004/04/29_13:58:44 info: standby: acquire [foreign] resources from brk1
heartbeat: 2004/04/29_13:58:44 info: acquire local HA resources (standby).
heartbeat: 2004/04/29_13:58:45 info: Acquiring resource group: brk2 IPAddr::192.168.10.202/24
ServeRAID::1::2 Filesystem::-Lbrk2::/var/mqm/brk2::ext3 mqsMod.pl::/var/mqm/brk2::primary
heartbeat: 2004/04/29_13:58:45 info: Running /etc/ha.d/resource.d/IPAddr 192.168.10.202/24
start
heartbeat: 2004/04/29_13:58:49 info: /sbin/ifconfig eth0:0 192.168.10.202 netmask 255.255.255.0
broadcast 192.168.10.255
heartbeat: 2004/04/29_13:58:49 info: Sending Gratuitous Arp for 192.168.10.202 on eth0:0 [eth0]
heartbeat: 2004/04/29_13:58:49 /usr/lib/heartbeat/send_arp -i 500 -r 10 -p
/var/lib/heartbeat/rsctmp/send_arp/send_arp-192.168.10.202 eth0 192.168.10.202 auto
192.168.10.202 ffffffff
heartbeat: 2004/04/29_13:58:51 info: Running /etc/ha.d/resource.d/Filesystem -Lbrk2
/var/mqm/brk2 ext3 start
heartbeat: 2004/04/29_13:58:51 ERROR: Couldn't mount filesystem -Lbrk2 on /var/mqm/brk2
heartbeat: 2004/04/29_13:58:51 ERROR: Return code 1 from /etc/ha.d/resource.d/Filesystem
heartbeat: 2004/04/29_13:58:51 info: Running /etc/ha.d/resource.d/mqsMod.pl /var/mqm/brk2
primary start
heartbeat: 2004/04/29_13:58:55 info: local HA resource acquisition completed (standby).
heartbeat: 2004/04/29_13:58:55 info: Standby resource acquisition done [foreign].
heartbeat: 2004/04/29_13:58:55 info: Initial resource acquisition complete (auto_failback)
heartbeat: 2004/04/29_13:58:55 info: remote resource transition completed.
```

---

After rebooting from a power failure, the `ipssend` utility fails to run properly at first. Consequently, during boot up you will see the errors listed at the bottom of Example 4-24. By the time the system has fully booted the `ipssend` is available for use again.

5. In order to continue the failback process, the scripts from `/etc/ha.d/haresources` need to be started manually.

To start them on `brk2`, use the code in Example 4-25.

*Example 4-25 Manually starting failback on brk2*

---

```
brk2# cd /etc/ha.d/resource.d
brk2# ./ServeRAID 1 2 start
brk2# ./Filesystem -Lbrk2 /var/mqm/brk2 ext3 start
brk2# ./mqsMod.pl /var/mqm/brk2 primary start
```

---

If `brk1` is node recovering from the power loss, the failback steps would be instead as in Example 4-26:

*Example 4-26 Manually starting failback on brk1*

---

```
brk1# cd /etc/ha.d/resource.d
brk1# ./ServeRAID 1 1 start
brk2# ./Filesystem -Lbrk1 /var/mqm/brk1 ext3 start
brk2# ./mqsMod.pl /var/mqm/brk1 primary start
```

---

## 4.8 Summary

In this chapter we demonstrated how a queue manager can be recovered on a secondary server by using a shared-disk configuration. Using our configuration was an enhancement from the original configuration because there was a much smaller wait interval for failover and failback. In addition, the nonfailing queue manager had no IO wait time when the failing server was powered off because the disk-mirroring think-time was no longer a constraint.

Archived

Archived



## Implementing HA queue managers: Part 3

This chapter discusses a WebSphere MQ configuration and two applications that demonstrate implementing a more robust queue manager recovery solution with Linux by using a SAN. Combined with the file system software GPFS, we now have continuous access to the filesystems from both nodes in the cluster.

## 5.1 Scenario three overview

This scenario expands on scenario two by showing the improvements that can be made by replacing the external SCSI enclosure with a low-end SAN, a FASTt200 disk enclosure. Highly scalable and fast, these SANs can be configured to provide hundreds of terabytes of shared storage, but at a much greater cost than in scenario two. The hardware topology used in this scenario will be almost the same as in scenario two, see Figure 4-1 on page 122. The only difference is the SAN switch in Figure 5-1.

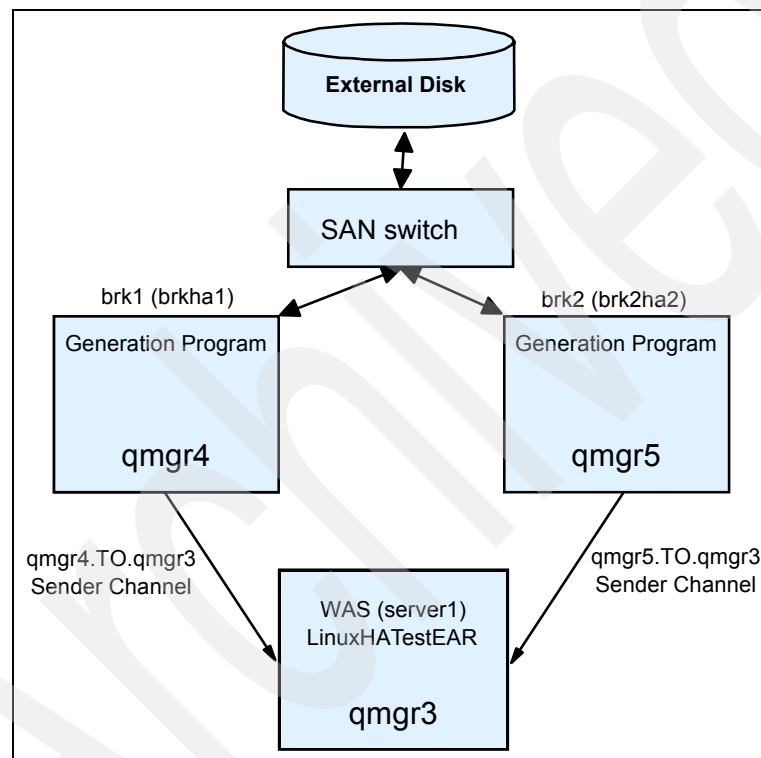


Figure 5-1 High-Availability through SAN

## 5.2 Planning an implementation

This section discusses the additional hardware and software components that are needed to implement this scenario.



## 5.2.1 Hardware setup

We begin our third highly available scenario with a plan very similar to the second. There are, however, two important changes:

1. We will not be using an external SCSI enclosure (IBM EXP200).
2. We use some additional hardware. See Table 5-1.

Table 5-1 lists in detail the hardware components that are required to use the SAN box and to connect it to the two Linux nodes.

Table 5-1 Additional hardware requirements

Quantity	Component
1	IBM FASTT200 disk enclosure
8	Fiber hard disk
2	FastT FC2/133 Host Adapter
1	SANBox2 Fiber Switch
2	SC to SC fiber cables
2	SC to LC fiber cables

## 5.2.2 Cluster components

As in our two previous scenarios, our goal is to have two or more queue managers running on two separate machines. When one machine fails, the other will take over the queue managers of the downed machine. We decided to stay with the use of the heartbeat software from linux-ha.org instead of installing CSM for monitoring cluster resources and reacting on the events.

Figure 5-4 on page 161 shows the layout of the cluster. The shared resources noted in bold are the primary for that node and the grayed resources are secondary.

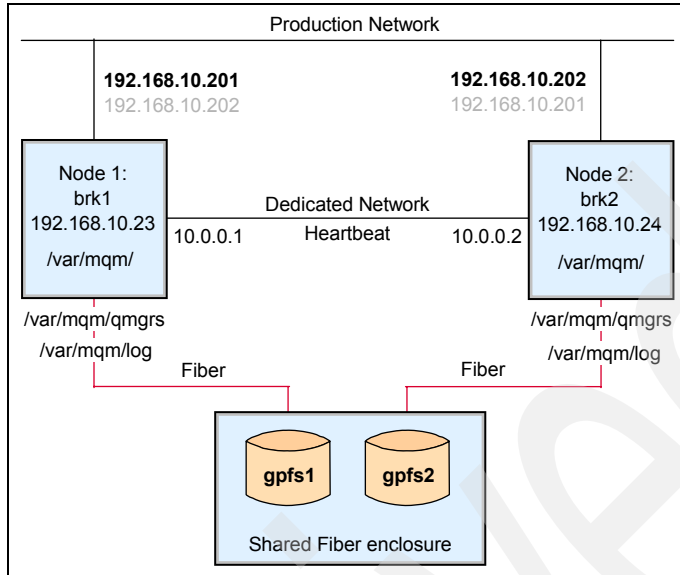


Figure 5-2 brk1 and brk2 in an HA cluster with shared fiber resource

For further information about how to setup this hardware, refer to Appendix C, “Configuring a FAStT200 disk system for Linux” on page 327.

The filesystem GPFS that is used for the file systems on the SAN device, requires two additional software components:

- ▶ System Resource Controller (SRC)
- ▶ IBM Reliable Scalable Cluster Technology (RSCT)

## System Resource Controller

The *System Resource Controller* (SRC) is an AIX service management framework which has been ported to Linux in order to provide consistency between the GPFS implementation on both platforms. The SRC provides a set of commands and subroutines to make it easier for the system manager and programmer to create and control subsystems. A *subsystem* is any program or process or set of programs or processes that is usually capable of operating independently or with a controlling system. A subsystem is designed as a unit to provide a designated function.

The SRC software was designed to minimize the need for operator intervention. It provides a mechanism to control subsystem processes using a common command line and the C programming interface.

## Reliable Scalable Cluster Technology

The *Reliable Scalable Cluster Technology* (RSCT) component provides high-availability services to the system. It includes two subsystems:

- ▶ Topology Services
- ▶ Group Services

Both of these distributed subsystems operate within a peer domain. A *domain* is a set of machines upon which the RSCT components execute and, exclusively of other machines, provide their services.

Topology Services provides high-availability subsystems with network adapter status, node connectivity information, and a reliable messaging service. It also establishes the heartbeat ring. It is started by GPFS and does not need administrative interventions. It consists of the following elements:

- ▶ Topology Services daemon (hatsd)  
This daemon runs on each node in the GPFS cluster and its task is to create the heartbeat ring.
- ▶ Pluggable Network Interface Methods (NIMs)  
NIMs are processes started by the Topology Services daemon to monitor each local adapter. It is responsible for communicating with the peer daemon, sending and monitoring heartbeat and monitoring the local adapter.
- ▶ Ports and sockets  
Ports and sockets provide UDP ports for intercluster communication and UNIX domain sockets for the communication between:
  - Topology Services clients and the local Topology Services daemon
  - Local Topology Services daemon and the NIMs
- ▶ Topology Services commands  
Contains the `cthatsctrl` control command, `cthats` startup command and `thatstune` tuning command.

We are using RSCT exclusively to provide a peer cluster for the purposes of GPFS communication. See Example 5-1 for information provided by the Topology Services daemon.

### Example 5-1 RSCT Topology Services daemon

```
brk1:/var/adm/ras # lssrc -ls cthats
Subsystem      Group      PID      Status
  cthats       cthats     962      active
Network Name  Indx Defd Mbrs St Adapter ID      Group ID
CG1           [ 0]    2    2  S 192.168.10.23  192.168.10.24
CG1           [ 0]  eth0      0x40ace1c7  0x40ace1cc
```

```
HB Interval = 1 secs. Sensitivity = 4 missed beats
Missed HBs: Total: 0 Current group: 0
Packets sent    : 1084566 ICMP 0 Errors: 0 No mbuf: 0
Packets received: 1340515 ICMP 0 Dropped: 0
NIM's PID: 1033
  2 locally connected Clients with PIDs:
  rmcd( 1597) hagsd( 963)
  Configuration Instance = 1083170894
  Default: HB Interval = 1 secs. Sensitivity = 8 missed beats
  Daemon employs no security
  Segments pinned: Text Data Stack.
  Text segment size: 641 KB. Static data segment size: 636 KB.
  Dynamic data segment size: 1348. Number of outstanding malloc: 85
  User time 15 sec. System time 1 sec.
  Number of page faults: 1010. Process swapped out 0 times.
  Number of nodes up: 2. Number of nodes down: 0.
```

---

### 5.2.3 GPFS file systems

Because we are using the GPFS file system, there are some special considerations for the disk setup. In this section, we examine how GPFS saves data to the disks.

A common strategy to increase file system performance is to distribute it across multiple disk controllers and disks. This distribution parallels requests, resulting in reduced latency and higher throughput. This is typically done in hardware, using RAID controllers, or in software, by the logical volume manager (LVM) layer. However, these operate below the file system layer and are, therefore, unaware of the file system makeup. Discrepancies between a file's record size and the volume stripe size or file fragmentation might even result in reduced performance. Instead, GPFS implements striping on a file system level. It divides large files into blocks of equal size and places the consecutive blocks on different disks in round-robin fashion.

To minimize seek overhead, the block sizes are typically large, 256KB for example. However, block sizes can be configured to anything between 16KB and 1MB. A large block size allows you to retrieve a large amount of data in a single I/O from each disk. Smaller files and the end of large files are stored in smaller units, subblocks, about 1/32 of the size of a full block. In order to take full advantage of the parallelism when reading large data files, GPFS prefetches data from a buffer pool and issues simultaneous I/O requests to disks until the capacity of the bandwidth of the switching fabric is saturated. In a similar fashion, writing to the disks happens in parallel.

Because WebSphere MQ's file system for logs and the file system for data have both different file size and different access patterns, it is important to create two separate file systems to get the best performance.

Log files tend to be large in size, up to 64MB, and access to these files is mostly append writes or large sequential reads during recovery situations. As a result, our log file system `/var/mqm/log` should have a large GPFS block size. We have chosen 1MB, the maximum, in order to receive the maximum benefits from the striping. In the lab, we used circular logging for WebSphere MQ, but in real life linear logging is normally required for reasons of data recovery.

Data files are smaller, depending on the size of the message, and access to them is random. Therefore, our data file system `/var/mqm/qmgrs` should have a relatively small block size in order to benefit from striping at all. We have chosen a 256KB block size because we have a small message size. Your implementation might require a larger block size if your messages are larger.

We also need to consider the fact that GPFS does not offer us any protection against data loss. We decided to go for RAID1, which has a disadvantage of using 200% of disk space and overhead with double writes. However it delivers excellent data security and good read performance. See Figure 5-3 on page 158 for our file system design.

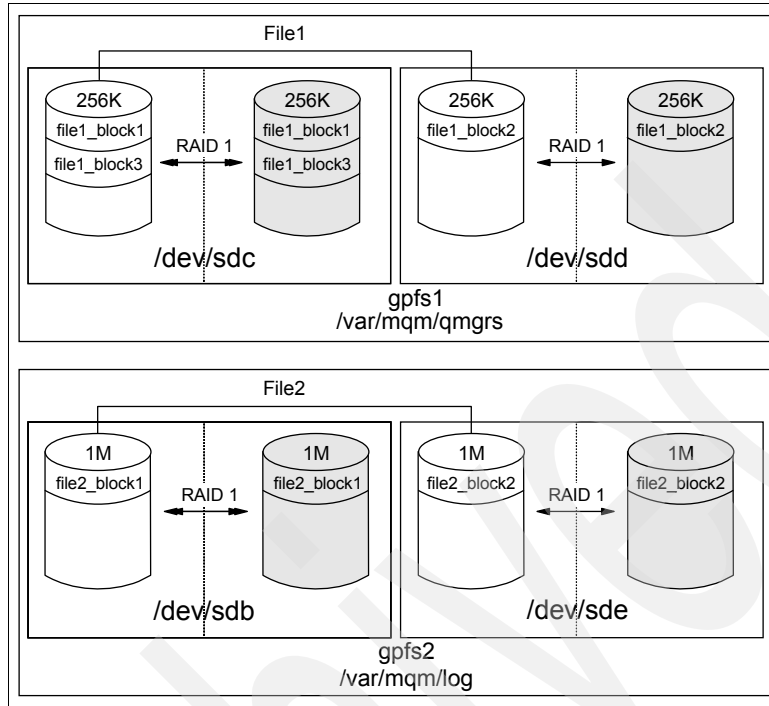


Figure 5-3 RAID1 combined with GPFS striping

## 5.3 Performing an implementation

This section discusses the implementation of all components required for GPFS, GPFS itself, and the setup of WebSphere MQ for use with this configuration.

### 5.3.1 Time synchronization

All nodes should have the same system time to ensure file creation and modification accuracy in the file system, as well as to make it easier to cross-reference information between the nodes during troubleshooting. We installed the Network Time Protocol (NTP) server package for time synchronization in the GPFS cluster. It can be installed from the SUSE installation CDs or downloaded from the SUSE Web site.

1. Perform this task to install the package on nodes brk1, brk2 and WebSphere Application Server:

```
# rpm -ivf ntp-*
```

2. Choose a management server which is part of the solution but not part of the cluster, especially if you are using an external time source and need a connection to the Internet.

Example 5-2 shows the configuration of the NTP server residing on the management node. We chose to use the was node, with reference to an external time source pool.ntp.org. This is not a single system, rather it is a pool of NTP servers maintained by volunteers on the Internet. For more information, see:

<http://www.pool.ntp.org/>

*Example 5-2 NTP configuration file /etc/ntp.conf on management node*

---

```
server pool.ntp.org
driftfile /etc/ntp/drift
broadcastdelay 0.008
authenticate no
```

---

All the cluster servers in turn synchronize their time with the management node. See Example 5-3 for NTP configuration file on node brk1.

*Example 5-3 NTP configuration file /etc/ntp.conf on brk1*

---

```
server was
driftfile /var/lib/ntp/ntp.drift
logfile /var/log/ntp
```

---

3. We also want to create a file /etc/ntp/step-tickers containing the host names of all the systems in the cluster. It is needed to initially set the clock from any of the listed servers before starting up ntpd. If you do not have your /etc/hosts file setup correctly, this may fail if DNS services are not available at the time of the NTP startup. This feature is useful if your hardware clock is not trusted to keep time while your system is down, because synchronization cannot happen if the time on the server is too far off when ntpd is started. See Example 5-4.

*Example 5-4 /etc/ntp/step-tickers*

---

```
was
brk1
brk2
wmq1
wmq2
```

---

4. Start the NTP service on was, brk1 and brk2 by issuing the following commands:

```
# /sbin/chkconfig ntpd on
# service ntpd start
```

## 5.3.2 Secure communications

To implement communication security, perform the following tasks:

1. Ensure that `ssh` is installed on your machines and the `sshd` daemon is running. It is needed for secure cluster communications. You need to be able to authenticate the servers as authorized hosts without the need to provide a password. Perform the following step as root on all the servers to enable this:

```
# ssh-keygen -t rsa -N ""
```

This will generate a private `id_rsa` and public `id_rsa.pub` keypair without password.

2. On `brk1`, run the following commands as root:

```
# cat ~/.ssh/id_rsa.pub > ~/.ssh/authorized_keys
```

3. This command copies the public key of `brk1` into a file `authorized_keys`. In the same way, execute the following command on `brk2`:

```
# ssh brk2 cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Repeat this step for all other nodes, replacing `brk2` with the relevant node name.

4. To change the mode of the file to be readable and writable by root only, execute the following command:

```
# chmod 0600 ~/.ssh/authorized_keys
```

5. Finally, copy the file containing the public keys to `brk2` and to all other nodes.

```
# scp ~/.ssh/authorized_keys brk2:~/.ssh
```

6. Ensure that you can use secure communication between the nodes without being prompted for a password.

## 5.3.3 Support programs for the FASTT device

A number of programs and tools can be installed to assist you in using the FASTT device. Download the latest code from the IBM FASTT Support Web site:

[http://www-1.ibm.com/support/docview.wss?rs=572&uid=psg1MIGR-53891&loc=en\\_US](http://www-1.ibm.com/support/docview.wss?rs=572&uid=psg1MIGR-53891&loc=en_US)

The following components should be available from this Web site:

- ▶ IBM FASTT Management Suite Java (MSJ) Diagnostic and Configuration Utility for Linux
- ▶ FASTT Storage Manager for x86 Linux



## FAST Management suite

*FAST MSJ* is a network-capable application that can connect to and remotely configure fiber channel host adapters in a SAN environment. FAST MSJ uses ONC remote procedure calls (RPC) for remote communication. With this utility you can:

- ▶ View event and error logs
- ▶ Test fail-over scenarios with Failover Watcher
- ▶ Manage configurations
- ▶ Perform adapter functions including (See Figure 5-4):
  - Configure NVRAM settings
  - Perform NVRAM updates
  - Perform flash updates

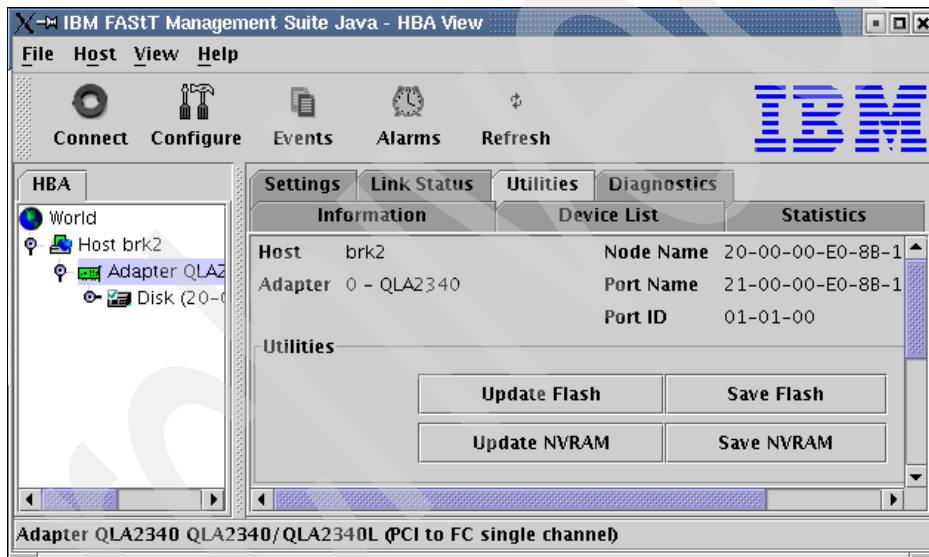


Figure 5-4 Host adapter Utilities tab

This tool can be used to configure the host adapter and change the NVRAM settings. Install the GUI and Linux Agent on both servers.

1. To run the program, issue command `q1remote` to start the agent.
2. Execute `/usr/FAST_MSJ` to launch the GUI.
3. The default password is `config`.

## FAST Storage Manager

The IBM FAST Storage Manager software is used to configure arrays and logical drives, assign your logical drives into storage partitions, replace and

rebuild failed disk drives, expand the size of arrays and logical drives and convert from one RAID level to another.

It also allows you to perform troubleshooting and management tasks, like checking the status of the FASTT Storage Server components, updating the firmware of RAID controllers, and similar activities. Additionally you can configure and manage FlashCopy® Volumes and Remote Volume Mirroring (RVM).

Storage management software provides the following two methods for managing storage subsystems:

- ▶ *Host-agent Management* manages the storage systems through the fiber channel I/O to the host.
- ▶ *Direct management* manages the storage systems over the network through the Ethernet connection.

### ***Configuring the disk arrays***

We created four arrays with two disks in each running RAID1 and selected automatic logical drive to LUN mapping. See Figure 5-5 on page 163. It is good practice to create at least two hot spares to be used between the four disk arrays outside lab conditions.

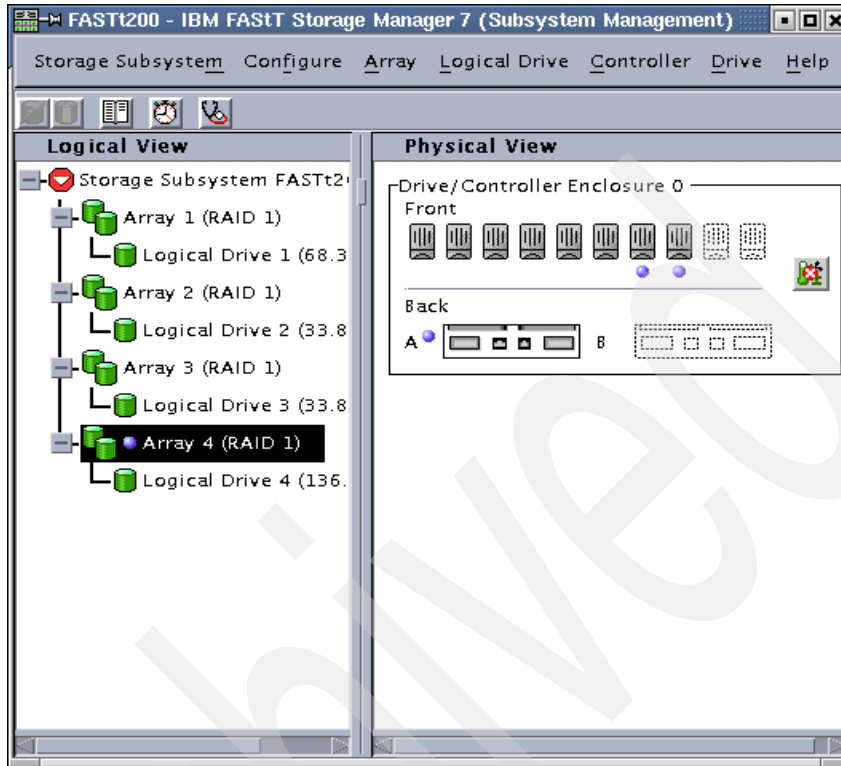


Figure 5-5 Storage Subsystem Management

1. Next we need to create a partition on each of the logical drives. We do this by first checking our disk definitions by issuing the following command:

```
# fdisk -l
```

It will indicate which devices have unused space on them.

On our servers, we created one primary partition on each of our four drives:

- /dev/sdb
- /dev/sdc
- /dev/sdd
- /dev/sde

Examine the final file system setup in Figure 5-6 on page 164. All arrays contain two disks which are mirrored with hardware RAID1. Under Linux these appear as devices /dev/sdb - /dev/sde. We create a primary partition on each, these block devices are respectively called /dev/sdb1 - /dev/sde1. Block devices /dev/sdc and /dev/sdd are to be used for creating the file system gpfs1 and /dev/sdb and /dev/sde to create file system gpfs2. Later we are going to mount these as:

- ▶ gpfs1 - /var/mqm/qmgrs
- ▶ gpfs2 - /var/mqm/log

**Note:** Striping performs best when all the disks are the same size. In a nonuniform disk configuration, a tradeoff between the throughput and space utilization is required. With GPFS, you can choose between maximum data placement capacity and throughput.

To maximize the space utilization, GPFS places more data on the larger disks, causing them to have proportionally more I/O requests, thus reducing the total throughput.

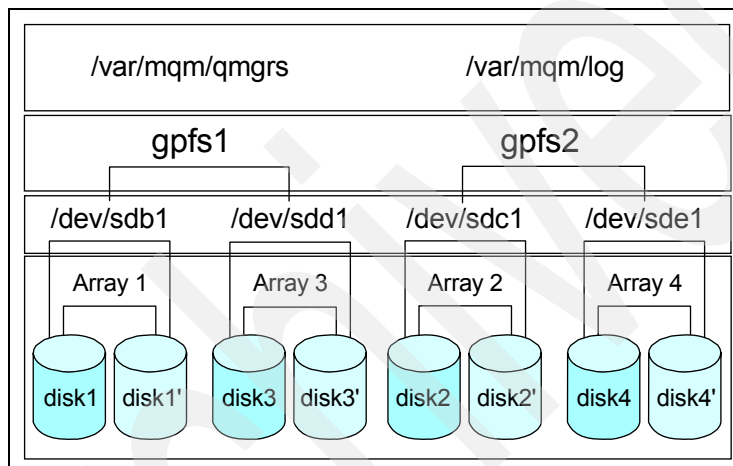


Figure 5-6 File system setup

### 5.3.4 RSCT and GPFS

In this book we go through the steps necessary for our setup. For more information about general GPFS setup, refer to the IBM Redbook *Linux Clustering with CSM and GPFS*, SG24-6601:

<http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/SG246601.html>

#### Installing RSCT and GPFS

Because we do not use CSM or xCAT, we install the SRC and RSCT packages.

1. Make a directory `~/gpfs` on server `brk1` and copy the following RPMs in it:
  - `src-1.2.0.4-0.i386.rpm`
  - `rsct.core.utils-2.3.2.1-0.i386.rpm`
  - `rsct.core-2.3.2.1-0.i386.rpm`

- rsct.basic-2.3.2.1-0.i386.rpm
- gpfs.gpl-2.2.0-0.noarch.rpm
- gpfs.gpl-2.2.0-1.noarch.rpm
- gpfs.base-2.2.0-0.noarch.rpm
- gpfs.base-2.2.0-1.noarch.rpm
- gpfs.msg.en\_US-2.2.0-0.noarch.rpm
- gpfs.msg.en\_US-2.2.0-1.noarch.rpm
- gpfs.docs-2.2.0-0.noarch.rpm
- gpfs.docs-2.2.0-1.noarch.rpm

2. The order in which these RPMs are installed is important, so run the following commands in Example 5-5 as listed to ensure a proper installation:

*Example 5-5 Installing RPMs*

---

```
# rpm -iv src-1.2.0.4-0.i386.rpm
# rpm -iv rsct.core.utils-2.3.2.1-0.i386.rpm
# rpm -iv rsct.core-2.3.2.1-0.i386.rpm
# rpm -iv rsct.basic-2.3.2.1-0.i386.rpm
# rpm -iv gpfs.gpl-2.2.0-0.noarch.rpm gpfs.base-2.2.0-0.noarch.rpm
gpfs.msg.en_US-2.2.0-0.noarch.rpm gpfs.docs-2.2.0-0.noarch.rpm
# rpm -iv gpfs.gpl-2.2.0-1.noarch.rpm gpfs.base-2.2.0-1.noarch.rpm
gpfs.msg.en_US-2.2.0-1.noarch.rpm gpfs.docs-2.2.0-1.noarch.rpm
```

---

3. Next, build the GPFS portability layer on brk1.

a. Modify and export the SHARKCLONEROOT variable:

```
# export SHARKCLONEROOT=/usr/lpp/mmfs/src
```

b. Move this variable to the /usr/lpp/mmfs/src/config directory and make a copy of the site.mcr.proto file:

```
# cd /usr/lpp/mmfs/src/config
# cp site.mcr.proto site.mcr
```

c. Edit the following sections in the file /usr/lpp/mmfs/src/config/site.mcr to indicate your distribution and kernel level. In our case, we made the build process believe that we were running SUSE with kernel 2.4.21.138 instead of United Linux. See Example 5-6:

*Example 5-6 /usr/lpp/mmfs/src/config/site.mcr*

---

```
/* Linux distribution
/* LINUX_DISTRIBUTION = REDHAT_LINUX */
/* LINUX_DISTRIBUTION = REDHAT_AS_LINUX */
LINUX_DISTRIBUTION = SUSE_LINUX
/* LINUX_DISTRIBUTION = KERNEL_ORG_LINUX */
.....
/* Linux kernel version
#define LINUX_KERNEL_VERSION 2042199
```

---

In addition to this we added a symbolic link called SuSE-release pointing to UnitedLinux-release:

```
# ln -s /etc/UnitedLinux-release /etc/SuSE-release
```

**Important:** Read through the file carefully looking for information for your particular distribution. You might have to edit additional parameters in this file, such as KERNEL\_HEADER\_DIR or #define LINUX\_DISTRIBUTION\_LEVEL.

If the last component of your kernel version is larger than 100, specify 99 instead. For example, our kernel version is 2.4.21.138, but we specified kernel version 2042199.

d. Create a customized portability layer to GPFS in the /usr/lpp/mmfs/src directory:

```
# make World
# make InstallImages
```

e. Copy the following files to /usr/lpp/mmfs/bin:

- mmfslinux
- mmfs25
- lxtrace
- tracedev
- dumpconv

f. Repeat starting at step 1 on page 164 for server brk2, with the exception of not installing gpfs.gpl-2.2.0-0.noarch.rpm and gpfs.gpl-2.2.0-1.noarch.rpm, then copy the files listed in step 2e to /usr/lpp/mmfs/bin.

i. RSCCT starts a number of processes which we can examine by issuing the # **lssrc -a** command. See Example 5-7. Notice that at this point the mmfs process, the GPFS filesystem, is still inoperative.

*Example 5-7 RSCCT processes*

---

```
brk1:~ # lssrc -a
Subsystem      Group          PID    Status
IBM.ConfigRM   rsct_rm        631    active
ctcas          rsct           692    active
cthats         cthats         962    active
cthags         cthags         963    active
ctrmc          rsct           1597   active
IBM.ERRM       rsct_rm        1607   active
IBM.AuditRM    rsct_rm        1636   active
mmfs           aixmm          inoperative
IBM.HostRM     rsct_rm        inoperative
```

---

## Creating a GPFS descriptor file

When creating your GPFS cluster, you need to provide a file containing a list of node descriptors, one per line for each node to be included into the cluster, including any storage nodes. Descriptors must be specified in a form:

```
primaryNetworkNodeName::secondaryNetworkNodeName
```

The value `primaryNetworkNodeName` refers to the host name on the primary network used for GPFS daemon communication. The value `secondaryNetworkNodeName` for the host name on the secondary network to be used in the event the first one is merely saturated. It is not necessary to have `secondaryNetworkNodeName` defined. However, in the event of excessive network traffic the RSCT might not be able to communicate with other nodes over the primary network. In that case, the RSCT assumes the other node or nodes failed and starts recovery.

We do not have a secondary network available in our setup. As a result, our descriptor file is as shown in Example 5-8:

*Example 5-8 /tmp/gpfs.allnodes*

---

```
brk1  
brk2
```

---

## Creating a GPFS cluster

Because we are not using CSM we need to manually create a RSCT peer domain.

1. Run the following command on both servers to create a peer domain `fasttdom` containing our servers `brk1` and `brk2`:

```
# preprnode brk1 brk1
```

2. Then on server `brk1` run:

```
# mkrpdomain fasttdom brk1 brk2  
# startdomain fasttdom
```

3. Create a GPFS cluster defining type Linux cluster (`-t`), server `brk1` as our primary repository server (`-p`) and `brk2` as secondary repository (`-s`), `ssh` as secure shell (`-r`) and `scp` as secure copy (`-R`) to be used by the cluster:

```
# mmcrcluster -t lc -p brk1 -s brk2 -n /tmp/gpfs.allnodes -r /usr/bin/ssh  
-R /usr/bin/scp
```

4. For complete command usage and help, run:

```
# mmcrcluster
```

**Attention:** It will take a while for this command to execute as it performs many operations on all the cluster nodes. If any of your nodes are not available when you run this command, you will have to add them later on to the cluster using the `mmaddcluster` command.

## Creating the GPFS nodeset

A *nodeset* is a group of some GPFS nodes in a cluster. You may have different nodesets within the GPFS cluster, but each node has to be defined as part of the cluster and can participate in one nodeset only.

1. Run the `mmconfig` command to create a nodeset containing all or both of our nodes `brk1` and `brk2` (`-a`), with name `fasttset` (`-C`), which will automatically start the GPFS daemon at boot time (`-A`). If no name is defined, GPFS assigns an integer as nodeset name:

```
# mmconfig -a -C fasttset -A
```

2. For complete command usage and help, run:

```
# mmconfig
```

3. You can examine the GPFS cluster we have created by issuing:

```
# mmlscluster
```

See Figure 5-7 for an example of the output of the command.

```
GPFS cluster information
=====
GPFS cluster type:          lc
GPFS cluster id:           gpfs1083171097
RSCT peer domain name:     fasttdom
Remote shell command:      /usr/bin/ssh
Remote file copy command:  /usr/bin/scp

GPFS cluster data repository servers:
-----
Primary server:   brk1.ibm.com
Secondary server: brk2.ibm.com

Nodes in nodeset fasttset:
-----
 1  (1)  brk1      192.168.10.23  brk1.ibm.com
 2  (2)  brk2      192.168.10.24  brk2.ibm.com
```

Figure 5-7 GPFS cluster configuration

4. Command `mmlsconfig` gives us useful information about the nodeset `fasttset` that we just created. See Figure 5-8 on page 169:



```

Configuration data for nodeset fasttset:
-----
clusterType lc
comm_protocol TCP
multinode yes
autoload yes
useSingleNodeQuorum no
useDiskLease yes
group Gpfs.fasttset
recgroup GpfsRec.fasttset
maxFeatureLevelAllowed 700

File systems in nodeset fasttset:
-----
/dev/gpfs1
/dev/gpfs2

```

Figure 5-8 Nodeset configuration

5. Start the GPFS file daemon by issuing:

```
# mmstartup -a
```

You can issue `-a` for all, `-C` for nodeset, or start the file system daemon separately on each node. Later when we have mount points for our GPFS file systems, they are mounted automatically when this command is run.

The `mmshutdown` command will stop the file system daemon and unmount the file system, forcefully if necessary.

6. The GPFS installation will add the service to be started automatically, but we have to manually add a line to `/etc/inittab` to start the RSCT peer domain. Because this service is needed to start the GPFS file system, place this script before the `mmauto` command. See Example 5-9.

Example 5-9 RSCT peer domain autostart in `/etc/inittab`

---

```

# Start the RSCT Peer Domain to enable the communication between the nodes
rsct:3:once:/usr/bin/starttrpdomain fasttdom >/dev/console 2>&1
# GPFS autostart
gpfs:3:once:/usr/lpp/mmfs/bin/mmauto >/dev/console 2>&1

```

---

GPFS does extensive logging and a new file identified with a timestamp, is created in `/var/adm/ras` every time the GPFS file system is started. The link to the current log file is `mmfs.log.latest`. All the mmfs commands including `mmauto` log into this file and it can give us important information in case of a problem.

7. A trace utility is also available for debugging purposes and can be started by issuing:

```
# /usr/lpp/bin/mmtrace
```

This will by default create a trace in /tmp/mmfs directory. After you have recreated the problem, stop the trace by issuing:

```
# /usr/lpp/bin/mmtrace stop
```

**Note:** If the RSCT peer domain is not started when the GPFS start script runs, `mmauto` places itself in a wait state for the next five minutes. If the RSCT peer domain is started during this time, the `mmauto` command starts the GPFS file system. However, if a time-out occurs, the GPFS file system has to be manually started by issuing the `mmstartup` command.

## GPFS disk definitions

Next we need to create a disk descriptor file for each of the failure groups we want to create. As mentioned in “General Parallel File System for Linux” on page 43, GPFS has two different attachment models. In a network attached model, one of two nodes has direct access to the disks serving as storage server and other nodes use the GPFS file system through NSD servers. We are running GPFS on directly attached disks, meaning that all our nodes have direct fiber connection to the disks with a switch. The syntax of the NSD descriptor file depends on your attachment model. For network attached disks the syntax is:

```
DeviceName:PrimaryNSDServer:SecondaryNSDServer:DiskUsage:FailureGroup
```

For a direct attached model, the primary and secondary server fields are to be left empty and the last field must indicate that there is no common point of failure with any other disk in the nodeset. As we discussed earlier, we want to be able to separate our WebSphere MQ data from the WebSphere MQ logs, therefore we want to create two different GPFS file systems. To accomplish this, we created two separate descriptor files.

1. You can examine our GPFS disk definitions as seen in Example 5-10 and Example 5-11 on page 171.

*Example 5-10 /tmp/desc\_gpfs1*

---

```
/dev/sdc::dataAndMetadata:-1  
/dev/sdd::dataAndMetadata:-1
```

---

We define a new line for each of our two logical disks `sdc` and `sdd` we want to use for this failure group. Note that we selected to group these logical drives because they had the same smaller size compared to `sdb` and `sde`. We are going to use this file system for our data and mount it on `/var/mqm/qmgrs`.

*Example 5-11 /tmp/desc\_gpfs2*

---

```
/dev/sdb::/dev/sde::

---


```

2. Our second failure group consists of sdb and sde, which are larger logical drives and are going to be used for our log file and mounted on /var/mqm/log.

Now we create the Network Shared Disks by running the following commands:

```
# mmcrnsd -F /tmp/desc_gpfs1  
# mmcrnsd -F /tmp/desc_gpfs2
```

After the file system is successfully created, the command modifies the descriptor files by commenting out the disk devices, then puts the GPFS assigned global name on the next line. See Example 5-12.

*Example 5-12 /tmp/desc\_gpfs1 after file system creation*

---

```
#/dev/sdc::gpfs7nsd::#/dev/sdd::gpfs8nsd::

---


```

## Creating the GPFS file systems

Next we need to create the GPFS file systems.

1. Run the following commands in order to create file systems gpfs1 and gpfs2:

```
# mmcrfs /var/mqm/qmgrs gpfs1 -F /tmp/desc_gpfs1 -A yes  
# mmcrfs /var/mqm/log gpfs2 -F /tmp/desc_gpfs2 -A yes
```

This command will format the disks, add the following entries to /etc/fstab, and propagate all the changes to other nodes. See Example 5-13.

*Example 5-13 /etc/fstab*

---

```
/dev/gpfs1 /var/mqm/qmgrs gpfs dev=/dev/gpfs1, autostart 0 0  
/dev/gpfs2 /var/mqm/log gpfs dev=/dev/gpfs2, autostart 0 0
```

---

**Attention:** Although GPFS places the file system mount points in `/etc/fstab`, it is not possible to change the mount point of the GPFS file system by editing this file. Next time the GPFS is started, the modified line is replaced by the mount point given when the `mmcrfs` command was issued. Hence changes to the mount points have to be done with the help of the `mmchfs` command. For example to change the mount point for `gpfs1` to `/var/mqm/test/qmgrs` issue the following command:

```
# mmchfs gpfs1 -T /var/mqm/test/qmgrs
```

Now we can examine the file systems created by issuing the `mm1nsd` command. See Figure 5-9 for output.

File system	Disk name	Primary node
gpfs1	gpfs7nsd	(directly attached)
gpfs1	gpfs8nsd	(directly attached)
gpfs2	gpfs9nsd	(directly attached)
gpfs2	gpfs10nsd	(directly attached)

Figure 5-9 NSD setup

2. If we want to closer examine file system `gpfs1` we can issue the command:

```
# mm1sfs gpfs1
```

See Figure 5-10 on page 173 for output.

flag	value	description
-s	roundRobin	Stripe method
-f	8192	Minimum fragment size in bytes
-i	512	Inode size in bytes
-I	16384	Indirect block size in bytes
-m	1	Default number of metadata replicas
-M	1	Maximum number of metadata replicas
-r	1	Default number of data replicas
-R	1	Maximum number of data replicas
-D	posix	Do DENY WRITE/ALL locks block NFS writes(cifs)
-a	1048576	Estimated average file size
-n	32	Estimated number of nodes that will mount file
-B	262144	Block size
-Q	none	Quotas enforced
-F	105472	Maximum number of inodes
-V	7.00	File system version. Highest supported version;
-z	no	Is DMAPI enabled?
-d	gpfs7nsd;gpfs8nsd	Disks in file system
-A	yes	Automatic mount option
-C	fasttset	GPFS nodeset identifier
-E	no	Exact mtime default mount option
-S	no	Suppress atime default mount option
-o	none	Additional mount options

Figure 5-10 gpfs1 file system

3. We can also issue the command `mm1sdisk gpfs1` to see which logical disks belong to this file system. See Figure 5-11 for the output.

disk name	driver type	sector size	failure group	holds metadata	holds data	status
gpfs7nsd	nsd	512	-1	yes	yes	ready
gpfs8nsd	nsd	512	-1	yes	yes	ready

Figure 5-11 Logical disks in file system gpfs1

1. We need to now either:
  - Manually mount the file systems by issuing:
 

```
# mount /var/mqm/qmgrs
# mount /var/mqm/log
```
  - Restart the gpfs file system by issuing:
 

```
# mmshutdown -a
# mmstartup -a
```

2. At this point it might be useful to examine our mount points to ensure our WebSphere MQ file system is as we planned. See Figure 5-12 for the output of the `mount` command run on server `brk1`.

```
/dev/sda2 on / type reiserfs (rw)
proc on /proc type proc (rw)
devpts on /dev/pts type devpts (rw,mode=0620,gid=5)
/dev/sda1 on /boot type ext2 (rw)
shmfs on /dev/shm type shm (rw)
usbdevfs on /proc/bus/usb type usbdevfs (rw)
/dev/sda11 on /var/mqm/brk1 type ext3 (rw)
/dev/sda11 on /var/mqm/brk2 type ext3 (rw)
/dev/gpfs2 on /var/mqm/log type gpfs (rw,dev=/dev/gpfs2,autostart)
/dev/gpfs1 on /var/mqm/qmgrs type gpfs (rw,dev=/dev/gpfs1,autostart)
```

Figure 5-12 File Systems mounted on server `brk1`

### 5.3.5 Configuring WebSphere MQ

Because we already have WebSphere MQ installed on servers `brk1` and `brk2` we will not perform a new installation. However it is unlikely that outside a lab a system with such a configuration would be installed. For new WebSphere MQ installation perform the steps as described in 3.3, “Installing and configuring WebSphere MQ” on page 68.

For our scenario we used the existing WebSphere MQ setup with the following changes:

1. We changed the ownership of the GPFS file systems to `mqm` and made sure that group `mqm` had write access to these file systems.

```
# chown mqm:mqm /var/mqm/log
# chown mqm:mqm /var/mqm/qmgrs
# chmod g+w /var/mqm/log
# chmod g+w /var/mqm/qmgrs
```
2. We then simply copied the content of `/var/mqm/brk1/qmgrs` directory to `/var/mqm/qmgrs`. We deleted the `qmgr` entries from the `/var/mqm/mqs.ini` to reflect our new directory structure and recreated the `qmgr4` and `qmgr5`. See Figure 5-13 on page 175:

```

DefaultPrefix=/var/mqm

ClientExitPath:
  ExitsDefaultPath=/var/mqm/exits

LogDefaults:
  LogPrimaryFiles=3
  LogSecondaryFiles=2
  LogFilePages=1024
  LogType=CIRCULAR
  LogBufferPages=0
  LogDefaultPath=/var/mqm/log

QueueManager:
  Name=qmgr5
  Prefix=/var/mqm
  Directory=qmgr5

```

Figure 5-13 /var/mqm/mqs.ini on server brk2

### 5.3.6 Message generating application

The message generating program used in this scenario is the same application used in the first scenario. See 3.4.1, “Design of the message generation application” on page 71 and 3.4.2, “Developing the message generation application” on page 71. No changes are required to the application.

### 5.3.7 Compiling and running the message generation application

Follow the instructions in scenario two in 4.4.1, “Compiling and running the message generation application” on page 134

### 5.3.8 Messaging retrieval application

The message retrieval application is same as the one used in the first scenario. Refer to 3.5, “Message retrieval application” on page 73.

### 5.3.9 Adding Linux high availability

Under normal operations, qmgr4 will be running on brk1 and qmgr5 will be running on brk2. GPFS file systems will be mounted from the FASTt200 device as illustrated in the Figure 5-14 on page 176.

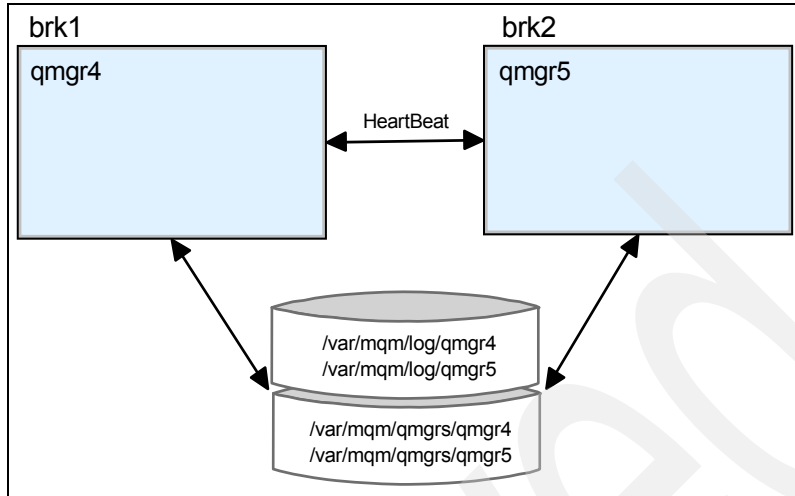


Figure 5-14 Scenario three under normal operation

For WebSphere MQ high availability to work in this scenario, the following configurations are required on both `brk1` and `brk2`.

1. We will again be using the MQ scripts that were discussed earlier in “Shell scripts to manage MQ resources” on page 80. Make these scripts available on the hosts `brk1` and `brk2`. In addition, copy the script `mqsMod.pl` to `/etc/ha.d/resource.d`.
2. Because the heartbeat software is already installed and configured on the hosts `brk1` and `brk2` for scenario two, we only need to make slight modifications to the setup. Refer to 4.2.3, “Configuring Linux-HA” on page 125 for complete configuration of `linux-ha`.
3. Modify the `etc/ha.d/haresources` file on both `brk1` and `brk2` to include the start and stop for the queue managers. Also there is no need to include the ServeRAID failover script (see Example 4-9 on page 136) because both machines have simultaneous access to the shared fiber enclosure. See Example 5-14 for the final version of this configuration file.

Example 5-14 `/etc/ha.d/haresources`

---

```
brk1    IPaddr::192.168.10.201/24 mqsMod.pl::/var/mqm::primary
brk2    IPaddr::192.168.10.202/24 mqsMod.pl::/var/mqm::secondary
```

---

### 5.3.10 HA configuration summary

In summary, our high availability configuration works as shown in Table 5-2 on page 177.



Table 5-2 Summary of HA Configuration

Node(s) Up:	IP Addresses:	Mounted Filesystems:
brk1 and brk2	brk1 - 192.168.10.101 brk2 - 192.168.10.102	brk1 - /var/mqm/qmgrs brk1 - /var/mqm/log brk2 - /var/mqm/qmgrs brk2 - /var/mqm/log
brk1 only	brk1 - 192.168.10.101 brk1 - 192.168.10.102	brk1 - /var/mqm/qmgrs brk1 - /var/mqm/log
brk2 only	brk2 - 192.168.10.102 brk2 - 192.168.10.101	brk2 - /var/mqm/qmgrs brk2 - /var/mqm/log

## 5.4 Running the shared-disk test

We are using the same test as in scenario two. Refer to 4.7, “Running the shared-disk test” on page 136.

In this section, we examine the differences in a failover scenario between the shared SCSI disk enclosure and shared fiber enclosure setups.

If brk2 node fails, brk1 starts a takeover of its function, that is running the queue manager qmgr5. See Figure 5-15 on page 178. As both brk1 and brk2 have simultaneous access to the disks, the file system is already available to brk1 and no disk takeover is needed as in the scenario with the shared SCSI enclosure.

In a failback situation, when brk2 recovers, the host brk2 first establishes access to the disks at boot time then takes over the task of running the qmgr5, without the need to unmount or mount file systems. Because both the nodes have a connection to the shared disks, both failback and failover situation become marginally faster than with shared SCSI disks.

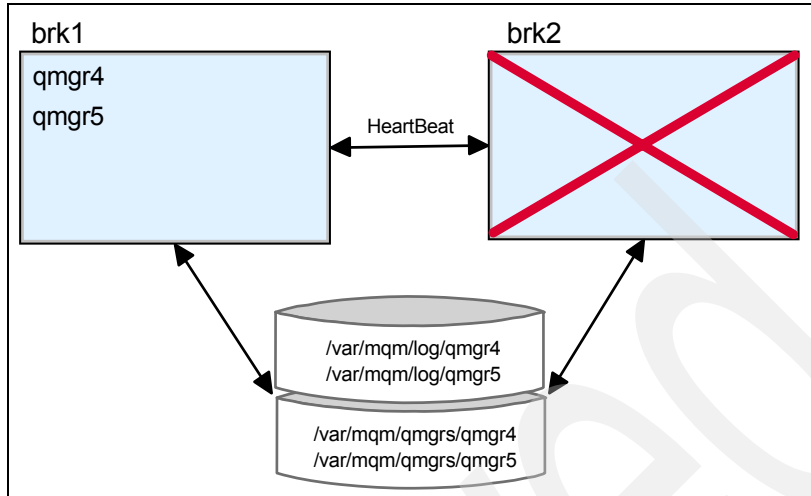


Figure 5-15 Scenario three in failover



## Using WebSphere MQ clustering

This chapter discusses a configuration that demonstrates the implementation of a high-availability solution through the use of WebSphere MQ clustering.

## 6.1 Scenario four overview

In this scenario, load-balancing and failover is done using WebSphere MQ clustering. The operating system and hardware configurations are the same as those used in scenarios one and two. Review Chapter 3, “Implementing HA queue managers: Part 1” on page 51 and Chapter 4, “Implementing HA queue managers: Part 2” on page 121.

### 6.1.1 Using WebSphere MQ clustering for high-availability

The WebSphere MQ cluster topology depicted in Figure 6-1 is used to demonstrate how WebSphere MQ cluster can provide workload balancing and high availability. The cluster consists of five queue managers, two of them created as the cluster's *full repositories*. Message generators will be putting messages to a queue cq with queue managers qmgr1 and qmgr2 who do not have an instance of the cluster queue cq. The queue managers will send the messages in a round-robin pattern to any available instance of the clustered queue on the queue managers qmgr4 and qmgr5.

**Note:** The round-robin pattern can only become apparent if queues are opened or created in such a way that messages, sent by a single program instance, are spread out over the available instances of a cluster queue. In other words, you can create or use a cluster queue in such a way that all messages, created using the same queue handle, are all sent to the same cluster queue. The selection of a cluster queue instance can be done for each MQPUT or only once at MQOPEN time.

The cluster queue cq is actually a remote queue pointing to a local queue on qmgr3. This local queue is serviced by the Web-based monitoring application that we have used before.

This design allows message receivers to retrieve all messages as long as one clustered queue is still available. In other words, as long as one of the queue managers qmgr4 and qmgr5 are available. If either qmgr4 or qmgr5 is not available, the sending queue managers qmgr1 and qmgr2 will only send their messages to the remaining queue manager.

While this design might look complicated, it is actually a first step to building a broker solution. The queue managers qmgr4 and qmgr5 will be used later in this redbook as the queue managers associated with message brokers. The cluster queue cq will become an input queue to a message flow that will manipulate the incoming messages. Refer to Chapter 7, “Implementing highly available brokers” on page 199 for more details about this multibroker setup.

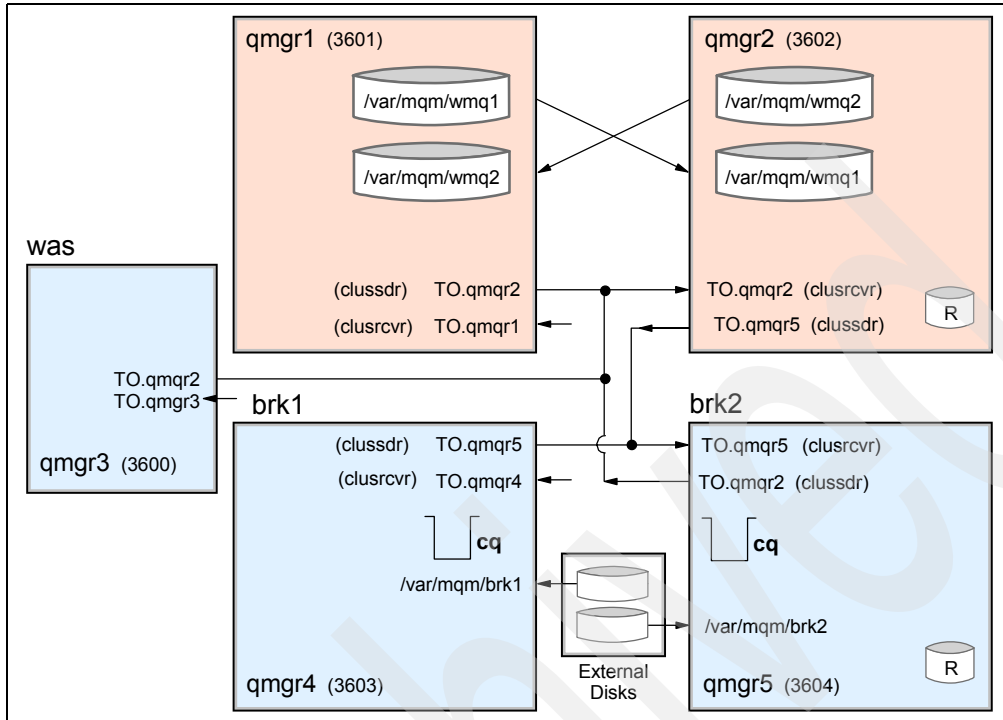


Figure 6-1 WebSphere MQ Cluster topology example

## 6.2 Introduction to WebSphere MQ Cluster

To simplify intercommunication and achieve workload balancing, two or more queue managers on one or more servers can be joined together using WebSphere MQ clustering.

WebSphere MQ clustering on Linux is fully supported. Joining a Linux-based queue manager with queue managers on other WebSphere MQ clustering supported operating systems is also permitted.

### 6.2.1 WebSphere MQ clusters benefits

The two major reasons for using WebSphere MQ clusters are:

1. Reduced system administration.

As soon as you establish even a small cluster, you will benefit from simplified system administration. Establishing a network of queue managers in a cluster involves fewer definitions than establishing a network that is to use standard distributed queuing. With fewer definitions to make, you can set up or change your network more quickly and easily, and reduce the risk of making an error in your definitions. Figure 6-2 on page 183 shows how complex queue manager intercommunication can be without the use of clustering. Six pairs of sender receiver channels and transmit queues would have to be created.

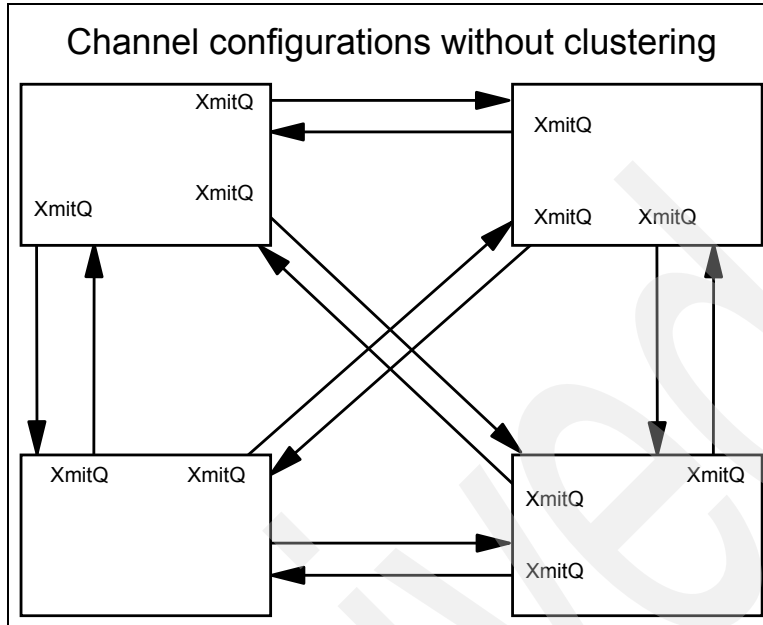
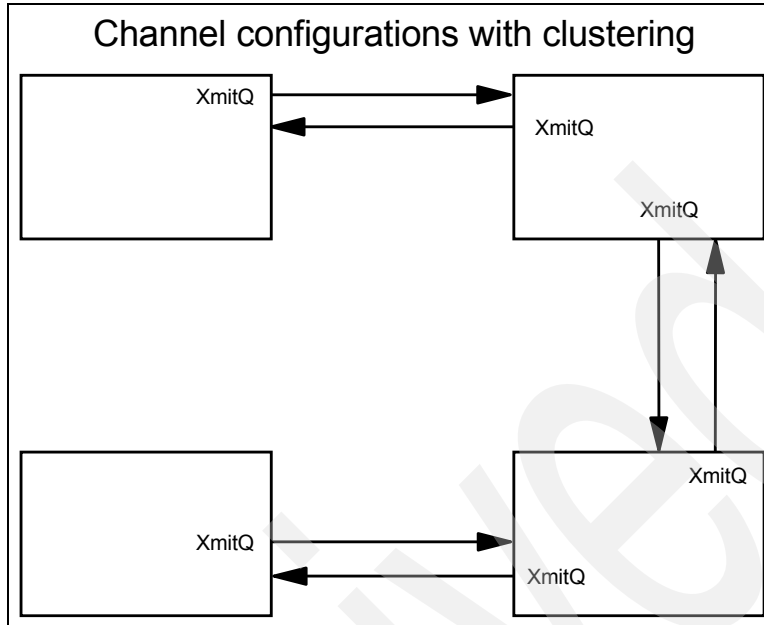


Figure 6-2 A four interconnected queue managers without clustering

Figure 6-3 on page 184 illustrates the same four queue managers would need only three pairs of cluster channels and no transmit queues. This obviously makes configuration much less complex.

The connections that are shown in Figure 6-2 but are not shown in Figure 6-3 will be created by WebSphere MQ itself. Thus, Figure 6-3 does not imply that messages have to hop over one or more queue managers to reach their destination. It only shows what objects have to be defined *manually*. All other objects are defined automatically.



*Figure 6-3 Four interconnected queue managers with clustering*

Simple clusters give you easier system administration. Moving to more complicated clusters offers improved scalability of the number of instances of a queue you can define, providing continuous operations. Because you can define instances of the same queue on more than one queue manager, the workload can be distributed throughout the queue managers in a cluster. Workload balancing is illustrated with Figure 6-4 on page 185.



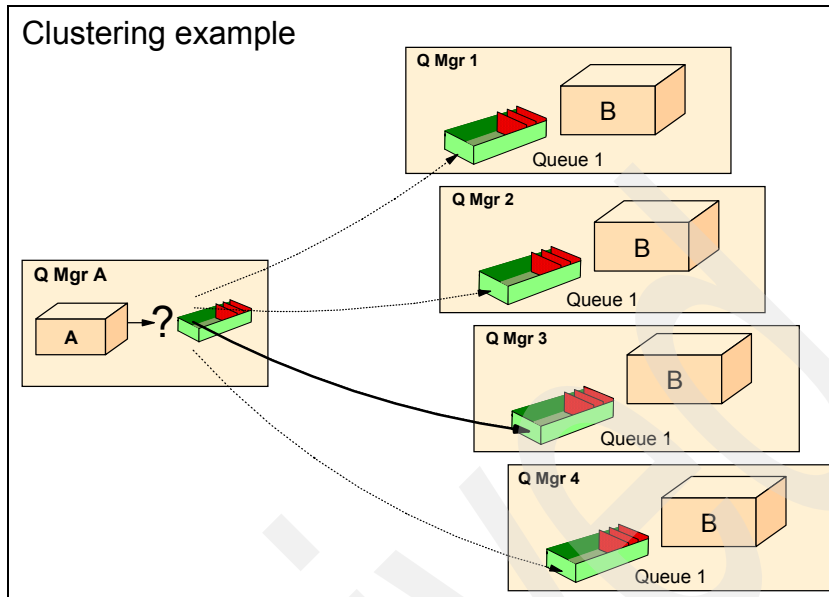


Figure 6-4 Clustering example

Continuous operation and workload balancing is discussed in detail in the remainder of this chapter.

## 6.2.2 WebSphere MQ cluster terminology

The major components that allow clusters to be created are:

- ▶ A *cluster repository* contains information about the cluster. There are two types of cluster repositories:
  - A *full* repository contains all information about the cluster.
  - A *partial* repository contains only information about the cluster that has been previously requested.
- ▶ Cluster channels
  - Each queue manager in a cluster has a cluster receiver channel. A *cluster receiver channel* acts as a template that allows other queue managers to connect to this queue manager.
  - Each queue manager has a *cluster sender channel*. It points to a full repository queue manager in the cluster.
  - Once these channels are defined, no other sender and receiver channels need to be created if the remote queue managers are also a member of the cluster.

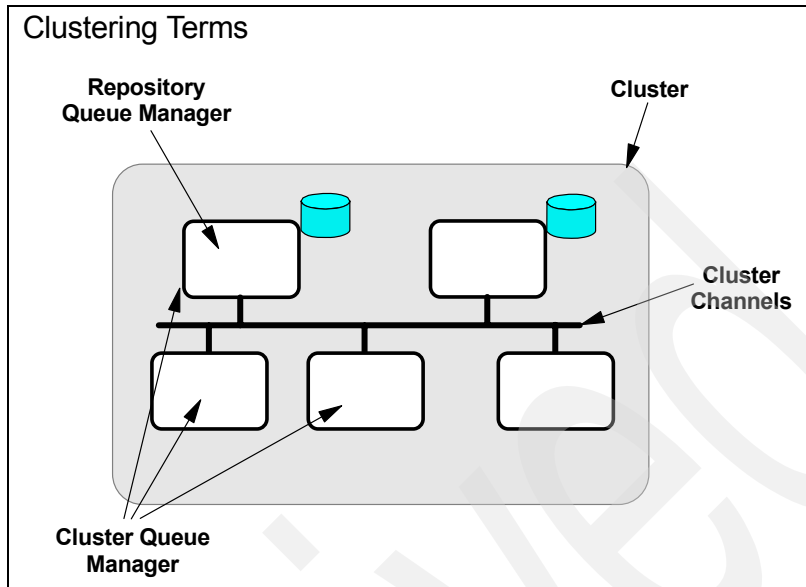


Figure 6-5 Clustering terminology

Further information can be found in the IBM publication *WebSphere MQ Queue Manager Clusters*, SC34-6061-02.

## 6.3 Installing and configuring WebSphere MQ

Because we already have WebSphere MQ installed on all the servers, we will not perform a new installation. For new WebSphere MQ installation, perform the steps as described in 3.3, “Installing and configuring WebSphere MQ” on page 68. No additional components are needed to implement WebSphere MQ clusters.

## 6.4 Message generation application

The message generating program used in this scenario is the same application used in the first scenario. See 3.4.1, “Design of the message generation application” on page 71 and 3.4.2, “Developing the message generation application” on page 71. No changes are required to the application.

## 6.5 Messaging retrieval application

The message retrieval application is same as the one used in the first scenario. Refer to 3.5, “Message retrieval application” on page 73.

## 6.6 WebSphere MQ cluster setup

If the queue managers qmgr1, qmgr2, qmgr3, qmgr4 and qmgr5 have not been created, please complete the steps outlined in 3.9, “Running the Bindings mode test” on page 86 and 4.7, “Running the shared-disk test” on page 136. Additional configuration will be required to join the existing queue managers to the cluster.

### 6.6.1 Defining the full repository queue managers

Select two queue managers to be a full repository queue manager. The first one is qmgr2, which can be considered as a *sending* queue or *backoffice* queue manager. The second full repository queue manager is qmgr5, which can be considered as the *intermediate* or *processing* queue manager.

#### Setup on qmgr2

Make qmgr2 as one of the WebSphere MQ cluster repository managers by creating a cluster sender channel to the other repository manager qmgr5. In addition, a cluster receiver channel is also required.

- ▶ Run `runmqsc qmgr2 < qmgr2_cluster.mqsc`. The `qmgr2_cluster.mqsc` file can be created from Example 6-1.

*Example 6-1 qmgr2\_cluster.mqsc*

---

```
alter qmgr repos(itso)

define channel('TO.qmgr2') +
  chltype(CLUSRCVR) +
  trptype(tcp) +
  conname('wmq2(3602)') +
  cluster(itso) +
  descr('Cluster Receiver channel for qmgr2')
```

```

define channel('T0.qmgr5') +
    chltype(CLUSSDR) +
    trptype(tcp) +
    conname('brk2(3604)') +
    cluster(itso) +
    descr('Cluster sender for respository qmgr qmgr5')

```

---

## Setup on qmgr5

Make qmgr5 one of the WebSphere MQ cluster repository managers by creating a cluster sender channel to the other repository manager qmgr2. In addition, a cluster receiver channel and a cluster queue is also required.

- ▶ Run the command `runmqsc qmgr5 < qmgr5_cluster.mqsc`. The qmgr5\_cluster.mqsc file can be created from Example 6-2.

*Example 6-2 qmgr5\_cluster.mqsc*

---

```

alter qmgr repos(itso)

define channel('T0.qmgr5') +
    chltype(CLUSRCVR) +
    trptype(tcp) +
    conname('brk2(3604)') +
    cluster(itso) +
    descr('Cluster Receiver channel for qmgr5')

define channel('T0.qmgr2') +
    chltype(CLUSSDR) +
    trptype(tcp) +
    conname('wmq2(3602)') +
    cluster(itso) +
    descr('Cluster sender for respository qmgr qmgr2')

DEFINE QREMOTE('cq') +
    RNAME('q1') +
    RQMNAME('qmgr3') +
    XMITQ('qmgr3') +
    cluster(itso) +
    defbind(notfixed) +
    DEFPSIST(YES)

```

---

## 6.6.2 Defining the cluster partial repository queue managers

In this section we define the partial repository queue managers.

## Setup on qmgr1

Create a cluster sender channel to the cluster full repository queue manager qmgr2, and a cluster receiver channel by running the command: **runmqsc qmgr1 < qmgr\_cluster.mqsc**. See Example 6-3.

*Example 6-3 qmgr1\_cluster.mqsc*

---

```
define channel('TO.qmgr1') +
  chltype(CLUSRCVR) +
  trptype(tcp) +
  conname('wmq1(3601)') +
  cluster(itso) +
  descr('Cluster Receiver channel for qmgr1')

define channel('TO.qmgr2') +
  chltype(CLUSSDR) +
  trptype(tcp) +
  conname('wmq2(3602)') +
  cluster(itso) +
  descr('Cluster sender for respository qmgr qmgr2')
```

---

## Setup on qmgr3

Create a cluster sender channel to the cluster full repository queue manager qmgr2, and a cluster receiver channel by running the command **runmqsc qmgr3 < qmgr3\_cluster.mqsc**. See Example 6-4.

*Example 6-4 qmgr3\_cluster.mqsc*

---

```
define channel('TO.qmgr3') +
  chltype(CLUSRCVR) +
  trptype(tcp) +
  conname('was(3600)') +
  cluster(itso) +
  descr('Cluster Receiver channel for qmgr3')

define channel('TO.qmgr2') +
  chltype(CLUSSDR) +
  trptype(tcp) +
  conname('wmq2(3602)') +
  cluster(itso) +
  descr('Cluster sender for respository qmgr qmgr2')
```

---

## Setup on qmgr4

Create a cluster sender channel to the cluster full repository queue manager qmgr5, and a cluster receiver channel by running the command **runmqsc qmgr4 < qmgr4\_cluster.mqsc**. See Example 6-5 on page 190.

*Example 6-5 qmgr4\_cluster.mqsc*

---

```
define channel('TO.qmgr4') +
  chltype(CLUSRCVR) +
  trptype(tcp) +
  conname('brk1(3603)') +
  cluster(itso) +
  descr('Cluster Receiver channel for qmgr4')

define channel('TO.qmgr5') +
  chltype(CLUSSDR) +
  trptype(tcp) +
  conname('brk2(3604)') +
  cluster(itso) +
  descr('Cluster sender for respository qmgr qmgr5')

DEFINE QREMOTE('cq') +
  RNAME('q1') +
  RQMNAME('qmgr3') +
  XMITQ('qmgr3') +
  cluster(itso) +
  defbind(notfixed) +
  DEFPSIST(YES)
```

---

### 6.6.3 Configuration verification

Cluster channels starts automatically and hence the status of the channels on all the queue managers will be in running state once they are defined. If the setup is correct, the status of the cluster channels on the clustered queue managers will be similar to Example 6-6, through Example 6-10 on page 193.

*Example 6-6 Checking status of cluster channels on qmgr1*

---

```
[root@wmq1 mqcluster]# runmqsc qmgr1
5724-B41 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.
Starting MQSC for queue manager qmgr1.

dis chs(*)
  1 : dis chs(*)
AMQ8417: Display Channel Status details.
  CHANNEL(TO.qmgr5)                XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
  CONNAME(brk2(3604))              CURRENT
  CHLTYPE(CLUSSDR)                 STATUS(RUNNING)
  RQMNAME(qmgr5)
AMQ8417: Display Channel Status details.
  CHANNEL(TO.qmgr2)                XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
  CONNAME(wmq2(3602))              CURRENT
```

```

        CHLTYPE(CLUSSDR)                STATUS(RUNNING)
        RQMNAME(qmgr2)
AMQ8417: Display Channel Status details.
        CHANNEL(T0.qmgr1)                XMITQ( )
        CONNAME(192.168.10.24)          CURRENT
        CHLTYPE(CLUSRCVR)              STATUS(RUNNING)
        RQMNAME(qmgr5)
AMQ8417: Display Channel Status details.
        CHANNEL(T0.qmgr1)                XMITQ( )
        CONNAME(192.168.10.22)          CURRENT
        CHLTYPE(CLUSRCVR)              STATUS(RUNNING)
        RQMNAME(qmgr2)

```

---

*Example 6-7 Checking status of cluster channels on qmgr2*

---

```

[root@wmq2 mqcluster]# runmqsc qmgr2
5724-B41 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.
Starting MQSC for queue manager qmgr2.

```

```

dis chs(*)
  1 : dis chs(*)
AMQ8417: Display Channel Status details.
        CHANNEL(T0.qmgr5)                XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
        CONNAME(brk2(3604))              CURRENT
        CHLTYPE(CLUSSDR)                STATUS(RUNNING)
        RQMNAME(qmgr5)
AMQ8417: Display Channel Status details.
        CHANNEL(T0.qmgr2)                XMITQ( )
        CONNAME(192.168.10.24)          CURRENT
        CHLTYPE(CLUSRCVR)              STATUS(RUNNING)
        RQMNAME(qmgr5)
AMQ8417: Display Channel Status details.
        CHANNEL(T0.qmgr2)                XMITQ( )
        CONNAME(192.168.10.20)          CURRENT
        CHLTYPE(CLUSRCVR)              STATUS(RUNNING)
        RQMNAME(qmgr1)
AMQ8417: Display Channel Status details.
        CHANNEL(T0.qmgr1)                XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
        CONNAME(wmq1(3601))              CURRENT
        CHLTYPE(CLUSSDR)                STATUS(RUNNING)
        RQMNAME(qmgr1)
AMQ8417: Display Channel Status details.
        CHANNEL(T0.qmgr4)                XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
        CONNAME(brk1(3603))              CURRENT
        CHLTYPE(CLUSSDR)                STATUS(RUNNING)
        RQMNAME(qmgr4)
AMQ8417: Display Channel Status details.
        CHANNEL(T0.qmgr2)                XMITQ( )

```

```

CONNAME(192.168.10.21)          CURRENT
CHLTYPE(CLUSRCVR)              STATUS(RUNNING)
RQMNAME(qmgr3)
AMQ8417: Display Channel Status details.
CHANNEL(T0.qmgr3)              XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
CONNAME(was(3600))            CURRENT
CHLTYPE(CLUSSDR)              STATUS(RUNNING)
RQMNAME(qmgr3)

```

---

*Example 6-8 Checking status of cluster channels on qmgr3*

---

```

[root@was mqcluster]# runmqsc qmgr3
5724-B41 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.
Starting MQSC for queue manager qmgr3.

```

```

dis chs(*)
  1 : dis chs(*)
AMQ8417: Display Channel Status details.
CHANNEL(T0.qmgr2)              XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
CONNAME(wmq2(3602))            CURRENT
CHLTYPE(CLUSSDR)              STATUS(RUNNING)
RQMNAME(qmgr2)
AMQ8417: Display Channel Status details.
CHANNEL(T0.qmgr3)              XMITQ( )
CONNAME(192.168.10.22)        CURRENT
CHLTYPE(CLUSRCVR)            STATUS(RUNNING)
RQMNAME(qmgr2)
AMQ8417: Display Channel Status details.
CHANNEL(T0.qmgr5)              XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
CONNAME(brk2(3604))            CURRENT
CHLTYPE(CLUSSDR)              STATUS(RUNNING)
RQMNAME(qmgr5)
AMQ8417: Display Channel Status details.
CHANNEL(T0.qmgr3)              XMITQ( )
CONNAME(192.168.10.24)        CURRENT
CHLTYPE(CLUSRCVR)            STATUS(RUNNING)
RQMNAME(qmgr5)

```

---

*Example 6-9 Checking status of cluster channels on qmgr4*

---

```

[root@was mqcluster]# runmqsc qmgr4
5724-B41 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.
Starting MQSC for queue manager qmgr4.

```

```

dis chs(*)
  5 : dis chs(*)
AMQ8417: Display Channel Status details.
CHANNEL(T0.qmgr5)              XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)

```



```

CONNAME(brk2(3604))          CURRENT
CHLTYPE(CLUSSDR)            STATUS(RUNNING)
RQMNAME(qmgr5)
AMQ8417: Display Channel Status details.
CHANNEL(TO.qmgr2)           XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
CONNAME(wmq2(3602))         CURRENT
CHLTYPE(CLUSSDR)           STATUS(RUNNING)
RQMNAME(qmgr2)
AMQ8417: Display Channel Status details.
CHANNEL(TO.qmgr4)           XMITQ( )
CONNAME(192.168.10.22)     CURRENT
CHLTYPE(CLUSRCVR)         STATUS(RUNNING)
RQMNAME(qmgr2)
AMQ8417: Display Channel Status details.
CHANNEL(TO.qmgr4)           XMITQ( )
CONNAME(192.168.10.24)     CURRENT
CHLTYPE(CLUSRCVR)         STATUS(RUNNING)
RQMNAME(qmgr5)

```

---

*Example 6-10 Checking status of cluster channels on qmgr5*

---

```

brk2:/home/mqm/mqcluster # runmqsc qmgr5
5724-B41 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.
Starting MQSC for queue manager qmgr5.

```

```

dis chs(*)
  1 : dis chs(*)
AMQ8417: Display Channel Status details.
CHANNEL(TO.qmgr2)           XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
CONNAME(wmq2(3602))         CURRENT
CHLTYPE(CLUSSDR)           STATUS(RUNNING)
RQMNAME(qmgr2)
AMQ8417: Display Channel Status details.
CHANNEL(TO.qmgr5)           XMITQ( )
CONNAME(192.168.10.22)     CURRENT
CHLTYPE(CLUSRCVR)         STATUS(RUNNING)
RQMNAME(qmgr2)
AMQ8417: Display Channel Status details.
CHANNEL(TO.qmgr4)           XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
CONNAME(brk1(3603))         CURRENT
CHLTYPE(CLUSSDR)           STATUS(RUNNING)
RQMNAME(qmgr4)
AMQ8417: Display Channel Status details.
CHANNEL(TO.qmgr5)           XMITQ( )
CONNAME(192.168.10.21)     CURRENT
CHLTYPE(CLUSRCVR)         STATUS(RUNNING)
RQMNAME(qmgr3)
AMQ8417: Display Channel Status details.

```

CHANNEL(T0.qmgr3)	XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
CONNAME(was(3600))	CURRENT
CHLTYPE(CLUSSDR)	STATUS(RUNNING)
RQMNAME(qmgr3)	
AMQ8417: Display Channel Status details.	
CHANNEL(T0.qmgr5)	XMITQ( )
CONNAME(192.168.10.20)	CURRENT
CHLTYPE(CLUSRCVR)	STATUS(RUNNING)
RQMNAME(qmgr1)	
AMQ8417: Display Channel Status details.	
CHANNEL(T0.qmgr1)	XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
CONNAME(wmq1(3601))	CURRENT
CHLTYPE(CLUSSDR)	STATUS(RUNNING)
RQMNAME(qmgr1)	

---

## 6.7 Configuration of WebSphere Application Server

No configuration changes are necessary to the server1 application server if the linuxHATestReceiverQCF's Transport Type is defined as **BINDINGS**, as we chose for scenario two. Review "WebSphere Application Server configuration" on page 93.

## 6.8 Application in action

In this section, the applications are started under normal operation and load-balancing is reviewed. Once a normal operating state is reached, the failover test is executed.

### Normal Operation

To start normal operation, do the following:

1. Ensure that the queue managers are running on all the servers by executing the `dspmq` command.
2. Start the message generating application on qmgr1 using the command `./msggen 1 mem cq qmgr1`.

**Note:** Cluster queue cq does not exist on qmgr1. The msggen application will succeed to put messages to cq because it is a cluster queue and it is accessible from any queue manager within the cluster

The messages will be sent to cluster queue cq on one of the queue managers qmgr4 or qmgr5. You can check this by executing the following commands.

Example 6-11 shows us the channel status on qmgr1. Because the cluster queue cq was defined with DEFBIND(NOTFIXED), the queue manager has spread the messages over the two channels. Qmgr1 has sent 34 messages to each instance of the queue cq.

*Example 6-11 Channel status on qmgr1*

---

```
dis chs('T0.qmgr4') msgs
  7 : dis chs('T0.qmgr4') msgs
AMQ8417: Display Channel Status details.
  CHANNEL(T0.qmgr4)                XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
  CONNAME(brk1(3603))              CURRENT
  CHLTYPE(CLUSSDR)                 STATUS(RUNNING)
  MSGS(34)                          RQMNAME(qmgr4)
dis chs('T0.qmgr5') msgs
  8 : dis chs('T0.qmgr5') msgs
AMQ8417: Display Channel Status details.
  CHANNEL(T0.qmgr5)                XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
  CONNAME(brk2(3604))              CURRENT
  CHLTYPE(CLUSSDR)                 STATUS(RUNNING)
  MSGS(34)                          RQMNAME(qmgr5)
```

---

Example 6-12 shows the same information as Example 6-11, this time for queue manager qmgr2. Here, 69 messages have been sent, spread over the two instances of cluster queue cq.

*Example 6-12 Channel status on qmgr2*

---

```
dis chs('T0.qmgr4') msgs
  11 : dis chs('T0.qmgr4') msgs
AMQ8417: Display Channel Status details.
  CHANNEL(T0.qmgr4)                XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
  CONNAME(brk1(3603))              CURRENT
  CHLTYPE(CLUSSDR)                 STATUS(RUNNING)
  MSGS(34)                          RQMNAME(qmgr4)
dis chs('T0.qmgr5') msgs
  12 : dis chs('T0.qmgr5') msgs
AMQ8417: Display Channel Status details.
  CHANNEL(T0.qmgr5)                XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
  CONNAME(brk2(3604))              CURRENT
  CHLTYPE(CLUSSDR)                 STATUS(RUNNING)
  MSGS(35)                          RQMNAME(qmgr5)
```

---

Both instances of the cluster queue cq actually point to the same local queue q1 on queue manager qmgr3. Inspecting the current depth of this queue, we see that all messages have arrived on this queue. See Example 6-13 on page 196.

### Example 6-13 Queue depth on qmgr3

---

```
5 : dis ql('q1') curdepth
AMQ8409: Display Queue details.
      QUEUE(q1)                                CURDEPTH(137)
```

---

### Failover scenario

Now that we are successfully sending and receiving messages, it is time to test the failover. Power off the brk2 server.

The message receiving application should continue to receive all messages sent from qmgr1 and qmgr2. This is because the cluster automatically detects that qmgr5 is no longer available. Messages being put via qmgr1 and qmgr2 are being forwarded to qmgr4 only, then forwarded on to qmgr3 for local retrieval.

Example 6-14 shows the channel statistics for qmgr1 during the failure of brk2. All messages are now sent via channel TO.qmgr4, while the channel TO.qmgr5 is retrying to establish the connection.

### Example 6-14 Channel status during the failure of qmgr5

---

```
dis chs('TO.qmgr4') msgs
  13 : dis chs('TO.qmgr4') msgs
AMQ8417: Display Channel Status details.
      CHANNEL(TO.qmgr4)                        XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
      CONNAME(brk1(3603))                      CURRENT
      CHLTYPE(CLUSSDR)                        STATUS(RUNNING)
      MSGS(76)                                 RQMNAME(qmgr4)

dis chs('TO.qmgr5') msgs
  14 : dis chs('TO.qmgr5') msgs
AMQ8417: Display Channel Status details.
      CHANNEL(TO.qmgr5)                        XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
      CONNAME(brk2(3604))                      CURRENT
      CHLTYPE(CLUSSDR)                        STATUS(RETRYING)
      MSGS(0)                                 RQMNAME(qmgr5)
```

---

### Failback scenario

Because this test is not showing the failover and failback of file system /var/mqm/qmgrs/qmgr5, all that needs to be done for failback is power up the brk2 server and wait for the queue manager to restart.

Once qmgr5 is started, it should automatically resolve incomplete transmissions and rejoin the cluster. Once qmgr5 has rejoined the cluster, it will immediately

begin receiving messages from qmgr1 and qmgr2 because the message generating applications are still running.

During our lab tests, we noticed one effect of clustering. Messages arrived at qmgr5 but not yet transmitted to qmgr3 at the time of the power failure, will be sent to qmgr3 after restarting brk2. This is shown in the Example 6-15.

The command **dis ql('q1') curedepth** was first executed with no generators running and host brk2 shut down. After restarting host brk2 and thus restarting qmgr5, two orphaned messages appeared. These messages were in transit on qmgr5 when the host brk2 was shut down.

*Example 6-15 Display of queue depth*

---

```
AMQ8409: Display Queue details.
  QUEUE(q1)                                CURDEPTH(87)
dis ql('q1') curdepth
  9 : dis ql('q1') curdepth
AMQ8409: Display Queue details.
  QUEUE(q1)                                CURDEPTH(89)
```

---

Figure 6-6 on page 198 shows the same symptom on the Web-based monitoring application. Messages 173 and 172 have been received out of order by the Web application and thus by qmgr3. These two messages were in transit on qmgr5 when the system failed.

Monitoring Resources	
Message Number	Message Payload
176	No messages received
175	No messages received
174	No messages received
173	memory,2004-02-30-16.58.45,qmgr1,1134
172	cpu,2004-02-30-16.58.45,qmgr2,69
171	cpu,2004-03-30-16.59.42,qmgr2,79
170	cpu,2004-03-30-16.59.41,qmgr2,54
169	memory,2004-03-30-16.59.40,qmgr1,3022
168	cpu,2004-03-30-16.59.40,qmgr2,19
167	memory,2004-03-30-16.59.39,qmgr1,3161
166	cpu,2004-03-30-16.59.39,qmgr2,90
165	memory,2004-03-30-16.59.38,qmgr1,2165
164	cpu,2004-03-30-16.59.38,qmgr2,78
163	memory,2004-03-30-16.59.37,qmgr1,787
162	cpu,2004-03-30-16.59.37,qmgr2,88
161	memory,2004-03-30-16.59.36,qmgr1,3603
160	cpu,2004-03-30-16.59.36,qmgr2,59
159	memory,2004-03-30-16.59.35,qmgr1,3135
158	cpu,2004-03-30-16.59.35,qmgr2,27
157	memory,2004-03-30-16.59.34,qmgr1,3530

Figure 6-6 Web page showing the recovery of messages

## 6.9 Summary

This chapter demonstrates how a queue manager clustering alone can create a more highly-available system by allowing sending and receiving applications to continue processing entirely uninterrupted. The issue that was not resolved in this scenario is the possibility of orphaned messages on the failing queue manager. All that is needed to fix this issue is use the hardware-based, high-availability solutions presented in either scenario one, two or three. Combining both solutions creates a complete high-availability solution for WebSphere MQ.



## Implementing highly available brokers

This chapter introduces WebSphere BI Message Broker and discusses a configuration that demonstrates implementing a highly-available solution through the use of WebSphere MQ clustering.

## 7.1 Scenario overview

The scenario discussed in this chapter extends the one created in Chapter 6, “Using WebSphere MQ clustering” on page 179 by building WebSphere BI Message Broker upon the brk1 and brk2 servers.

### 7.1.1 Using brokers with WebSphere MQ clustering for high availability

The WebSphere BI Message Broker topology depicted in Figure 7-1 on page 201 is used to introduce readers to WebSphere BI Message Broker and to demonstrate how WebSphere MQ clustering can provide workload balancing and high availability for broker-based services. Message generators will be putting messages to queue managers not housing an instance of the cluster queue. The queue managers will be allowed to send the messages in a round-robin pattern to any available instance of the cluster queue. A broker instance will retrieve the messages and insert them into an external database. This design allows a running broker and queue manager combination to retrieve all messages as long as one matching cluster queue is still available.

The WebSphere MQ configuration supporting this scenario is slightly different from the configuration used in Chapter 6, “Using WebSphere MQ clustering” on page 179. This time, the cluster queue cq is a local queue and not a remote queue. Example 7-1 shows the commands to be executed on qmgr4 and qmgr5 to redefine the queue cq as a local cluster queue.

*Example 7-1 Redefine the queue cq as a local queue*

---

```
delete qr('cq')
define ql('cq') defpsist(YES) defbind(NOTFIXED) cluster(ITS0)
```

---



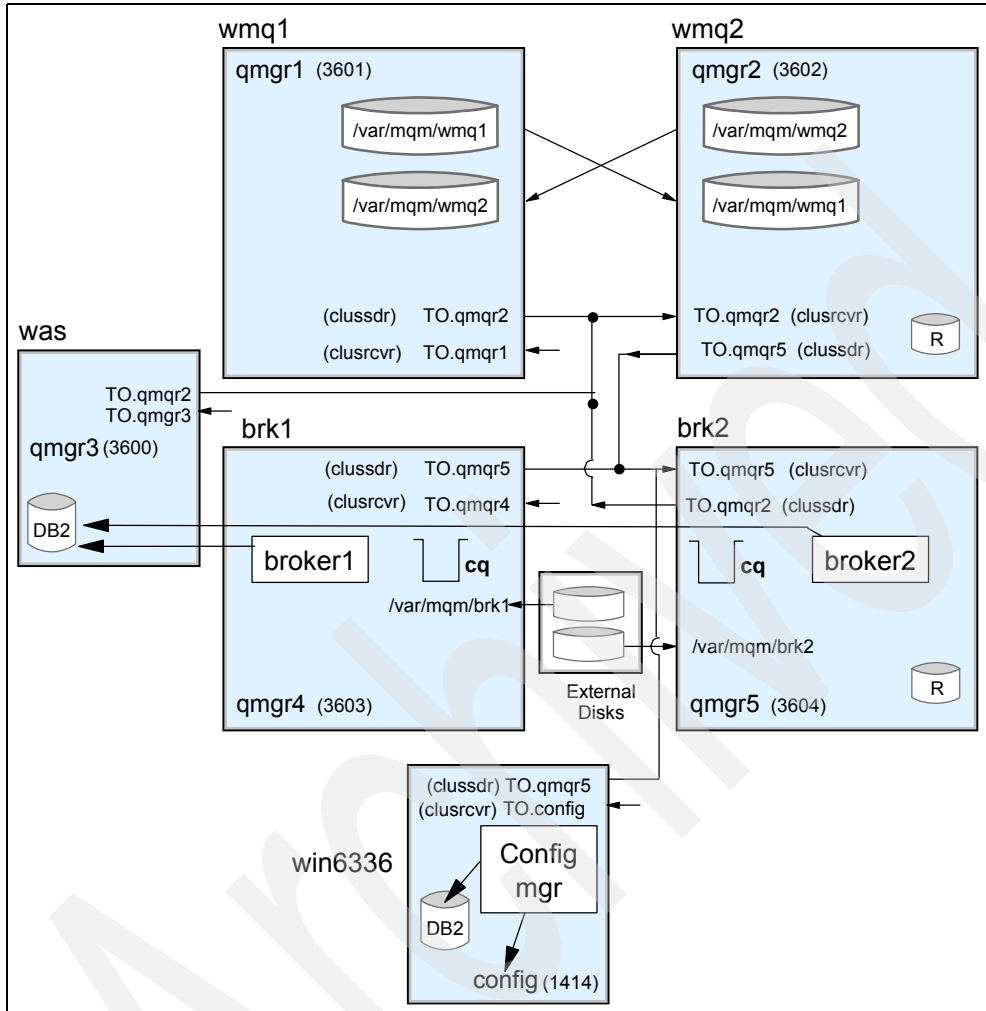


Figure 7-1 WebSphere MQ Cluster and brokers

## 7.1.2 Using hardware failover for high-availability in brokers

WebSphere BI Message Broker and its underlying queue manager and database can be configured to be highly available by exploiting operating system support for failure detection and failover to a standby machine. A hardware failover configuration for the broker resources is not discussed in this book, however it would be similar to one of the HA scenarios presented earlier. The queue manager housing the broker would be recovered using one of the methods

discussed in either scenario one, two or three. In addition the broker files and database would also need to be monitored and recovered.

A complete high availability solution for a broker would probably involve queue manager clustering and hardware failover as was described in Chapter 6, “Using WebSphere MQ clustering” on page 179. In this book, high availability for brokers is only discussed through the use of WebSphere MQ clustering. Examples of HA configurations for the message broker can be found at:

<http://www-306.ibm.com/software/integration/support/supportpacs/>

## 7.2 Introduction to WebSphere BI Message Broker

WebSphere BI Message Broker enables information, packaged as messages, to flow between different business applications, ranging from large existing systems to unmanned devices such as sensors on pipelines as illustrated in Figure 7-2.

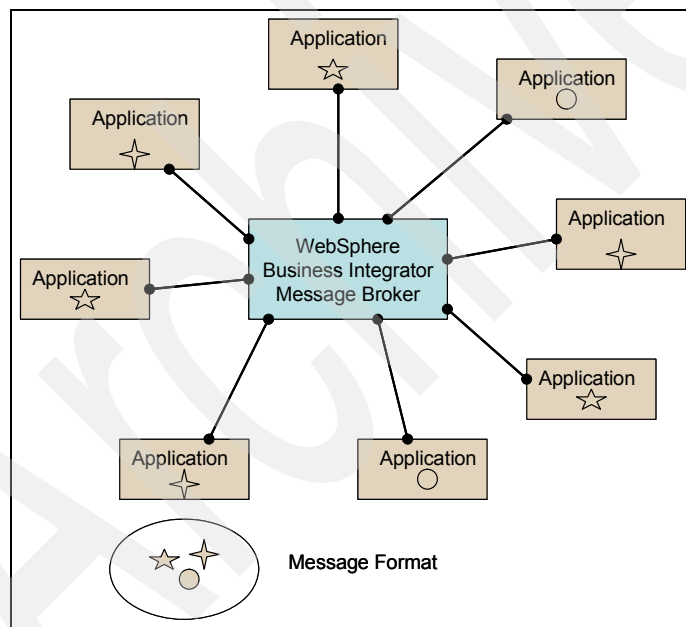


Figure 7-2 WebSphere BI Message Broker application

There are two ways in which WebSphere Business Integration Message Broker can act on messages: message routing and message transformation.

## Message routing

Messages can be routed from sender to recipient based on the content of the message.

## Message transformation

Messages can be transformed before being delivered:

- ▶ They can be transformed from one format to another, perhaps to accommodate the different requirements of the sender and the recipient.
- ▶ They can be transformed by modifying, combining, adding or removing data fields, perhaps involving the use of information stored in a database. Information can be mapped between messages and databases.

## Business needs for message routing and transformation

There are many reasons to implement a message broker within an enterprise. A few needs are listed below, and all are addressed by the message broker:

1. Information that flows between business systems can come from, and be sent to, different types of applications and processes, ranging from large existing systems to unmanned devices such as sensors.
2. The format and the content of the information required by each business application can be very different. Exchanging messages is easier if the format of the data and the content of the message is transformed and enriched while the message is in transit.
3. The sender application might not know the final destination of a message. This means that messages have to be sent to every application on the network, increasing the number of information flows. Targeting messages to the correct recipient is easier if the decision on where to send the message is based on the information in the message.
4. Businesses need to be able to respond quickly and effectively to information about events. These events can be reported from any device at any time anywhere in the world.

### 7.2.1 WebSphere BI Message Broker architecture

WebSphere BI Message Broker consists of following components as illustrated in Figure 7-3 on page 204.

- ▶ One or more brokers
- ▶ A configuration manager
- ▶ One or more users using the message brokers toolkit
- ▶ A repository, shared between the users of the message brokers toolkit.

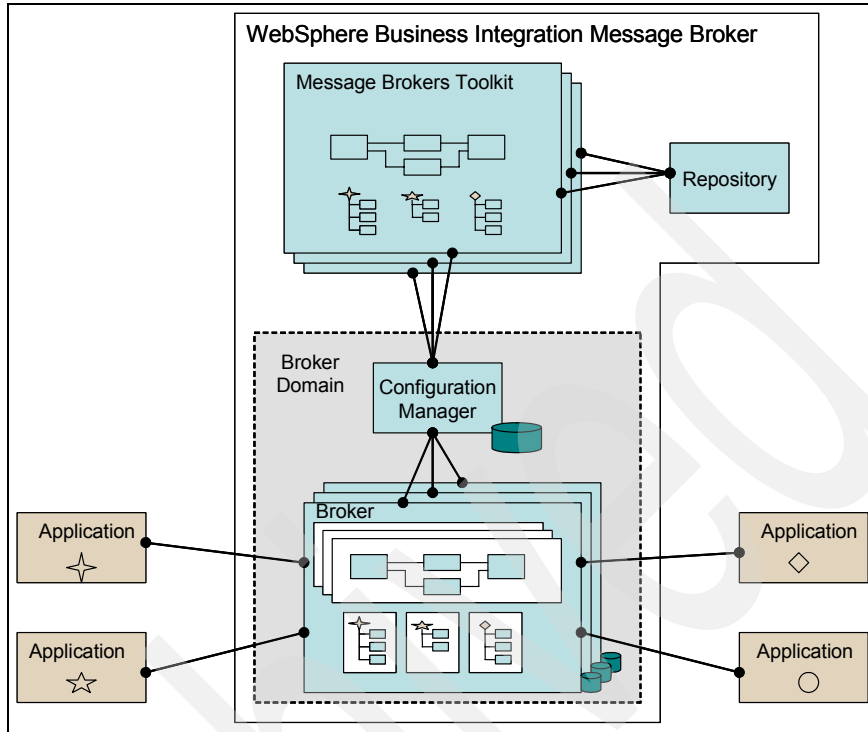


Figure 7-3 Components of WebSphere BI Message Broker

## 7.2.2 The broker

As illustrated in Figure 7-4 on page 205, the broker processes inflight messages based on message flows and message sets to provide message routing and transformation functionality. A broker can run many message flows simultaneously.

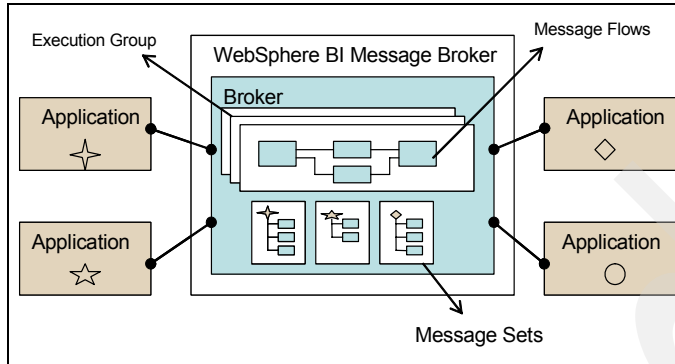


Figure 7-4 Broker functionality

### Execution group

Each message flow is assigned to a unique broker process called an *execution group* which runs one or more message flows assigned to it in one or more threads from the execution group's thread pool. Many execution groups can be created within a single message broker.

## 7.2.3 Introducing message flows

Message flows can route messages, transform messages, filter messages (topic-based or content-based), and have database capabilities for enrichment the messages or for warehousing them. WebSphere Business Integration Message Brokers also provides a framework for extending the functionality with *plug-ins*, user-written or third-party solutions for specific requirements.

*Message flows* define a set of operations that are performed on a message when it is received by a broker. A flow consists of a number of *message flow nodes*, which are wired together to allow messages to be processed. Each node in a flow represents a processing step. A node has *terminals* for input, output or both.

A message flow can also be thought of as a graphical representation of message flow nodes representing actions performed on a message when it is received and processed by a broker. See Figure 7-6 on page 207. Each node in a message flow represents a processing step and the connections between the nodes represent the direction that the flow takes, based on the outcome of each step. A message flow must include an input node that provides the source of the messages to be processed. A message flow is deployed to an execution group in a broker, where the flow runs in one or more threads from the execution group's thread pool.

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

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

In this book, only the MQInput and MQOutput input and output nodes are reviewed. While several other processing nodes are available, only a few are reviewed.

Message flows can be created for the following communication models:

- ▶ Point-to-point
- ▶ Publish/subscribe

A point-to-point message flow is used in this chapter. It is similar to the one illustrated in Figure 7-5 on page 207. The message flow takes a message from a client and delivers copies, some transformed, to any number of back-end servers for processing. While a message passes through a message flow, it is transformed and routed according to the nodes it encounters.

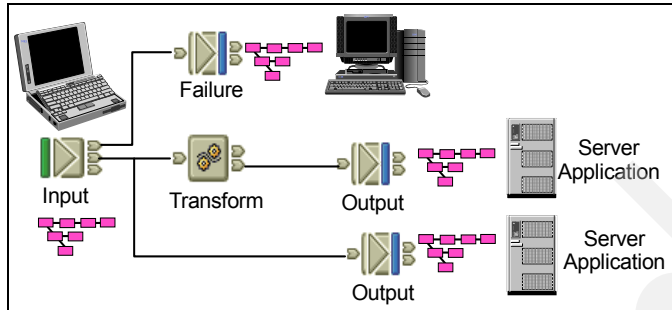


Figure 7-5 Point-to-point model of message flow

## Building message flows

The Message Brokers Toolkit is used to build flows by placing nodes on the canvas and connecting the terminals. Figure 7-6 shows a simple message flow opened in the message flow editor in the Toolkit.

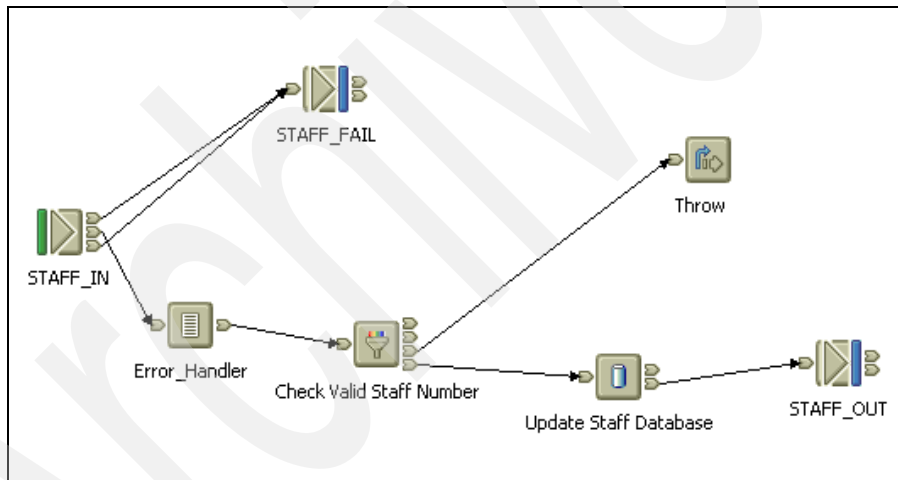


Figure 7-6 Example of a message flow

Message flow nodes are configured through their properties. The properties that are available depend on the type of the node. For example, the MQOutput node has a mandatory property for the queue name and an optional property for the queue manager name. For the MQInput node, you would specify the queue name, the type of message, and other relevant options. See Figure 7-7 on page 208.

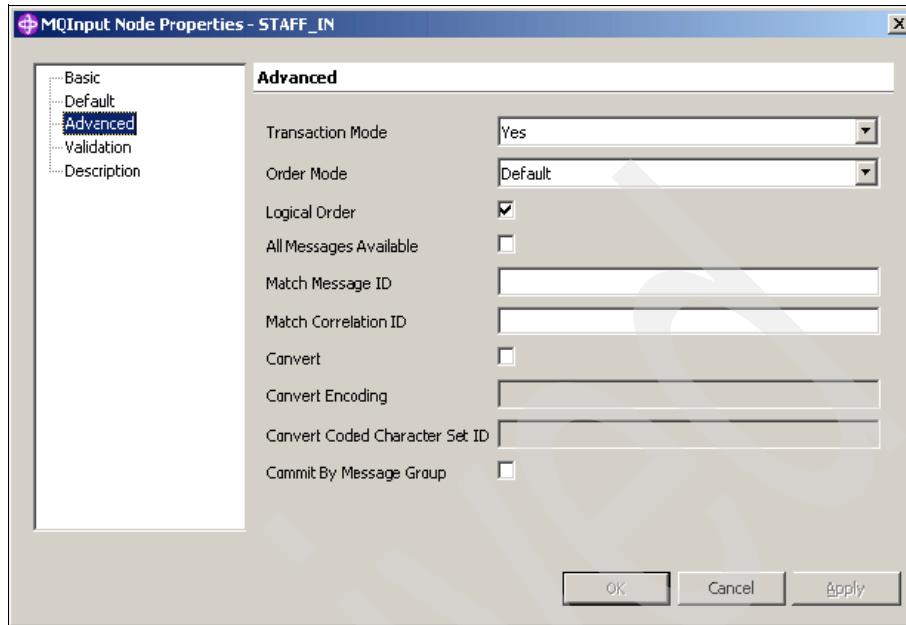


Figure 7-7 Properties of the MQInput node

Some nodes can process messages using ESQL statements. *Extended SQL* (ESQL) is an interpreted language based on Structured Query Language (SQL) that supports the manipulation of data within a message flow. ESQL can be used to develop complex business logic, but it is more commonly used to perform simple operations on messages and database contents. The language includes looping and branching capabilities. ESQL is most often used in Compute nodes and Filter nodes. Example 7-2 shows a sample ESQL module that is used to split one single messages into multiple messages using the keyword PROPAGATE.

*Example 7-2 ESQL module for a Compute node*

---

```

CREATE COMPUTE MODULE Propagate_multiple_messages
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
DECLARE I INTEGER;
SET I=1;
WHILE I<=Environment.Variables.NoPassengers
DO
    SET OutputRoot = InputRoot;
    SET OutputRoot.XML = NULL;
    SET
OutputRoot.XML.PassengerReservationResponse.ListOfConfirmations.Confirmation =
InputRoot.XML.Reservation.ListOfPassengers.PassengerDetails[I];
    PROPAGATE;

```



```
    SET I=I+1;
END WHILE;
RETURN FALSE;
END;
END MODULE;
```

---

Other nodes provide a graphical interface to define operations on messages.

A message flow can be embedded in another flow. Embedded flows are known as *subflows*. Subflows allow common processing, for example error handling, to be reused. A subflow appears as a single node in the main message flow. The `Error_Handler` node in Figure 7-6 on page 207 represents the subflow shown in Figure 7-8. Subflows are comprised of the same nodes as main flows, except that they have Input and Output nodes to connect them to the main flow, and not the usual `MQInput` and `MQOutput` nodes.

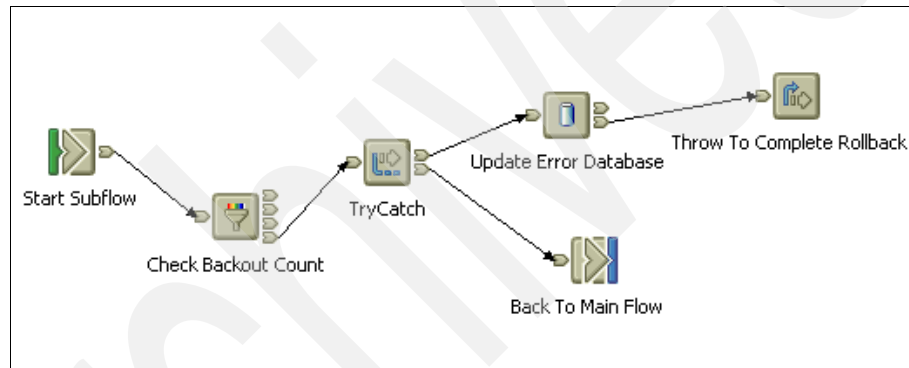


Figure 7-8 An error handling subflow: `Error_Handler`

## Wiring nodes together

Each node has input and output terminals. Terminals vary with the node type, but typically you might have an:

- ▶ *In* terminal that receives the message from the previous node or transport
- ▶ *Out* terminal used to pass the message along to the next node for processing
- ▶ A *failure* terminal to handle failure conditions

Nodes are strung together to form a message flow by wiring an output terminal of one node with the input terminal of another.

Figure 7-9 on page 210 shows the toolkit. The right side of the figure shows the development of a message flow. With the connector selected, a pop-up window tells us that the two nodes are connected from the out terminal to the in terminal of the next node. In the bottom left corner of Figure 7-9, there is the view Outline.

By expanding the Outline tree, you can again see how one node is connected to the next node.

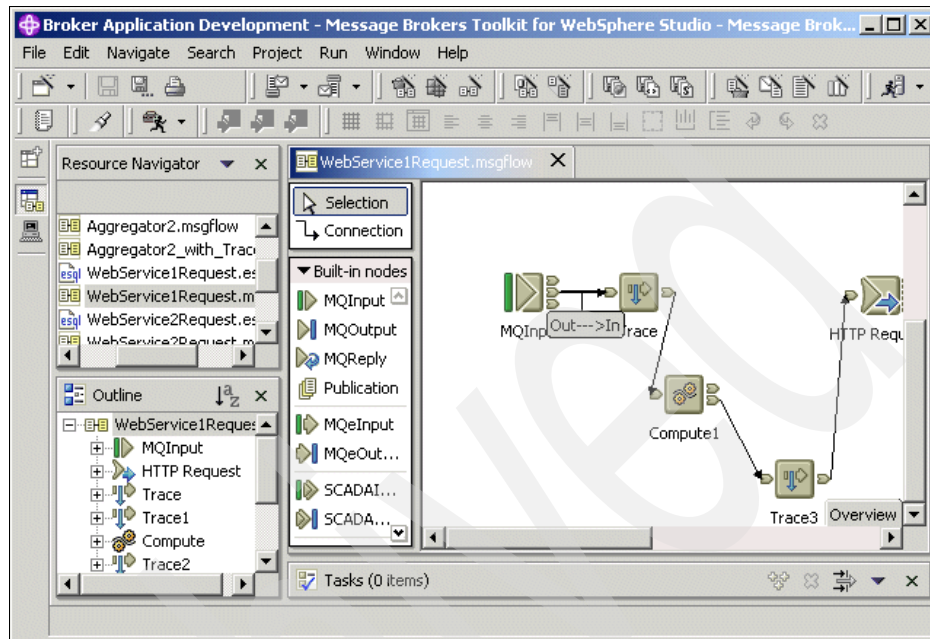


Figure 7-9 Connecting nodes

## Node types

WebSphere Business Integration Message Brokers comes with a set of built-in nodes ready to use for building message flows. The built-in nodes shown in Table 7-1 show the broad range of function provided.

Table 7-1 Message manipulation nodes

Node	Function
<b>Nodes for message manipulation</b>	
► Compute	Changes a message or contents of a database using ESQL
► Database ► DataDelete ► DataInsert ► Dataupdate ► Warehouse	Used to interact with an ODBC data source.

Node	Function
▶ Extract	Used to extract the exact contents of the input message that you want to be processed by later nodes in the message flow.
▶ Mapping	Use the Mapping node to construct one or more new messages by creating new messages and populating them with new information, with modified information from the input message, or with information taken from a database.
▶ XMLTransformation	Applies a stylesheet to an XML message.
▶ AggregateControl ▶ AggregateReply ▶ AggregateRequest	Used to combine the generation and fan-out of a number of related requests with the fan-in of the corresponding replies, and compile those replies into a single aggregated reply message.
<b>Nodes for decision making</b>	
▶ Check	Validates the format of a message
▶ Filter	Routes a message based on conditional logic
▶ FlowOrder	Used to control the order in which a message is processed by a message flow.
▶ Label, RouteToLabel	Use the Label node in combination with a RouteToLabel node to dynamically determine the route a message takes through the message flow, based on its content.
▶ ResetContentDescriptor	Used to request that the message is reparsed by a different parser.
<b>Nodes for error handling</b>	
▶ Throw	Used to throw an exception within a message flow
▶ Trace	Used to generate trace records that can incorporate text, message content, and date and time information, to help you to monitor the behavior of the message flow.
▶ TryCatch	Used to provide a special handler for exception processing.

If you need to perform processing that is not supported by the built-in nodes, WebSphere BI Message Broker allows the development of custom user-defined nodes. These can be written in either C or Java.

Additional nodes are available as SupportPacs and are free to download from the IBM Web site:

<http://www-306.ibm.com/software/integration/support/supportpacs/>

**Note:** Always check to be certain that a specific node is compatible with all of the target platforms on which the message flow will be deployed.

## Transport support for applications

This book only reviews standard WebSphere MQ messaging as a transport type supported by a WebSphere BI Message Broker, but there are several other protocols available. Table 7-2 and Figure 7-10 on page 213 summarize the available transport mechanisms and their supporting nodes that allow clients to communicate with other applications through message flows.

Table 7-2 Application transport support

Nodes	Transport
<ul style="list-style-type: none"> <li>▶ SCADAInput</li> <li>▶ SCADAOutput</li> </ul>	<p><i>WebSphere MQ Telemetry Transport</i> is a lightweight publish/subscribe protocol flowing over TCP/IP. This protocol is used by specialized applications on small footprint devices that require a low bandwidth communication, typically for remote data acquisition and process control.</p>
<ul style="list-style-type: none"> <li>▶ Real-timeInput</li> <li>▶ Real-timeOptimizedFlow</li> <li>▶ Publication</li> </ul>	<p><i>WebSphere MQ Multicast Transport</i> is used by dedicated multicast-enabled JMS application clients to connect to brokers. Applications communicate with the broker by writing data directly to TCP/IP ports. This protocol is optimized for high volume, one-to-many publish/subscribe topologies.</p>
<ul style="list-style-type: none"> <li>▶ HTTPInput</li> <li>▶ HTTPReply</li> <li>▶ HTTPRequest</li> <li>▶ Publication</li> </ul>	<p><i>WebSphere MQ Web Services Transport</i> allows Web services clients using XML messages and the HTTP protocol running over TCP/IP to communicate with applications through message flows in a broker.</p>
<ul style="list-style-type: none"> <li>▶ Real-timeInput</li> <li>▶ Real-timeOptimizedFlow</li> <li>▶ Publication</li> </ul>	<p><i>WebSphere MQ Real-time Transport</i> is a lightweight protocol optimized for use with non-persistent messaging. JMS applications communicate with the broker using TCP/IP ports.</p>
<ul style="list-style-type: none"> <li>▶ MQInput</li> <li>▶ MQOutput</li> <li>▶ MQReply</li> <li>▶ Publication</li> </ul>	<p><i>WebSphere MQ Enterprise Transport</i> supports WebSphere MQ applications that connect to WebSphere BI Message Broker by writing data to and reading data from message queues.</p>

Nodes	Transport
<ul style="list-style-type: none"> <li>▶ MQeInput node</li> <li>▶ MQeOutput node</li> <li>▶ Publication</li> </ul>	<p><i>WebSphere MQ Mobile Transport</i> is used exclusively by WebSphere MQ Everyplace clients. WebSphere MQ Everyplace is an application designed primarily for messaging to, from, and between pervasive devices. These are typically small, handheld devices, such as mobile phones and PDAs. A bridge queue on the broker's queue manager provides an interface for the WebSphere MQ Everyplace clients to the broker services.</p>

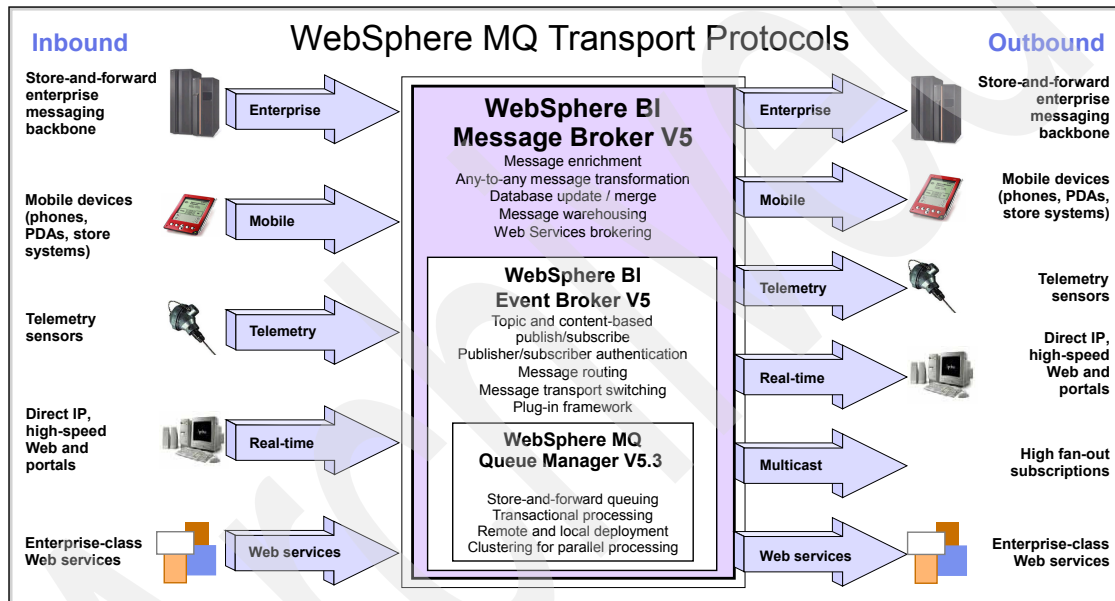


Figure 7-10 Application transport graph

## Message flow tuning

The design of WebSphere BI Message Broker generally provides several ways of tuning message flows for performance. Several levels of tuning are provided for paralleling message processes:

- ▶ Multiple brokers
- ▶ Multiple execution groups within a broker
- ▶ Multiple flow instances within an execution group

This parallelism can break message ordering requirements, and the configuration can be managed at deployment to mitigate for this. However in

general, it turns a multithreaded system back into a single-threaded one. The best solution is try to eliminate the need for ordering of messages or message affinities unless it can not be avoided.

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

## 7.2.4 Introduction to message modeling and message sets

Message flows are designed to process application messages as per business rules. Before starting to build the message flow, the messages to be processed by the message flow should be clearly defined. Defining messages is performed in two steps:

1. Defining the logical format
2. Defining the physical format.

WebSphere BI Message Broker supports various logical and physical formats of messages.

### Messages

A *message* is a set of data passed from one application to another. A message can be modeled to describe the structure and content of the message meaningful to the applications sending and receiving these messages. An example of the message is illustrated in Figure 7-11.

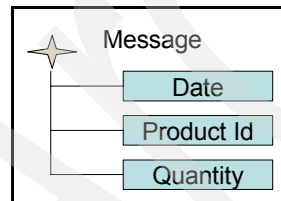


Figure 7-11 A message

### Message set

A message *set* is a logical grouping of messages and the objects that comprise them. The message set file provides information that is common across all the messages in the message set. An example of message set is illustrated in Figure 7-12 on page 215.

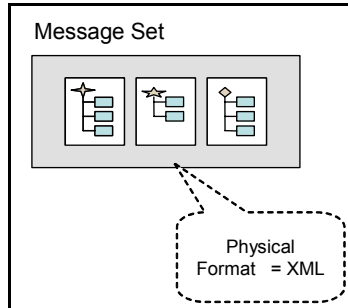


Figure 7-12 A message set

## Message definition

A message *definition* consists of elements organized into a logical structure agreed between the sending and receiving applications. The message definition is stored in .xsd files.

## Elements

An *element* is a field within a message and has a specific meaning as agreed to by the applications that create and process the message. An element is always based on a type, String and Time, for example. An element can be a simple or complex element and can be a global or local element. An element's value can be constrained using value constraints which define the range of acceptable values for the element. An example of an element is illustrated in Figure 7-13.

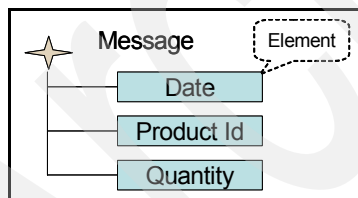


Figure 7-13 An element

There are four kinds of elements:

- ▶ A *simple element* is a single field of a message based on a simple type such as a String, Integer or Date.
- ▶ A *complex element* contains other elements. For example, a complex element named ServerResourceUsage might contain elements such as ResourceName and Usage. A complex element can also contain other complex elements.

- ▶ A *global element* can be used in several different messages or even in several places within the same message. It must be given a unique name so it can be referenced.
- ▶ A *local element* is defined in one position within one complex element and is not available for reuse elsewhere in the message model.

## Data types

Type defines the abstract definition of data, and are classified as simple or complex.

- ▶ A *simple type* is an abstract definition of an item of data such as a number, a string or a date. The purpose of a simple type is to define the content of the simple element. For example a simple element 'ResourceName' is of simple type String.
- ▶ A *complex type* is used to define the complex element. They are an essential part of every message model, because they define the logical structure of the messages by combining other simple and complex elements. Almost any message structure can be modeled.

## Message categories

A message category provides a way for grouping messages for documentation or convenience purposes or for assisting in the generation of Web Services Description Language ( WSDL) files.

## Model importers

The message definition can be created using one of the supplied importer tools. Importers are available for XML DTD, XML Schema, C structures and COBOL structures.

## Message set project

A *message set project* is a container for all the resources associated with exactly one message set.

## Model editors

The message set files, message definition files and message category files have their own editors in the message broker toolkit which can be used to create and maintain their resources.

## Model generators

Model generators can be used to generate the HTML documentation. Generators are also used for generating a message dictionary which is deployed



to a broker and to generate Web Services Description Language (WSDL) files used by Web services clients.

### Message model components

Figure 7-14 illustrates the different components of a message model that we have discussed in this section.

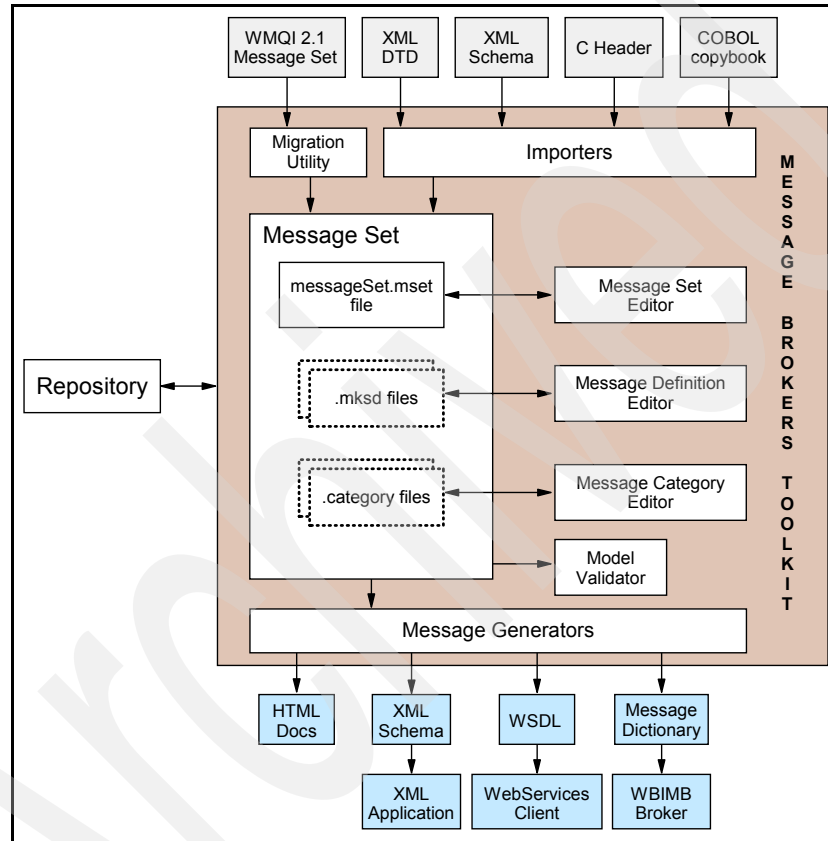


Figure 7-14 Message model components

## 7.2.5 Message Brokers Toolkit for WebSphere Studio

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

The Message Brokers Toolkit consists of a Workbench window which displays one or more perspectives. A *perspective* is a group of views and editors required to perform tasks associated with a role. Each perspective consists of views that provide alternative presentations of resources or ways of navigating through information in the Workbench. As a user works with the Workbench, the data representing the projects and working environment are stored in a workspace directory on the local file system.

**Attention:** Currently the Message Broker Toolkit is available on Windows platforms only. The broker itself can run on many platforms.

A number of perspectives are provided with the Message Brokers Toolkit. These perspectives are briefly discussed in the following sections.

### **Broker Administration perspective**

The Broker Administration perspective is the perspective where the broker domain resources (also referred to as domain objects) that are defined on one or more Configuration Managers are managed. Typically, one can perform the following tasks in this perspective:

1. Setting up the broker domain
2. Creating and removing domain connections
3. Connecting and removing brokers
4. Adding and removing execution groups
5. Creating and deploying broker archive (bar) files
6. Defining and managing publish/subscribe topic hierarchies
7. Querying, viewing and deleting subscriptions
8. Viewing, clearing and filtering deploy event log entries
9. Activating, deactivating, filtering and clearing alerts.

The Broker Administration perspective is as illustrated in Figure 7-15 on page 219.



If a message definition is opened from the Resource Navigator, a message editor will be opened in the top right.

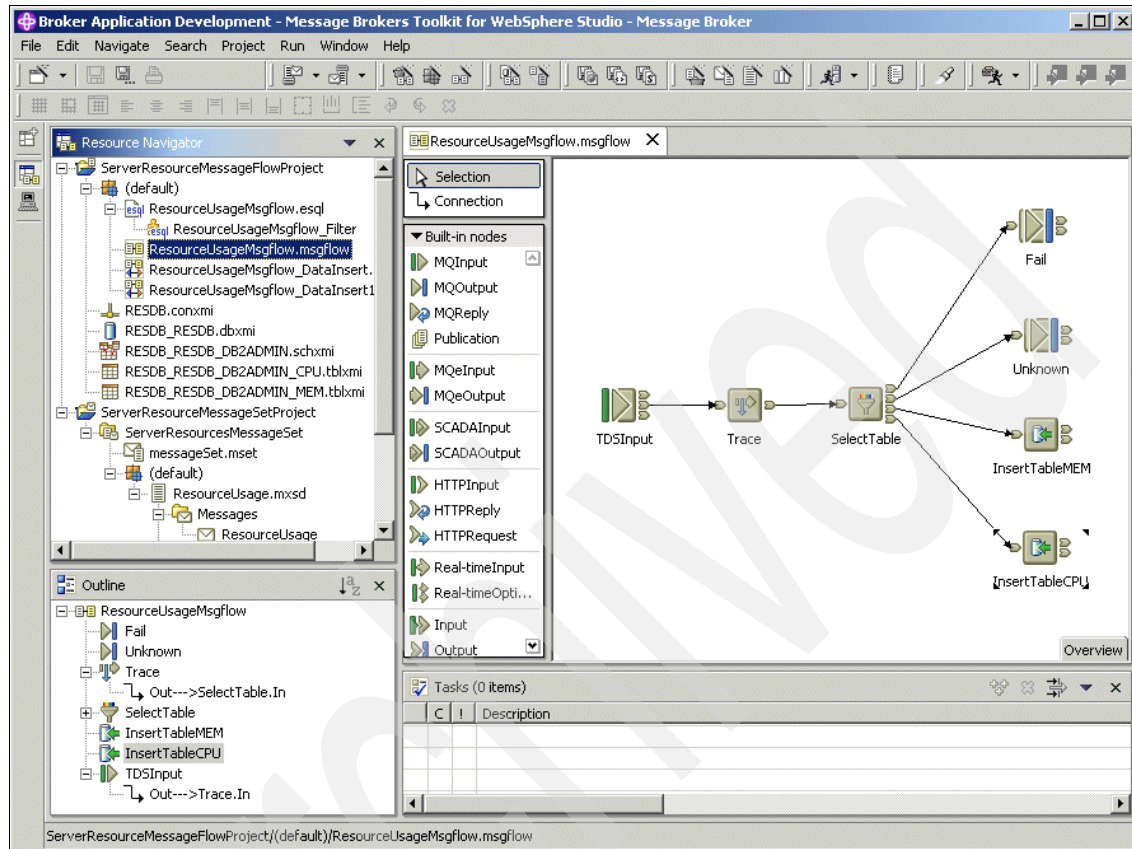


Figure 7-16 Broker Application Development perspective

### Flow debug perspective

This perspective is used to test and debug message flows in a graphical way.

### Plug-in development perspective

This perspective helps you develop user-defined plug-in nodes in the workbench. A user-defined plug-in node is an Eclipse plug-in that adds a category of nodes to the message flow editor palette.

The New Plug-in Project wizard creates the supporting workbench files for your user-defined plug-in node. The New Message Flow Plug-in Node wizard creates a message node file (.msgnode) and launches the editor for developing the visual representation of your user-defined plug-in node in the workbench.

## Server perspective

This perspective is used to manage server resources. Typical tasks carried out in this perspective include:

1. Creating server processes
2. Configuring a server
3. Controlling rapid application development (RAD) execution groups (servers)
4. Deploying resources

### 7.2.6 WebSphere BI Message Broker configuration manager

The configuration manager interfaces between the Message Broker Toolkit workbench and an executing set of brokers. It provides brokers with their initial configuration, and also updates them with any subsequent changes. These configuration delivery conversations between a configuration manager and a broker are called *deployments*. It also maintains the overall broker domain configuration. The configuration manager is not required to be running, at broker execution time, only at configuration time.

There is only one configuration manager for each broker domain. As of now, the only supported platform for the configuration manager is the Windows platform. The configuration manager requires the use of a WebSphere MQ queue manager and this can be shared with one broker in the broker domain. The main functions of the configuration manager are:

1. Maintain the configuration repository, a set of database tables that hold a central record of broker domain components.
2. Deploy the broker topology and message processing operations in response to actions initiated through the workbench. Broker archive files are deployed through the configuration manager to execution groups of the brokers in the broker domain.
3. Report current status back to the workbench, relating to the results of deployment operations and current broker status.

### 7.2.7 User name server

The user name server is an optional runtime component that provides authorization of users and groups performing publish/subscribe operations.

If you have applications that use the publish/subscribe broker services, you can apply an additional level of security to the topics on which messages are published and subscribed. This additional security, known as *topic-based security*, is managed by the user name server. It provides administrative control over who can publish and who can subscribe. For example, if a client application

publishes messages containing sensitive company finance information, or personnel details, the user name server can be used to restrict access to those messages. The user name server interfaces with operating system facilities to provide information about valid users and groups in a broker domain.

## 7.3 Installing and configuring the broker

This section discusses installing the broker.

### 7.3.1 Installing and configuring WebSphere MQ

Perform the same steps outlined in 3.3, “Installing and configuring WebSphere MQ” on page 68. There are no specific installation requirements for this scenario.

Note that a broker solution requires at least one configuration manager which needs to run on Windows. The configuration manager also requires a queue manager. The installation of WebSphere MQ for use by the configuration manager does not require any special attention.

### 7.3.2 Installing and configuring DB2

WebSphere BI Message broker uses DB2 on Linux to store information required for broker operations. In our setup, DB2 was installed on the was server and DB2 client was installed on the brk1 and brk2 servers, where WebSphere BI Message Broker was installed. A detailed description on the steps to install DB2 server is beyond the scope of this document. Refer to the IBM Redbook *Up and Running with DB2 for Linux* (SG24-6899-00) .

### 7.3.3 Installing and configuring WebSphere BI Message Broker

The installation of WebSphere BI Message Broker is divided into two sections. Installing the Configuration Manager and Message Broker Toolkit on Windows and installing the broker on Linux. The first is only briefly described because it is fully installed on a Microsoft Windows operating system and is detailed in many other documents as well. Installing and configuring the WebSphere BI Message Broker support on Linux is discussed in more detail.

#### Installation and configuration references

The complete instructions for installing and configuring the components can be found in the documentation located at:

<http://www-306.ibm.com/software/integration/wbimessagebroker/library/>

In addition you can review the IBM Redbook *Migration to WebSphere Business Integration Message Broker V5* (SG24-6995-00), which contains many helpful hints about installing and configuring WebSphere BI Message Broker.

## Windows 2000 installation components overview

Before the main software components of WebSphere BI Message Broker are installed some additional support software is required:

- ▶ Windows 2000 Service Pack 3
- ▶ Java Runtime Environment 1.3.1

Prior to installing WebSphere Business Integration Message Broker V5 components, install the required software:

- ▶ Install WebSphere MQ V5.3 and the latest CSD.
- ▶ Install DB2 V8.1 and apply Fixpack 2 or later.

Up-to-date system requirements can be found at:

<http://www-306.ibm.com/software/integration/wbimessagebroker/requirements/>

For our lab environment, we performed a custom install of both products, installing all components.

**Tip:** Once you have completed the base install, immediately complete the installation of the latest CSD for WebSphere BI Message Broker. All CSDs can be downloaded from the IBM Web site:

<http://www-306.ibm.com/software/integration/mqfamily/support/summary>

## Security requirements

On completion of the installation of the product and the latest CSD, perform the following steps:

1. Start the Security Wizard as illustrated in Figure 7-17 on page 224 by clicking **Start** → **Programs** → **IBM WebSphere Business Integration Message Brokers** → **Security Wizard**.

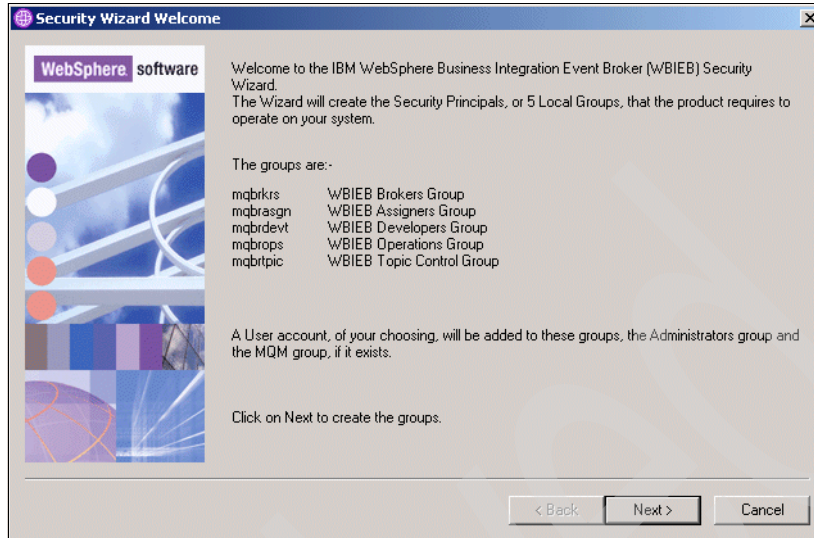


Figure 7-17 Security Wizard

2. Clicking **Next** displays the Security Wizard User Select and Create page as illustrated in Figure 7-18 and enter the name of the user. In our case, the user ID was db2admin. This user ID is used to run the runtime components of WebSphere BI Message Broker.

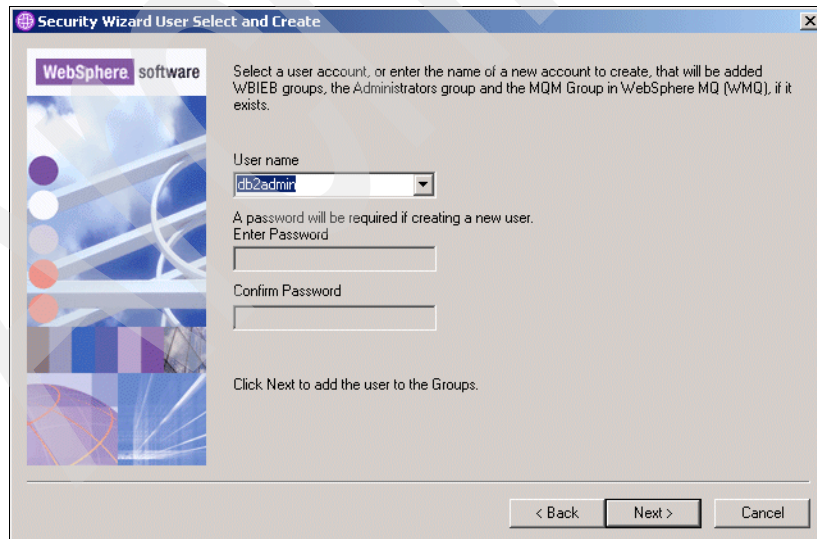


Figure 7-18 User Select and Create



3. Click **Next** to complete the Security Wizard, which creates the user ID if it was not present and adds the user ID to the following groups in Table 7-3.

Table 7-3 Required Security group

Group Name	Description
mqrbrks	IBM WBIMB Brokers Group
mqrbrasn	IBM WBIMB Assigners Group
mqrbrdevt	IBM WBIMB Developers Group
mqrbrps	IBM WBIMB Operations Group
mqrbrtpic	IBM WBIMB Topic Control Group
mqrbrmqm	IBM MQSeries Administration Group
Administrators	Windows Administrators Group

Though we do not use the broker installation on the Windows server, we did install all the available components.

**Tip:** Before continuing with setting up the components to interact with the Linux based brokers, you might want to create the default configuration. Creating the default configuration will give you a working configuration with all of the latest documentation available from the toolkit.

The default configuration wizard creates a configuration manager and a broker, then connects them to the workbench so that WebSphere Business Integration Message Broker resources can be deployed to it.

The benefit is that the Getting Started Wizard creates the necessary WebSphere MQ, DB2 and Windows resources required by the configuration manager and broker.

To access it, start the Message Brokers Toolkit for WebSphere Studio - Message Broker and select **Create a Default Configuration** from the Welcome page as illustrated Figure 7-19 on page 226. To reopen the Welcome page, select **Help** → **Welcome**.



Figure 7-19 Default configuration

## Installing the broker on Linux

The broker is the component that actually processes message data. It runs on the Windows, UNIX, and z/OS platforms. Following steps are used to install WebSphere BI Message Broker on Linux.

### Security considerations

The broker processes run with the authority of a certain user and this user ID should be a member of a group called `mqbrkrs`. Also, this user ID should have sufficient administrative authority to work with WebSphere MQ objects. The following steps define the user ID `mqsiuser` and add it to the groups `mqbrkrs` and `mqm`. Also the root user ID is added to this group `mqbrkrs`.

1. Enter the following command to create a user group with the name `mqbrkrs`:
2. Edit the file `etc/group` to add the user ID `root` to the `mqbrkrs` group.
3. Enter the following command to create a user ID and add it to the groups `mqbrkrs` and `mqm`:

```
useradd -G mqbrkrs,mqm -c "WBIMB User" mqsiuser
```

**Important:** The `useradd` command creates a locked account by default. Enter the following command to unlock the account to enable it to be used:

```
passwd mqsiuser
```

## **Linux installation steps**

Following are the steps to install WebSphere BI Message Broker on Linux:

1. Logon as root.
2. Change to the directory where installables are present and execute:  

```
./setuplinuxia32
```
3. After initializing the wizard, select the language to be used during the installation.
4. The Welcome panel is shown. Click **Next** to continue.
5. The following steps ask you to confirm that all preparation steps have been performed for migration purposes. Select the **Yes** radio button and click **Next** to continue.
6. After accepting the licence agreement, the installer will verify that software requirements are met. Click **Next** to continue.
7. You can select either a typical or custom install. If you select **custom** install, you can specify which components you want to install in the next step.
8. Select the components you want to install. You will only see this panel if you selected custom install in the previous step.
9. The installer will then present you with an installation summary. Click **Next** to continue.
10. When the install is complete, click **Finish**.

## **Installation verification**

Before starting to use WebSphere BI Message Broker, the following steps should be completed:

1. Execute `.profile` by using command  

```
./opt/mqsi/samples/profile/profile.lin
```

**Attention:** Be sure to include the dot (.) at the start of the command.

This command sets all the environment variables required for working with WebSphere BI Message Broker.

2. Execute the `mqsisetcapacity -c <number of processors>` command. This command sets the licensed processor count.
3. Execute the `mqsilist` command to list all the broker resources present on the server as shown in the example Example 7-3 on page 228. At this time, no resources will be listed. However, the successful execution of `mqsilist` indicates that the environment is set up correctly.

```
mqsilist  
BIP80711: Successful command completion.
```

---

## 7.4 Implementation steps

In this section, we review the application scenario for this chapter. Message generators running on hosts `wmq1` and `wmq2` send messages to hosts `brk1` and `brk2` through MQ cluster channels. The messages are processed by message brokers running on the hosts `brk1` and `brk2`. The message processing consists of storing the incoming messages in an external database. This database will be used by the Web-based message retrieval application.

The WebSphere MQ setup for this scenario was discussed in the previous chapter. The only difference for clustering with message brokers is to redefine the cluster queues as local queues, as discussed in 7.1.1, “Using brokers with WebSphere MQ clustering for high availability” on page 200.

1. Message brokers rely on the services of a database system to manage persistent data. We need to create three databases:
  - We will need to create this database on the database server, which is also the server that hosts WebSphere Application Server. A single database can be used for both brokers.
  - A second database needs to be created to store the application data, the messages generated by the message generator application. This second database is also created on the server used by WebSphere Application Server.
  - A third database needs to be created for use by the configuration manager. We choose to use a local database on the Windows server that is also used to run the configuration manager.
2. The next step is to develop the message set. Based on the message layout that is used by the message generators, we will define the message definition through the Message Brokers Toolkit. This definition includes the logical format, what elements and the physical format. Is it XML or delimited, for example?
3. After creating the message set, we can develop the message flow, which specifies what kind of processing should be performed on each incoming message.
4. The development in the Message Brokers Toolkit is then finished and we can complete the setup of the runtime environment. A broker instance needs to be created and connected to the broker domain. We can then assign the

message flow and, by implication, the message set to an execution group of this new broker on Linux.

## 7.5 Creating databases and tables

DB2 UDB V8 is installed on the host was and on a Windows machine used for the configuration manager.

### 7.5.1 Creating databases on the WebSphere Application Server

Logon to the WebSphere Application Server and switch to the user ID for the database instance owner, for example `db2inst1`. Open a DB2 command session by entering `db2` at the command line.

1. To create the broker database, enter the following at the db2 command prompt:

```
create database WBIBKDB
```

2. When the database is created, connect to it by entering:

```
connect to WBIBKD
```

3. Bind the DB2 CLI interface to this new database by using the command:

```
bind sqllib/bnd/@db2cli.1st grant public CLIPKG 5
```

4. End the DB2 command line environment by typing `quit`.

**Note:** The same database can be used by multiple brokers.

Tables and other database objects are added to this database during broker creation.

We can now create the second database, which will be used to store the generated messages and by the Web-based message retrieval application.

5. Open the DB2 command session and enter the commands shown in Example 7-4. After creating the database, we connect to it and create three tables.

*Example 7-4 Creating RESDB and tables*

```
create database RESDB
connect to RESDB
CREATE TABLE CPU(TIME TIMESTAMP NOT NULL, RESOURCE VARCHAR(3), USAGE INTEGER,
QMGR VARCHAR(20), CONSTRAINT PRIMARY KEY(TIME));
```

```
CREATE TABLE MEM(TIME TIMESTAMP NOT NULL, RESOURCE VARCHAR(3), USAGE INTEGER,  
QMGR VARCHAR(20), CONSTRAINT PRIMARY KEY(TIME));  
CONNECT RESET  
QUIT
```

---

## 7.5.2 Database setup on Windows win6336

This Windows machine will perform two functions:

- ▶ Host the configuration manager
- ▶ Develop message flows and message sets using the Message Brokers Toolkit.

During the development of these components, we will need client access to the application database RESDB, created in the previous step.

Creating a DB2 client connection can be performed in many ways. Example 7-5 shows the commands that can be entered in a DB2 command window to set up such a connection.

Start a DB2 command window by selecting **Start** → **Programs** → **IBM DB2** → **Command Line Tools** → **Command Window**.

*Example 7-5* Client database setup on host win6336

---

```
attach to db2 user db2admin using <Windows password>  
catalog tcpip node student2 remote student2 server 50001  
catalog database RESDB as RESDB at node student2 authentication server  
connect to RESDB user db2inst1 using <Linux password>  
quit
```

---

Besides a client connection to the application database, we also need to create a database for use by the configuration manager. We could have added the configuration manager database to the same database server used by the broker and WebSphere Application Server. However, there will only be one user of this database, which is the configuration manager. Therefore, we decided to create a local database specifically for the configuration manager.

1. Open a DB2 Command Window.
2. Attach to the DB2 instance using the instance ID db2admin:  

```
db2 attach to db2 user db2admin using <Windows password>
```
3. Create the database for the configuration manager:  

```
create database WBICMDB
```

### 7.5.3 DB2 client setup on hosts brk1 and brk2

The message brokers need a client connection to their own database, WBIBKDB, and also to the application database RESDB. The message flow needs such a connection to insert the incoming message data in the correct table.

The creation of such a client connection is similar to the commands shown in Example 7-5 on page 230. Example 7-6 shows the commands performed in a DB2 command window on the Linux machines.

*Example 7-6 Client database setup*

---

```
attach to db2 user db2inst1 using <Linux password on brk1/brk2>
catalog tcpip node student1 remote student1 server 50001
catalog database RESDB as RESDB at node student1 authentication server
catalog database WBIBKDB as WBIBKDB at node student1 authentication server
connect to WBIBKDB user db2inst1 using <Linux password on was>
connect to RESDB user db2inst1 using <Linux password on was>
quit
```

---

The two **connect** commands are used so that we can access these databases from the hosts brk1 and brk2.

## 7.6 Design and development of the message set

In this scenario, we define the message that contains the details of the server resource usage to the broker. This message is produced by the message generating application.

### 7.6.1 Designing the message set

Example 7-7 shows the format of the message generated by message generating application.

*Example 7-7 Sample message from the application*

---

```
cpu,2004-04-03-18.48.00,qmgr1,99
```

---

Because the message generated is comma-separated values, we choose *Tagged Delimited String* (TDS) as the physical format (or *wire* format). The message is basically composed of a single complex type consisting of a four simple elements.

**Note:** Physical format describes the precise appearance of the message bitstream during transmission. WebSphere BI Message Broker supports various physical formats such as XML, TDS, and CWF( Custom Wire Format). Each physical format is explained in detail in WebSphere BI Message Broker Help. See **Concepts** → **Message modeling**).

## 7.6.2 Developing the message set

We created a message set project and a message set for modelling the message generated by the message generation program. To create the message set project and message set, use the following steps:

1. Start the Message Brokers Toolkit by selecting **Start** → **IBM WebSphere Business Integration Message Brokers** → **Message Brokers Toolkit**.
2. Switch to the Broker Application Development perspective by selecting **Window** → **Open perspective** → **Broker Application Development**.
3. Launch the new message set project wizard by selecting **File** → **New** → **Message Set Project**.
4. Provide a name for the project, for example `ServerResourceMessageSetProject` as shown in Figure 7-20 and click **Next**.

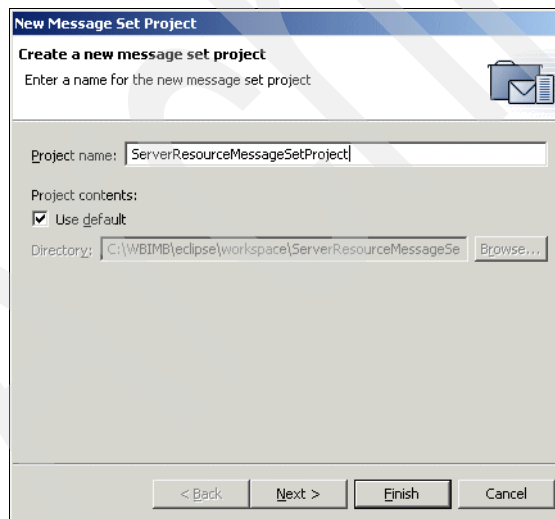


Figure 7-20 Create a new message set project

5. In the next step, provide a name for the message set itself, for example `ServerResourcesMessageSet` as shown in Figure 7-21 on page 233 and click



**Next.** We chose to create a message set without namespace for simplicity and therefore left the use of namespaces option unchecked.

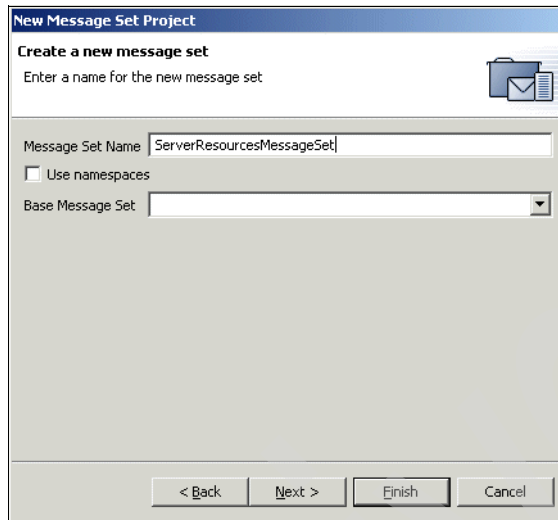


Figure 7-21 Create a new message set

6. Check the option **Tagged/Delimited String Fformat** boc and provide a name for this format. The default value TDS1 is chosen in Figure 7-22. Click **Finish**.

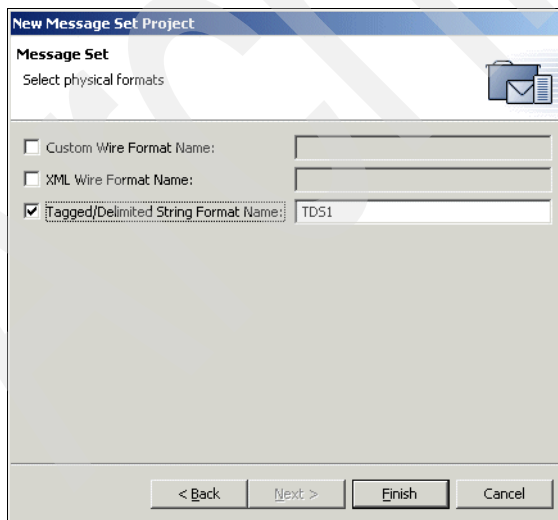


Figure 7-22 Select physical formats

The wizard completes the creation of message set project with a message set. The message set editor will open the message set file as illustrated in Figure 7-23.

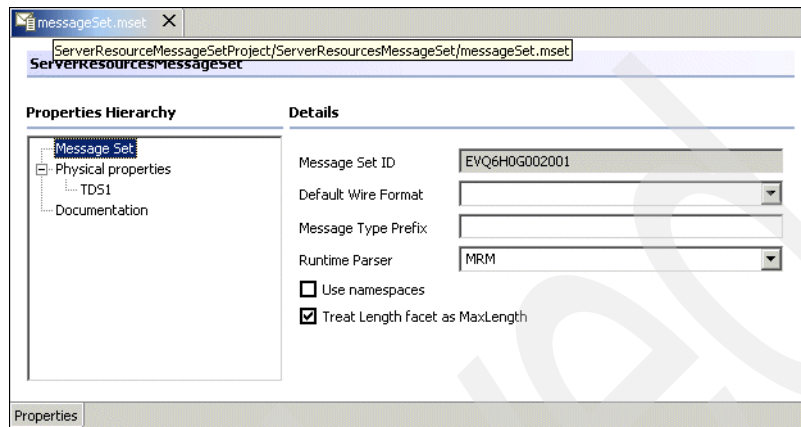


Figure 7-23 Properties of message set

7. In the properties hierarchy, under Physical properties, click **TDS1** to change the default properties of physical format. In this scenario, we modified the value of the property Datetime settings for the TDS wire format. See Figure 7-24 on page 235.
8. Select the radio button **Use default dateTime format**, enabling the editing of the dateTime field text box. Change the Datetime format to yyyy-MM-dd-HH.mm.ss format and save it. This format matches the format shown in Example 7-7 on page 231.
9. In the same list of properties, you can also see the value for the property Delimiter. Set this to a comma (,), the delimiter used by the message generation application.

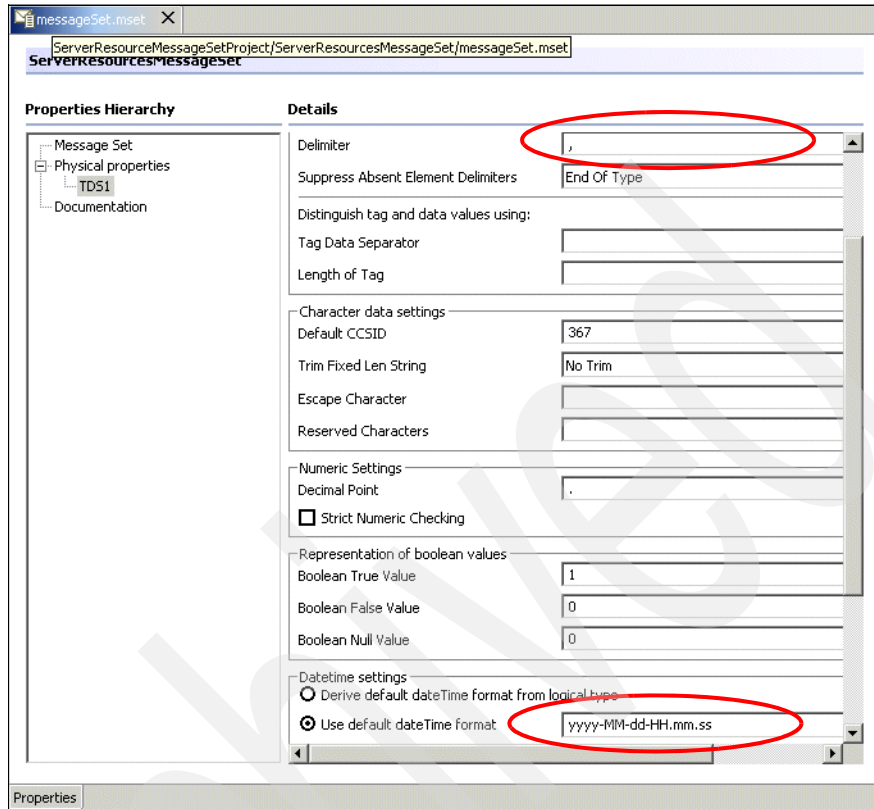


Figure 7-24 TDS properties of message set

### 7.6.3 Creating the message definition file

Within the message set, we can now define a message definition file.

1. Select the **ServerResourceMessages** message set, then right-click and select **New** → **Message Definition File**.
2. A message can be defined starting with a resource definition from another environment, for example a COBOL copybook. In this case, we do not have a source. Select **Next**, as shown in Figure 7-25 on page 236.

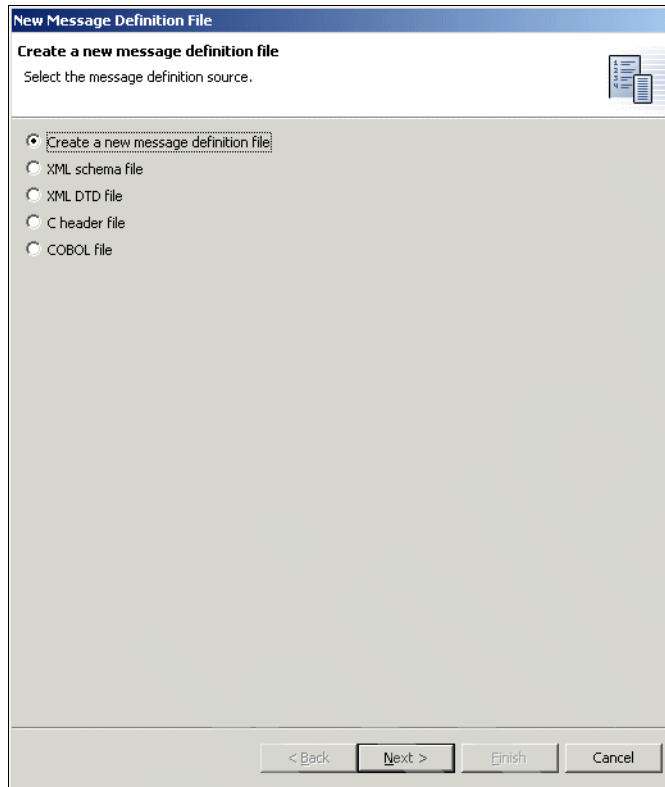


Figure 7-25 Message definition: Step 1

3. Select the **ServerResourceMessageSet** and provide a filename, for example ResourceUsage. Click **Next**. See Figure 7-26 on page 237.

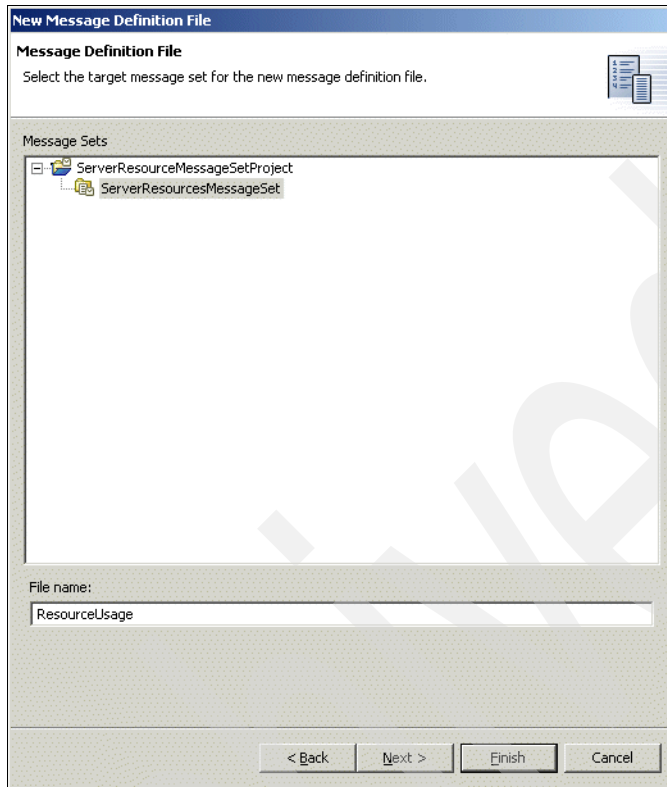


Figure 7-26 Message definition name

4. Because we are not using namespaces, leave all the default values and click **Finish**. See Figure 7-27 on page 238.

**New Message Definition File**

**Message definition namespace**  
Specify the namespace for the new message definition contents.

Schema for Schema settings

Prefix:

Namespace:

Use target namespace

Target Namespace settings

Prefix:

Namespace:

< Back   Next >   Finish   Cancel

Figure 7-27 Message definition namespace

5. The message definition file ResourceUsage.mxsd will be created and the message definition editor will open by default as illustrated in Figure 7-28 on page 239.

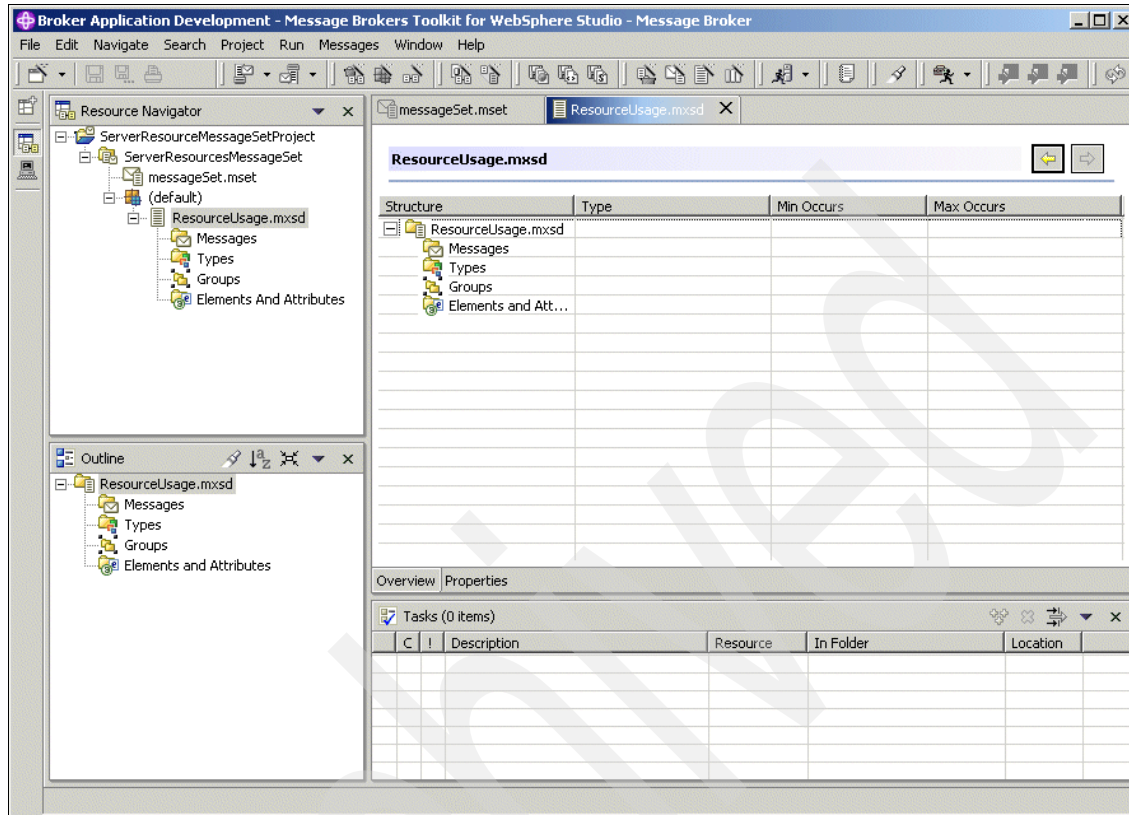


Figure 7-28 Message definition editor

## 7.6.4 Creating the logical structure of the message

The message used in this scenario has four elements shown in Table 7-4. All elements are of the simple type . The complete message ResourceUsage itself is an element of complex type which consists of all the four elements described in the table.

Table 7-4 Elements of the message ResourceUsage

Element	Type	Comments
Time	datetime	Time when the Resource usage was recorded ex: '2004-04-26-23:59:59'
ResourceType	string	cpu, mem

Element	Type	Comments
ResourceUsage	long	in %
Qmgr	string	queue manager on which message generation application is running

The first step is to create the elements. Follow these steps to define the message:

1. In the message definition editor, right-click the folder **Messages** and select **Add Message**. Provide a name for the message, for example ResourceUsage. See Figure 7-29.

Creating a message creates also a complex type and a complex element. This can be seen by expanding the folder Types and Elements and Attributes as shown in Figure 7-29.

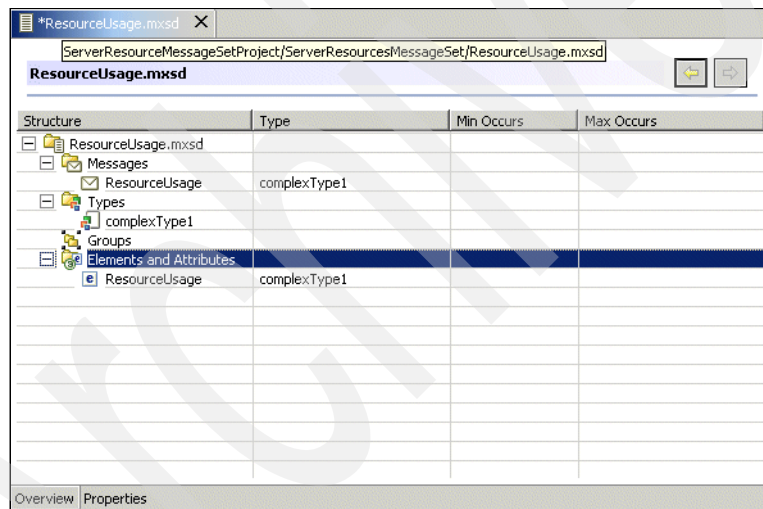


Figure 7-29 Message name, type and elements

2. Change the name of the type from complexType1 to ResourceUsageType. Right-click on the type **complexType1** and select **Rename**. Provide a better fitting name, such as ResourceUsageType as the name of the complex type. The message definition will now look as shown in Figure 7-30 on page 241.



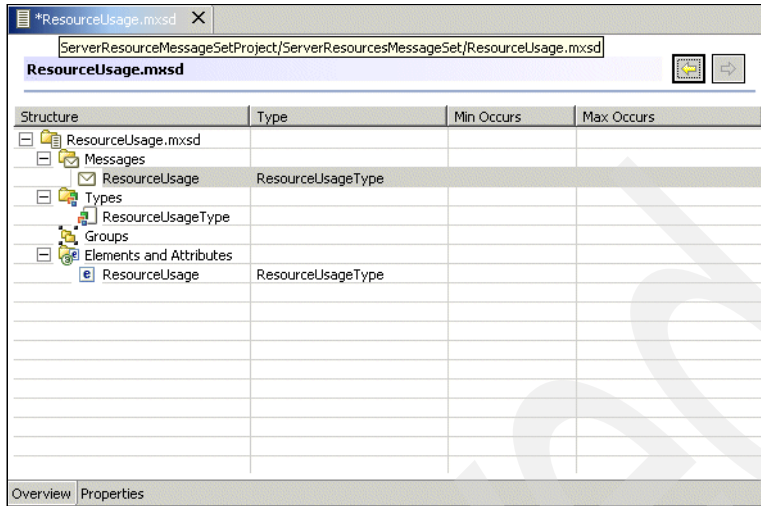


Figure 7-30 Renaming a complex type

- The next step is to define four elements. To define the first element, **Time**, right-click the message **ResourceUsage** and select **Add local element**. Provide a name for the element, for example **Time**. See Figure 7-30.

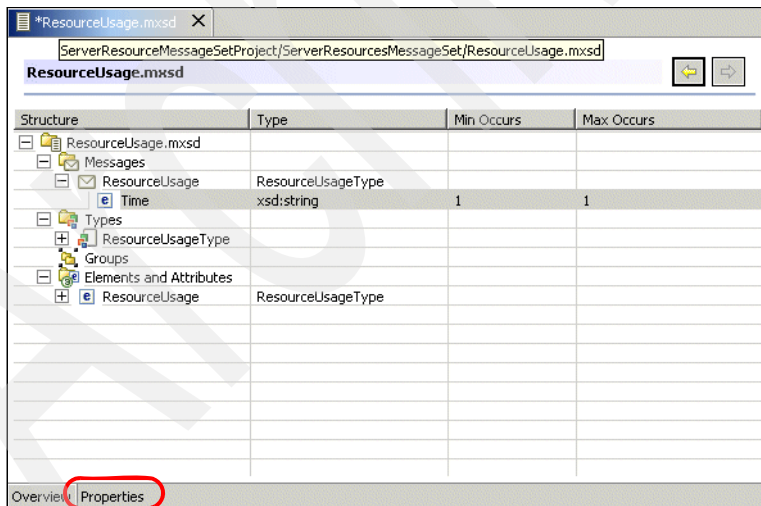


Figure 7-31 Changing the properties of local element

- By default, a local element of type String will be created. To change the type select the tab **Properties** as illustrated in Figure 7-32 on page 242.

In the properties tab, change the type of the element to **dateTime** as illustrated in Figure 7-32.

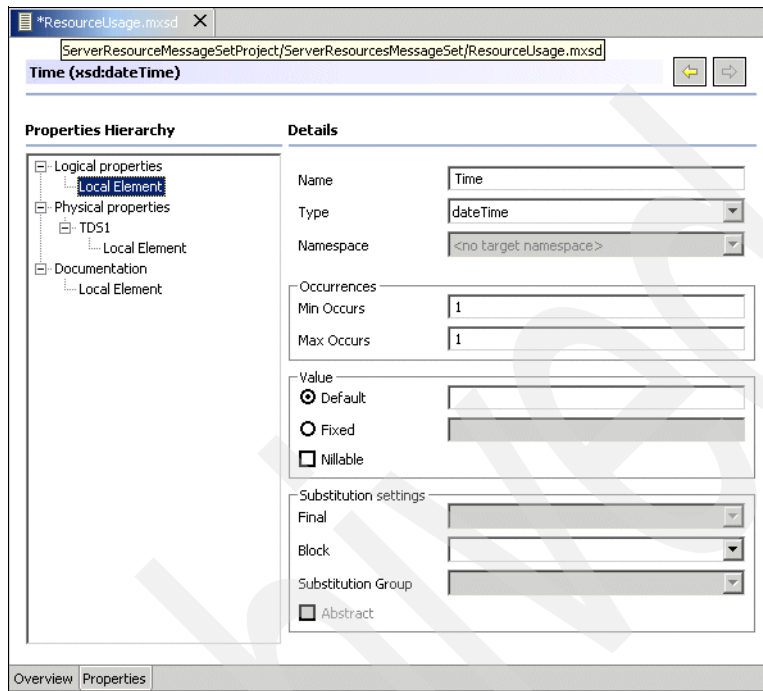


Figure 7-32 Changing the type of simple element

5. Select the Local Element present under TDS1 and change the Length field to 19 as illustrated in Figure 7-33 on page 243.

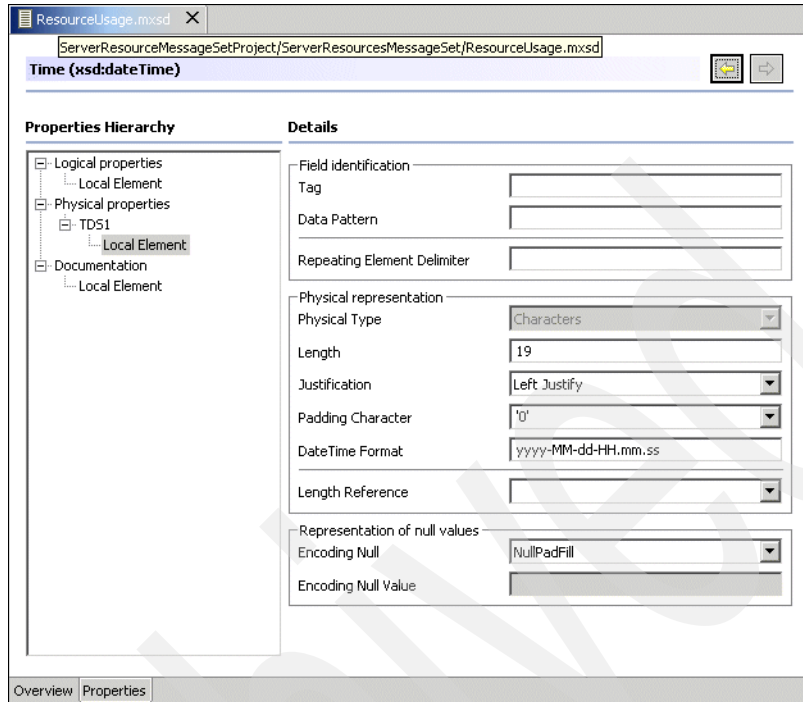


Figure 7-33 TDS properties of Time field

6. Repeat these steps to create the other three elements, Resource Type with length 3, ResourceUsage with length 2 and Qmgr with length 20. The message definition will now look as shown in Figure 7-34 on page 244.

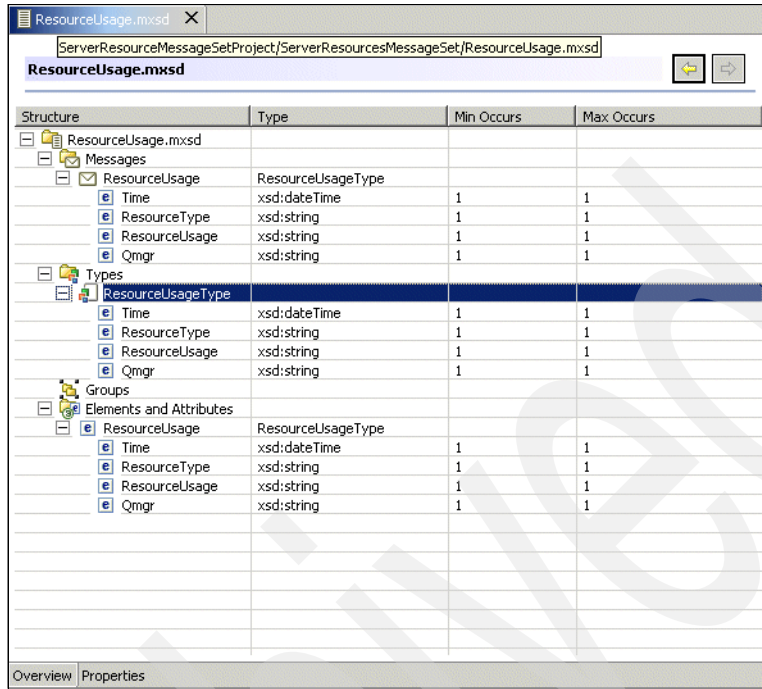


Figure 7-34 Message definition with all elements defined

- The order of the elements within the type and message is important. If the message is XML, then the element can be identified by its tag. For a delimited message, the only indication is the real order of the elements. If the order of the elements does not match with the order in the sample message (shown in Example 7-7 on page 231), then drag the elements around until it matches, as shown in Figure 7-35 on page 245.

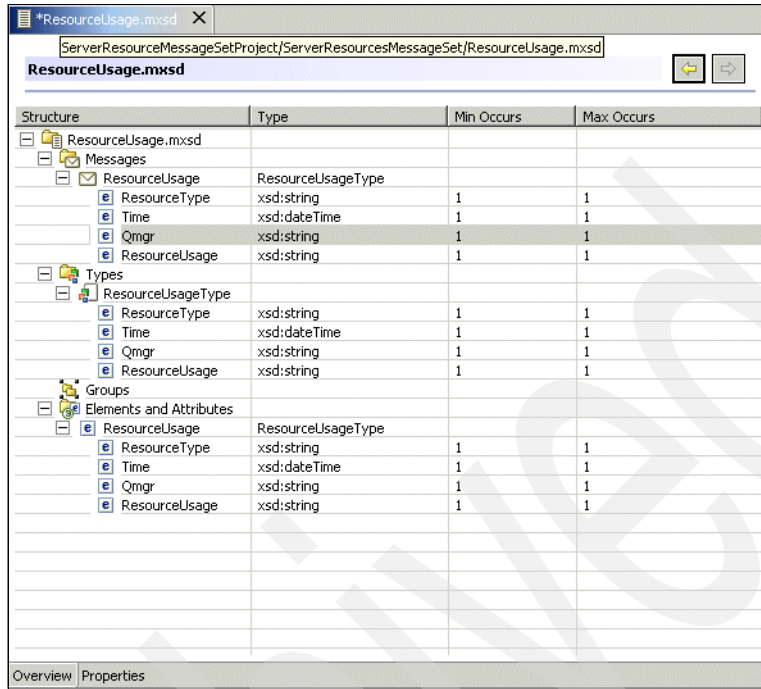


Figure 7-35 Elements reordered in message

8. One final step is to set the physical properties of the type **ResourceUsageType**. Select **ResourceUsageType** listed under the Types folder. Select the **Properties** tab. In the left section, select **Complex Type** for physical format **TDS1**.
9. Change the property Data Element Separation to All Elements Delimited, as in Figure 7-36 on page 246.
10. The property Delimiter should already be set to a comma. This value is inherited from the message set.
11. Save the definition and close the message editor.

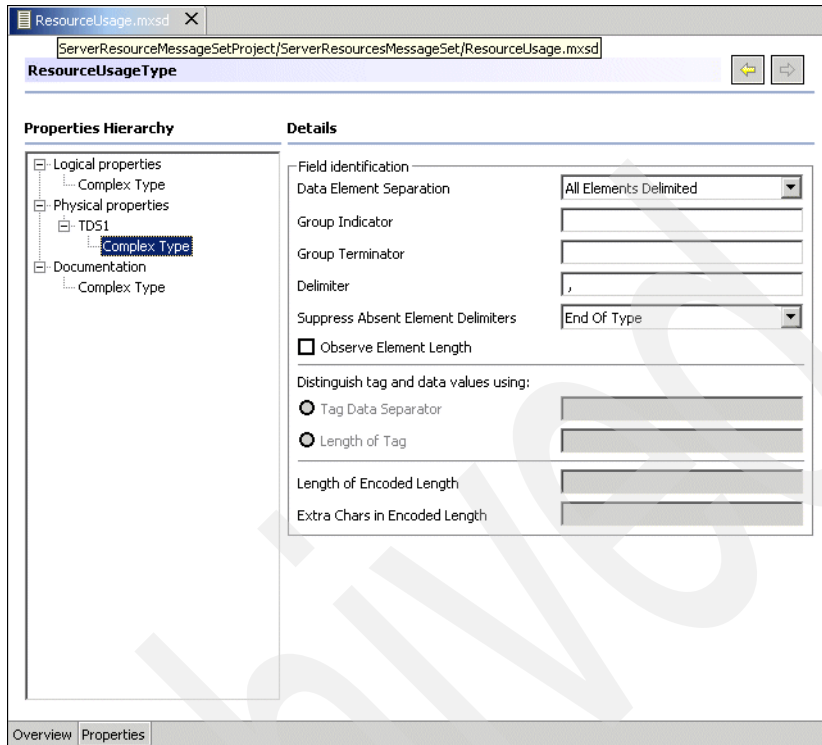


Figure 7-36 Set the physical properties of the type

## 7.7 Design and development of message flow

Now that the message model is defined, we can proceed by creating a message flow that specifies the processing of the messages by the broker.

### 7.7.1 Message flow design

The message flow is designed to take the input message as a common separated string. The message flow is designed to parse the input message and insert the data to the appropriate table in the database. The message flow includes an MQInput node, MQOutput nodes, a Trace node, a Filter node and Database Insert nodes. The complete message flow is illustrated in Figure 7-37 on page 247. The following subsections describe the development process of this message flow.

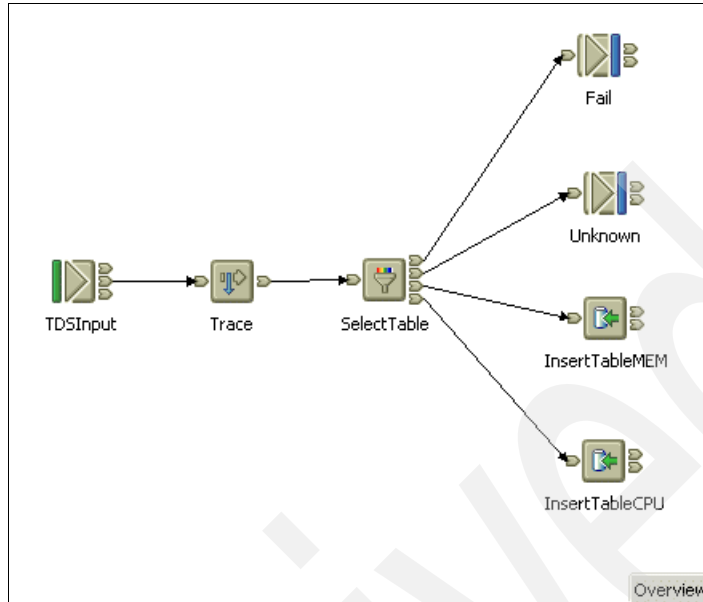


Figure 7-37 Message flow

## 7.7.2 Message flow development

In this section we describe the steps required to create a message flow project and the message flow.

### Create a message flow project

To create a project, perform the following steps:

1. Start the Message Brokers Toolkit by selecting **Start** → **IBM WebSphere Business Integration Message Brokers** → **Message Brokers Toolkit**.
2. Switch to the Broker Application Development perspective by selecting **Window** → **Open Perspective** → **Broker Application Development**.
3. Launch the new Message Flow project wizard by selecting **File** → **New** → **Message Flow Project**.
4. Provide a name for the message flow project, for example `ServerResourceMessageFlowProject` as shown in Figure 7-38 on page 248 and click **Next**.

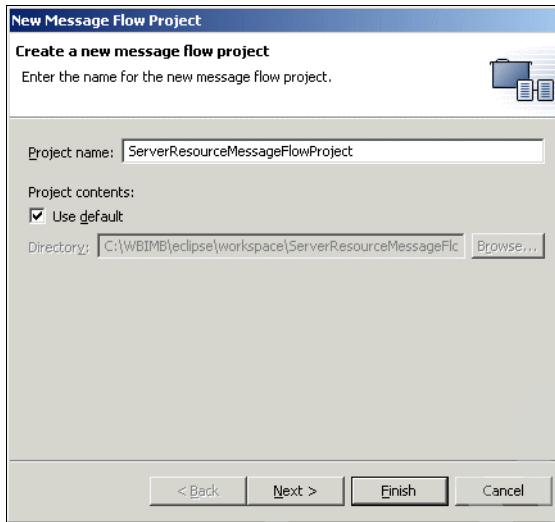


Figure 7-38 Create message flow project: Step 1

5. Select the **ServerResourceMessageSetProject** under referenced Projects as shown in Figure 7-39 and click **Finish**.

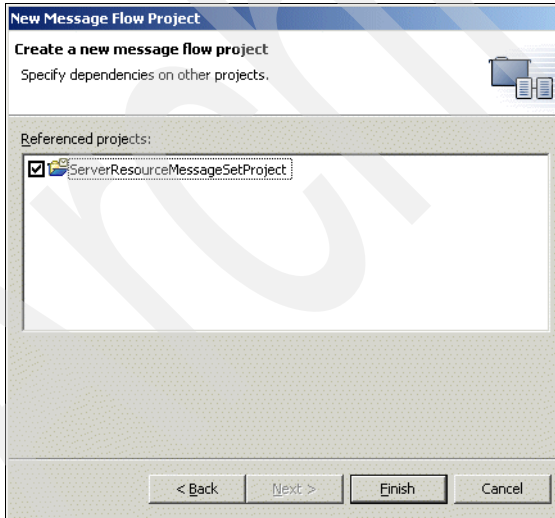


Figure 7-39 Create message flow project : Step 2

## Create a message flow

Now that we have a project to hold a message flow, we can proceed by creating the message flow itself.



1. In the Resource Navigator view, right-click the message flow project that we just created and select **New** → **Message Flow**.
2. In the next dialog box shown in Figure 7-40, provide a name for the message flow, for example ResourceUsageMsgflow. Leave the Schema value blank and click **Finish**.

A new file ResourceUsageMsgflow.msgflow will appear under the folder ServerResourceMsgflowProject in Resource Navigator view.

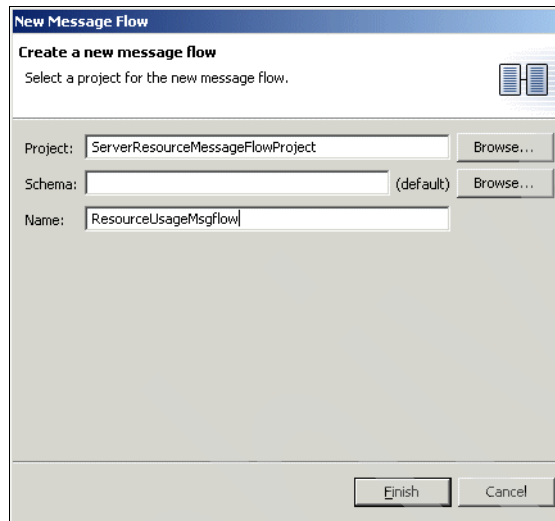


Figure 7-40 Create a new message flow dialog box

3. The message flow editor should be opened automatically as illustrated in Figure 7-41 on page 250.

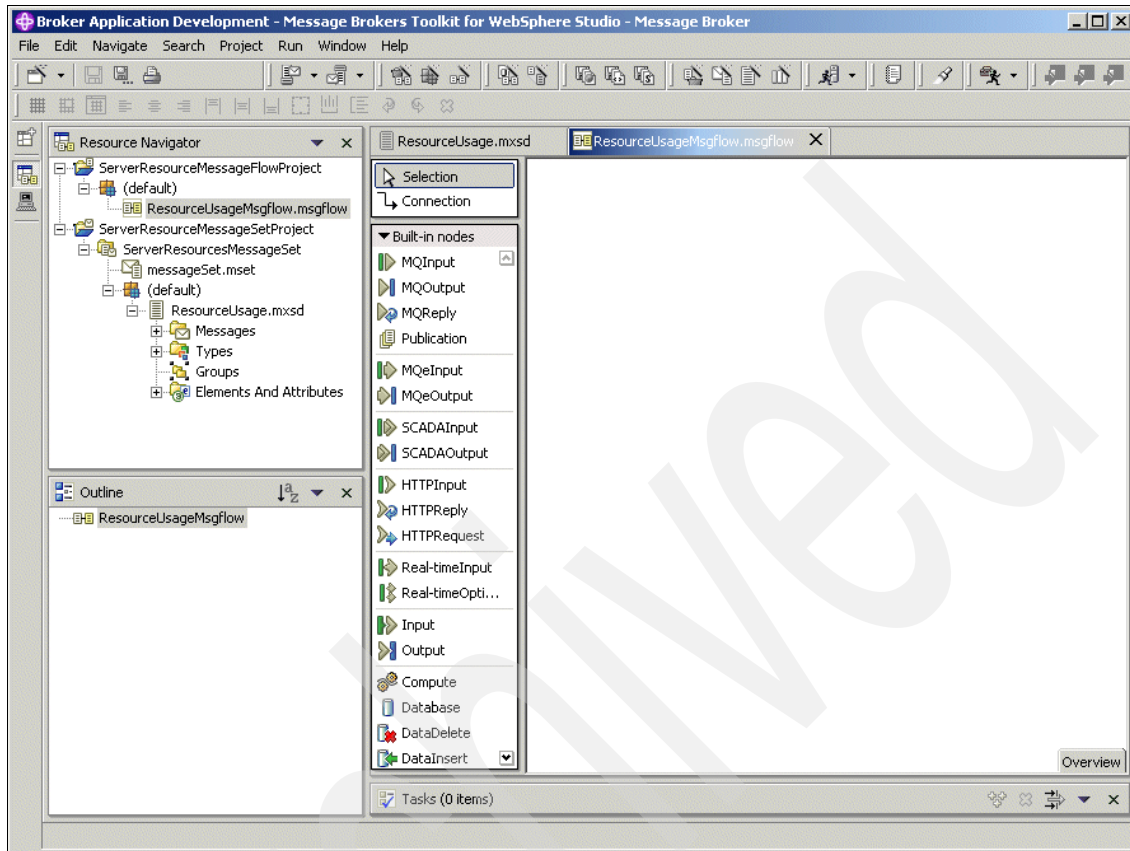


Figure 7-41 Message flow editor

### Add nodes to the message flow

Message flow development usually starts by adding one, more, or all required message nodes to the message flow. Nodes can then be connected to each other to perform the required logic. Finally, nodes can have one or more properties that need to be customized. Some nodes require some programming logic expressed in the ESQL programming language.

Add the following nodes to the canvas:

- ▶ MQInput node
- ▶ Trace node
- ▶ Filter node
- ▶ Two MQOutput nodes
- ▶ Two DataInsert nodes

**Tip:** Adding nodes to the canvas is performed using a click-and-click technique:

1. Select the required node in the palette.
2. Click the desired location on the canvas where the node is to appear.
3. An instance of the node will then appear on the canvas in approximately the selected location as illustrated in Figure 7-42.

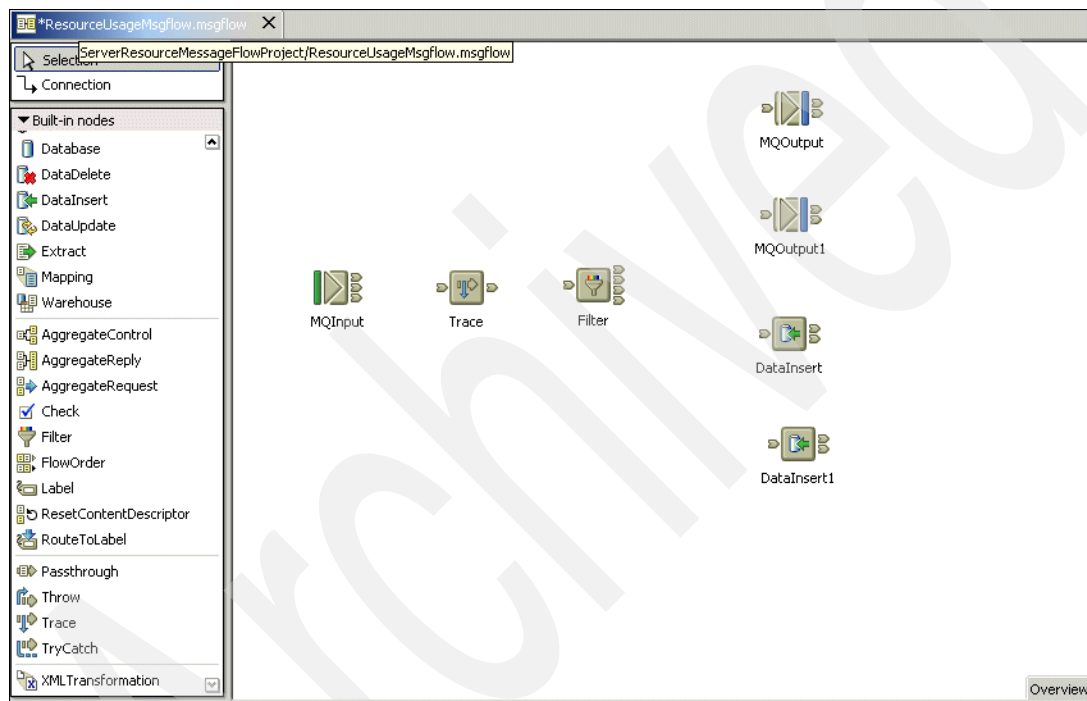


Figure 7-42 Populated canvas with all required nodes

## Connecting the nodes

Nodes have to be connected to each other to perform the intended logic. Nodes have one or more input and output terminals. Connecting the out terminal from one node to the in terminal of the next node creates the connection. Note that you can have multiple out terminals and that each out terminal is not the same. You can have out terminals associated with normally completed or with failed operations of a node.

To connect the nodes, use the following steps:

1. Click **Selection** in the top-center of Figure 7-41 on page 250.
2. Right-click a starting node and select **Create Connection**.
3. A dialog box appears that lists the node's output terminals. Select an output terminal (which becomes the origin of the connection), and click **OK** to return to the editor view.
4. To complete the connection, position the mouse pointer over the desired destination input terminal of the target node and click that node.

The message flow with all the nodes connected is shown in Figure 7-43.

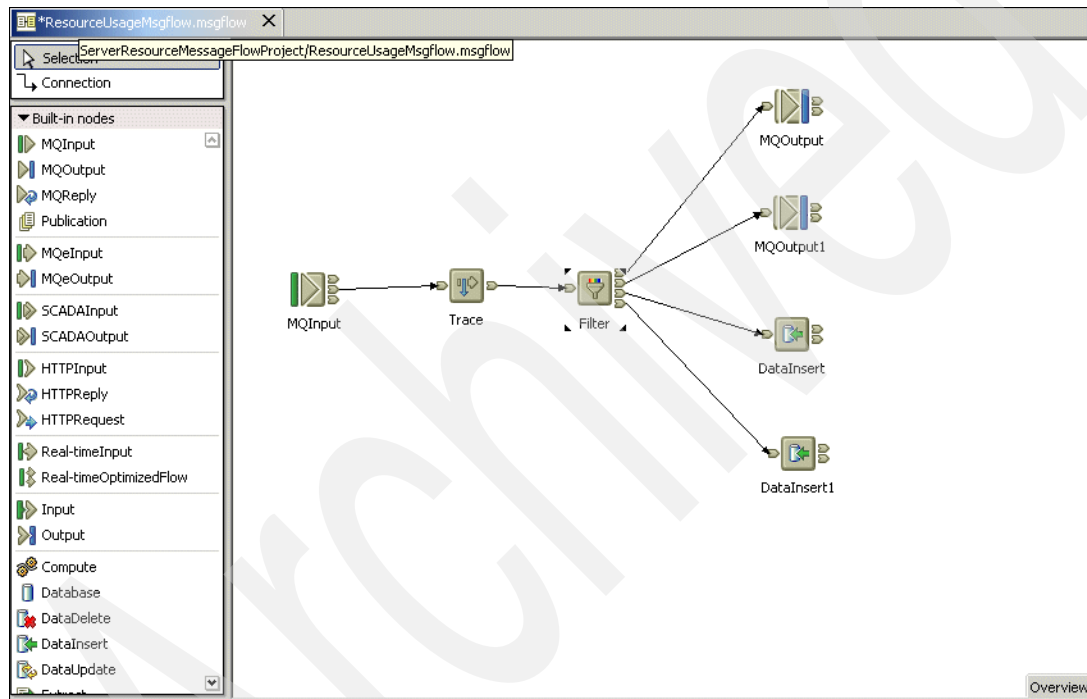


Figure 7-43 Nodes connected to each other

### Alternative views

An alternative way of looking at the connections between nodes is the Outline view, shown in Figure 7-44 on page 253.

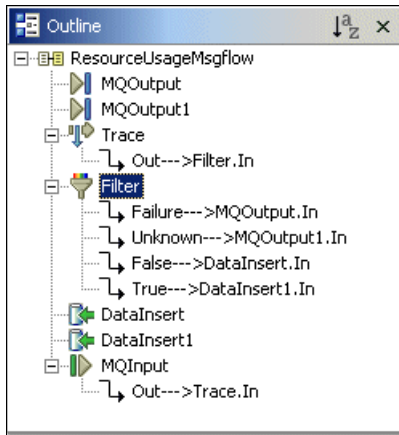


Figure 7-44 Outline view of the connected nodes

To see a neater view of the message flow, right-click any place in the message flow editor and select **Manhattan Layout**. The resulting formatting change to the message flow is shown in Figure 7-45.

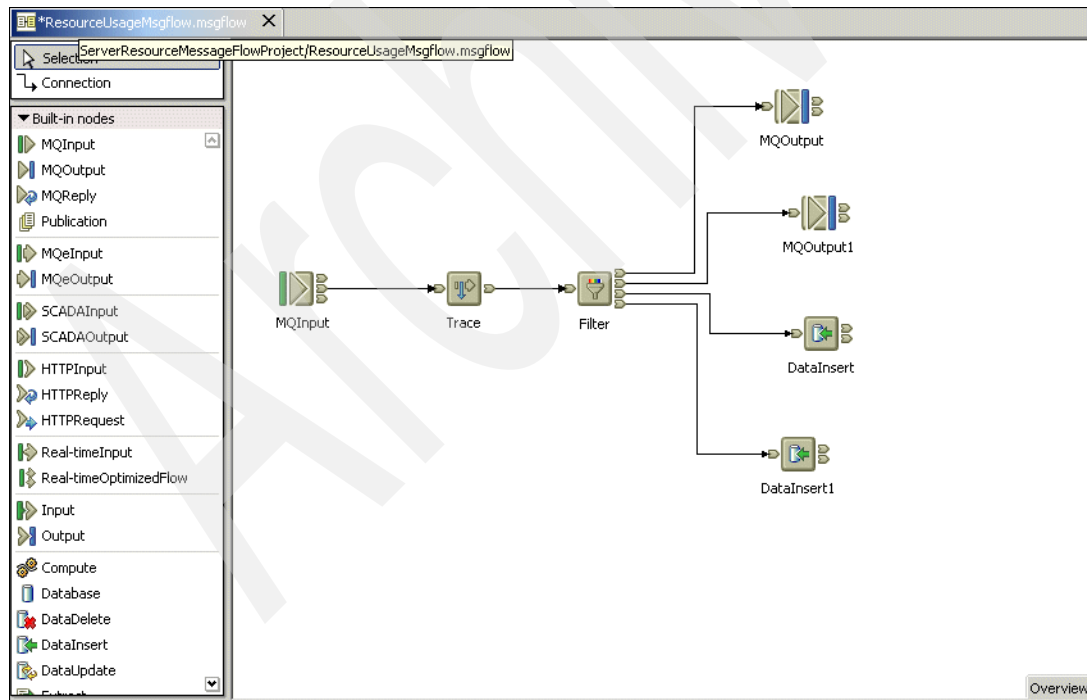


Figure 7-45 Message flow in Manhattan layout

The default names of the nodes do not clarify the intended function. Therefore, naming the nodes appropriately might help you to quickly understand a message flow. Right-click any node and select **Rename**.

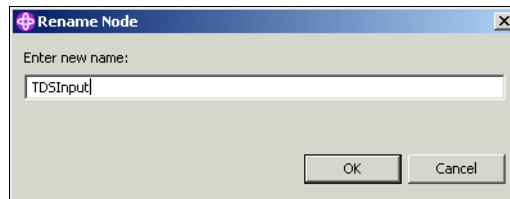


Figure 7-46 Rename node for functionality

Figure 7-47 shows the message flow with the nodes renamed to reflect the intended functionality.

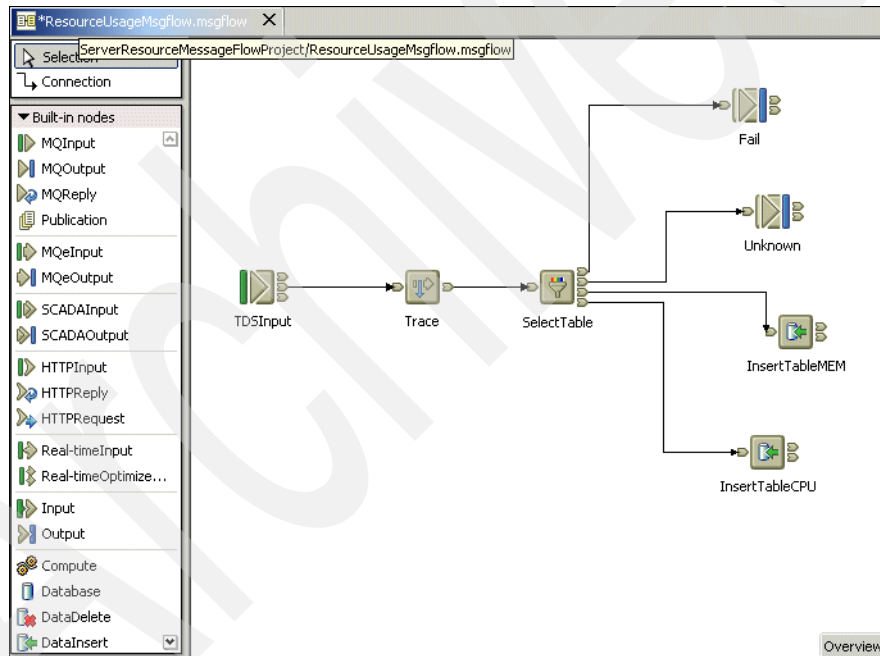


Figure 7-47 Message flow after renaming nodes

## Setting node properties

Several properties need to be set on the nodes in this message flow. The properties of a node are accessed by right-clicking on a node and selecting **Properties**.

### **Properties of the MQInput node**

The input of this message flow are the messages generated by the message generation application. This message is a delimited message that does not carry any header to detail the type or format of the message. Such a header, called MQRFH2, would limit the amount of information to be provided in the MQInput node. Since this header is not required, the message flow developer has to provide these details so that the broker knows how to handle any incoming message.

1. To set the properties of the MQInput node, right-click the node and select **Properties** (see Figure 7-48). In the section Basic, provide the name of the input queue, which is cq in our scenario.

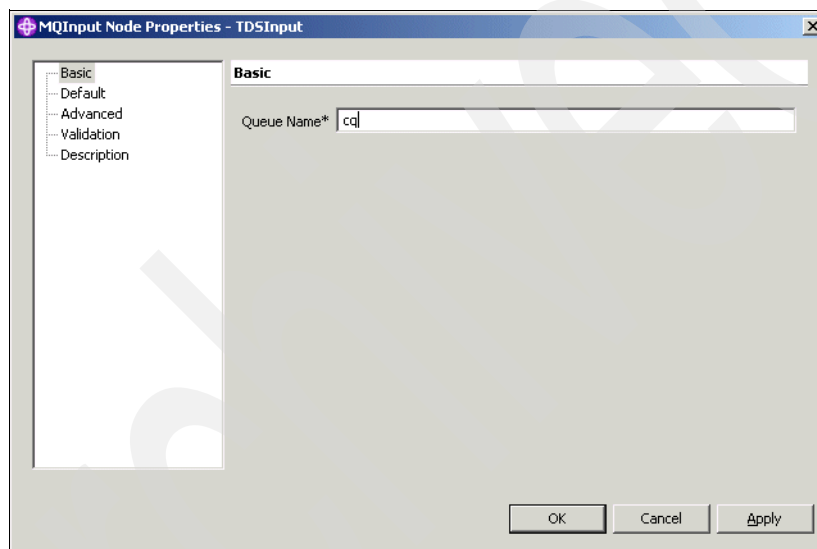


Figure 7-48 TDSInput Node properties

2. Select the **Default** section and provide details about the incoming messages, as shown in Figure 7-49 on page 256. In this window, we specify the Message Domain, which is MRM. Next, select the message set from the drop-down list. Note that the identifier is unique to our environment. The field Message Type should be set to the name of the message as defined in the message set, which is ResourceUsage. The Message Type tells the broker what the logical format of the message is. The Message Format field, selected from a drop-down list, provides the name of the physical format of the message. Here the Message format is TDS1.

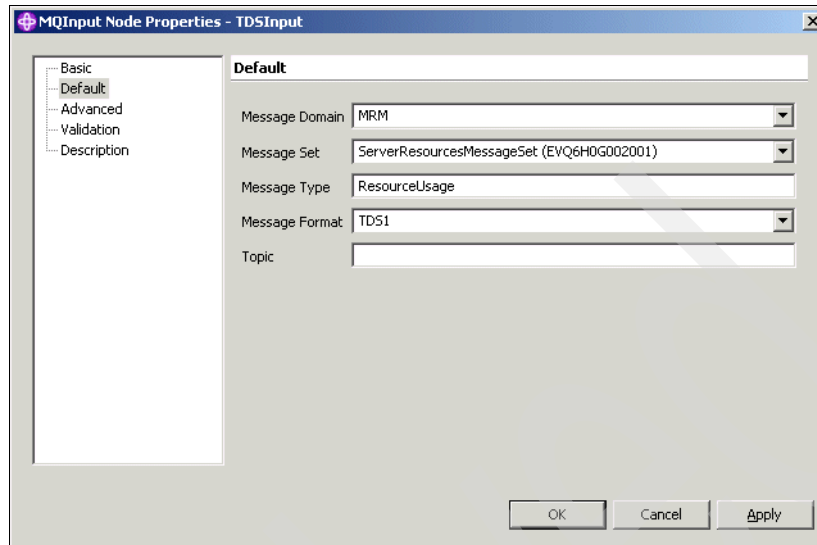


Figure 7-49 TDSInput node default property

### **Properties of the Trace node**

The Trace node is one way to debug a message flow and to obtain information at runtime about the processing of a message. The output of a Trace node can be redirected to a file on the file system, or to an operating system specific logging mechanism or to the user trace. The user trace is a broker trace that can be activated and deactivated at runtime.

Trace records generated by the Trace node can include any text, date and time information and also portions or the complete parsed message.

You can identify parts of the message to be written to the trace record, specifying the full field identifiers enclosed within the characters `{` and `}`. If you want to record the entire message, specify `{Root}`, as shown in Figure 7-50 on page 257.



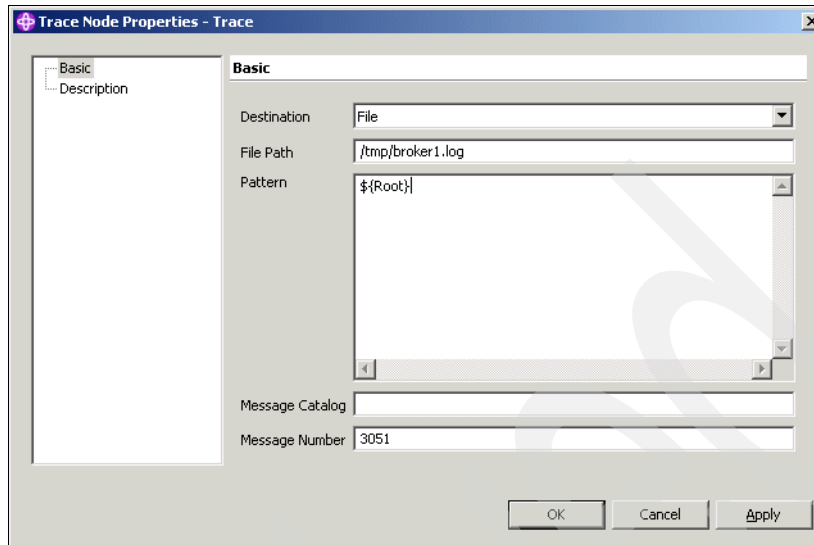


Figure 7-50 Trace node properties

### **Properties of the Filter node**

The Filter node routes a message according to the message content. You define the route by coding a filter expression in ESQL. The filter expression for our scenario checks the message field Resource, which can be mem, cpu, or disk.

Right-click the Filter node and select **Open ESQL**. A new editor opens for an ESQL file. The ESQL functionality used in our scenario is shown in Example 7-8. In the ESQL function, if the Resource field in the message contains the string cpu, then the message will be routed to TRUE terminal. If the Resource field in the message contains the string mem, then the message will be routed to FALSE terminal. Otherwise, the message will be routed to UNKNOWN terminal.

#### **Example 7-8 ESQL for Filter node**

```
CREATE FILTER MODULE ResourceUsageMsgflow_Filter
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
    IF Body.ResourceType like 'cpu' THEN
        RETURN TRUE;
    END IF ;
    IF Body.ResourceType like 'mem' THEN
        RETURN FALSE;
    END IF;
```

```
    RETURN UNKNOWN;  
  END;  
END MODULE;
```

---

**Note:** The Filter node is clearly best-suited for a traditional IF-THEN-ELSE programming construct. For the equivalent of a CASE programming construct, use the RouteToLabel and Label nodes. In the ESQL for a Compute node, you can populate an element called DestinationList. This list should contain the appropriate label (or case). Label nodes are added to the message flow for each possible value of the label. The RouteToLabel node typically follows the Compute node and jumps to the appropriate Label node that corresponds with the value in the element DestinationList.

### ***Properties of MQOutput nodes***

The MQOutput node is used to write a message to an MQ queue. The message flow uses two MQOutput nodes. One is used to write a message when the Filter node fails. The other one is used to write a message when the Filter node returns Unknown as the result of the filter. Set the properties of the two MQOutput nodes as illustrated in Figure 7-51 and Figure 7-52 on page 259.

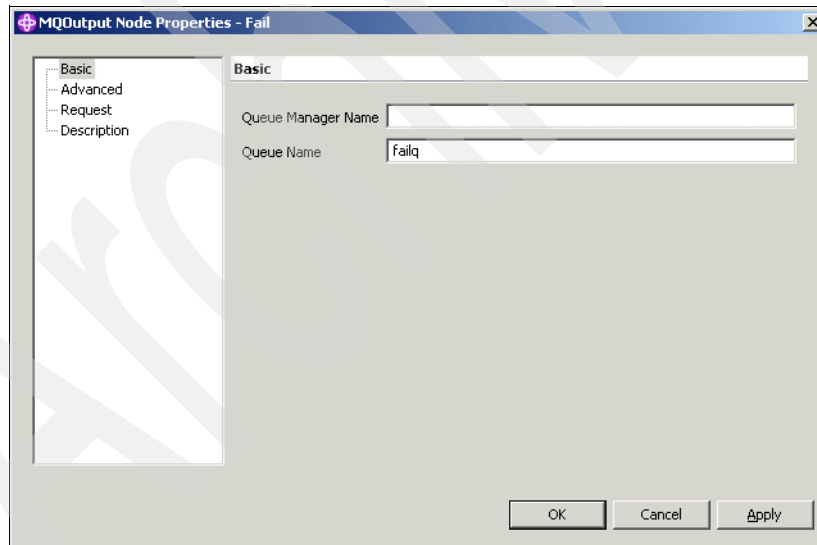


Figure 7-51 Fail MQOutput Node property

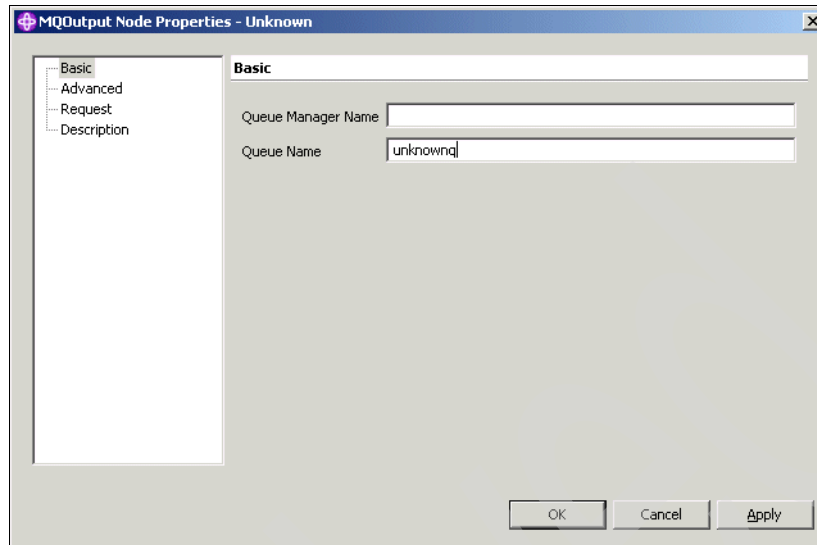


Figure 7-52 Unknown MQOutput Node property

## Properties of DataInsert nodes

A DataInsert node is used to insert records in a database table using parts of a message that hits the node at its in terminal. The database should be defined as an ODBC data source.

The DataInsert node is a specialized version of the Database node and can be used for all kinds of database interaction. The Database node itself can be considered a specialized version of the Compute node. In the end, everything that you can do with a DataInsert node or with a Database node can be performed as well using a Compute node. The only difference is that you need to write ESQL in a Compute node, while you have graphical tools to help you customize the Database and DataInsert nodes.

1. The first step is to set the database name in the properties of the DataInsert node. This name is actually the name of the ODBC data source. Set the properties of two DataInsert nodes as illustrated in Figure 7-53 on page 260 and Figure 7-54 on page 260.

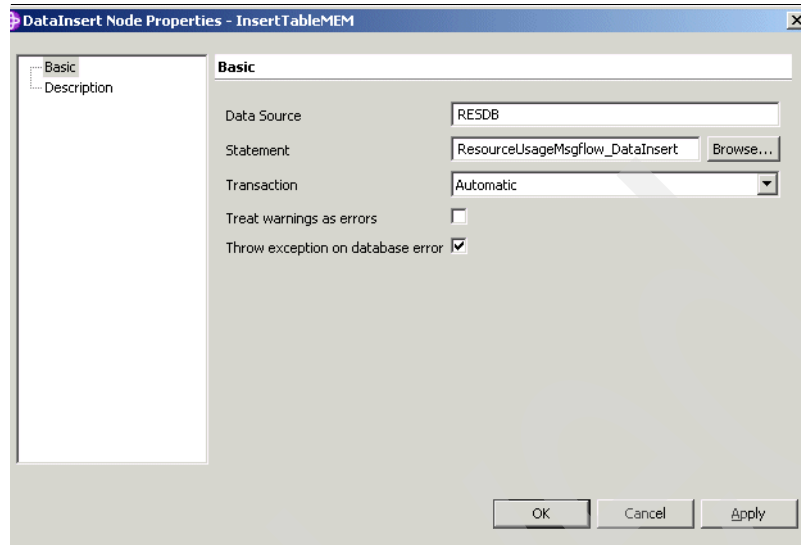


Figure 7-53 DatabaseInsert node properties

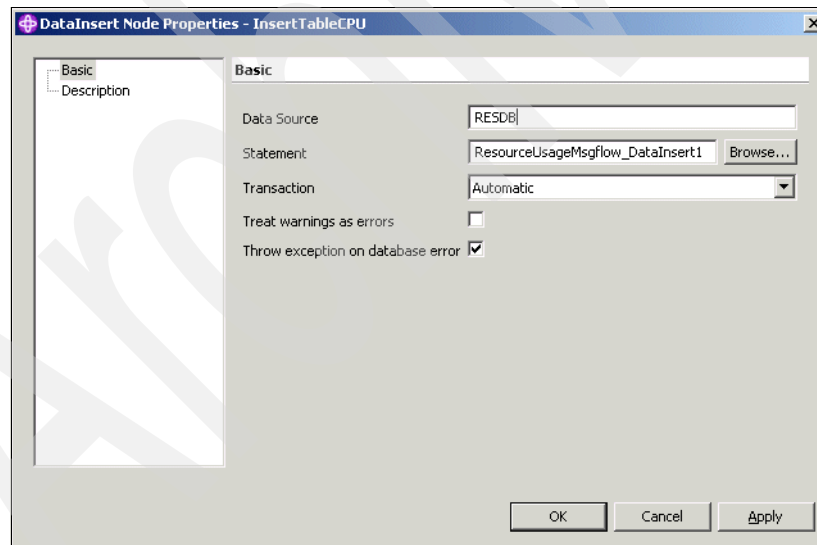


Figure 7-54 DatabaseInsert node properties

2. The next step is to define the mapping between the elements of the incoming message and the columns of the database table. In this scenario, the mapping is straightforward. We map the elements of the message to the columns in the database. The mappings are shown in Table 7-5 on page 261.

Table 7-5 Mapping between message elements and table columns

Message element	Column in table
Time	TIME
ResourceType	RESOURCE
Usage	USAGE
Qmgr	QMGR

To implement this mapping in the Message Brokers Toolkit, perform the following steps:

- a. Right-click the DataInsert node and select **Open Mappings**. The mapping editor will open as illustrated in Figure 7-55.

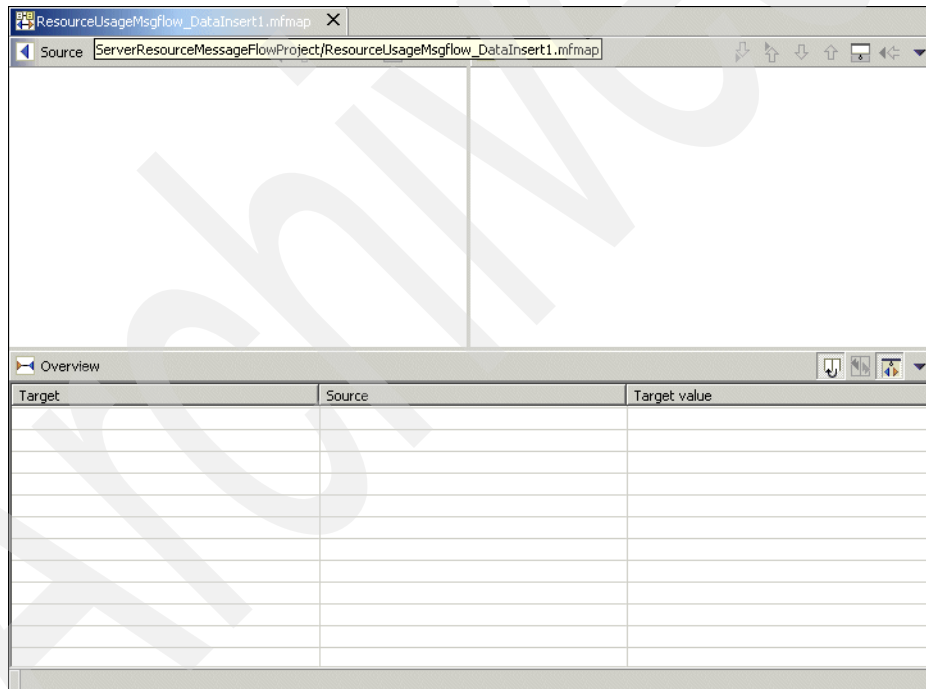


Figure 7-55 Mapping editor

- b. In the mapping editor, right-click in the top area labeled **Source** and select **Add Message Mapping Input**. In the Add Message window, select the ResourceUsage message definition and click **Finish** as illustrated in Figure 7-56 on page 262.

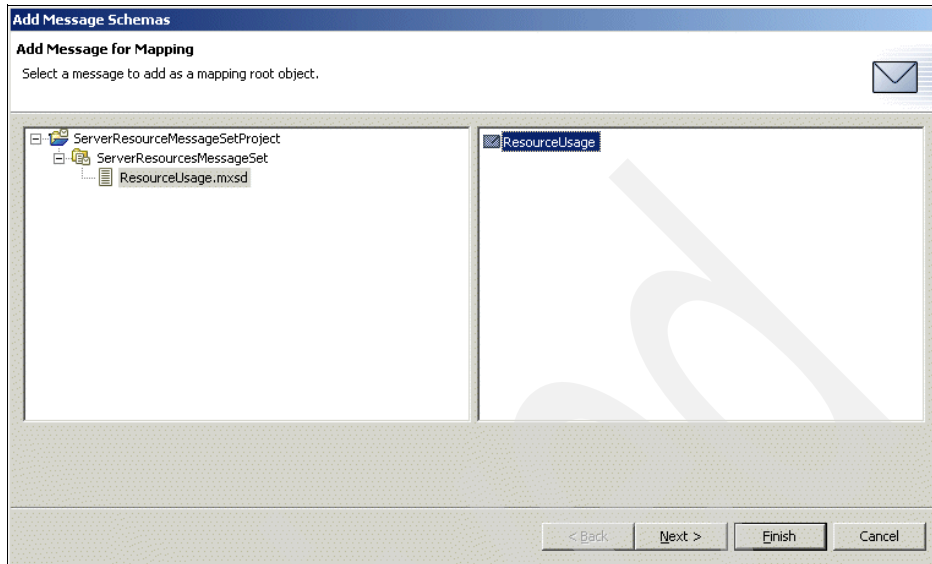


Figure 7-56 Message Mapping Input

- c. The ResourceUsage message definition will be listed in the source area of the mapping editor. Right-click now in the area labeled Target and select **Add RDB Table Mapping Output**.
- d. In the Add Database Table window, select the option to import and add table schemas from a database and click **Next** as illustrated in Figure 7-57 on page 263.

**Note:** Database connection information and table layout information can be saved and reused. If, for example, you want to create another message flow that also includes mapping with the table MEM, you can choose the option to reuse the table schema information stored in the workspace of the Message Brokers Toolkit.

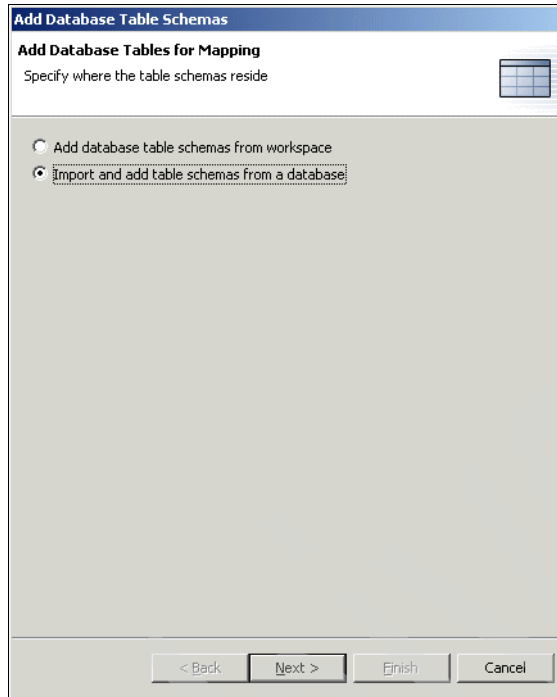


Figure 7-57 Add Database Table information for mapping

3. In the next step, provide information that allows the Message Brokers Toolkit to connect to the database and to retrieve schema information. Figure 7-58 on page 264 shows us that you need to provide:
  - Database name
  - User ID authorized to connect to the database and to retrieve information  
This is a user known to the Linux system that hosts the database.
  - The password of that user ID
  - The database type, which is DB2 V8.1 in our case
  - The JDBC driver
  - The zip file (or jar file for other database products) implementing the JDBC driver.

Click **Next** to proceed to the following step.

**Add Database Table Schemas**

**Database connection**  
Enter database name and connection information.

Database name: RESDB

Userid: db2inst1

Password: \*\*\*\*\*

Database vendor type: DB2 Universal Database V8.1

JDBC Driver: IBM DB2 APP DRIVER

Host:

(Optional) Port number:

Server name:

Database location: Browse...

JDBC driver class: COM.ibm.db2.jdbc.app.DB2Driver

Class location: C:\SQLLIB\java\db2java.zip Browse...

Connection URL: jdbc:db2:RESDB

< Back Next > Finish Cancel

Figure 7-58 Add Database Table Schemas

4. In the next window, select the database table **MEM** as illustrated in Figure 7-59 on page 265.



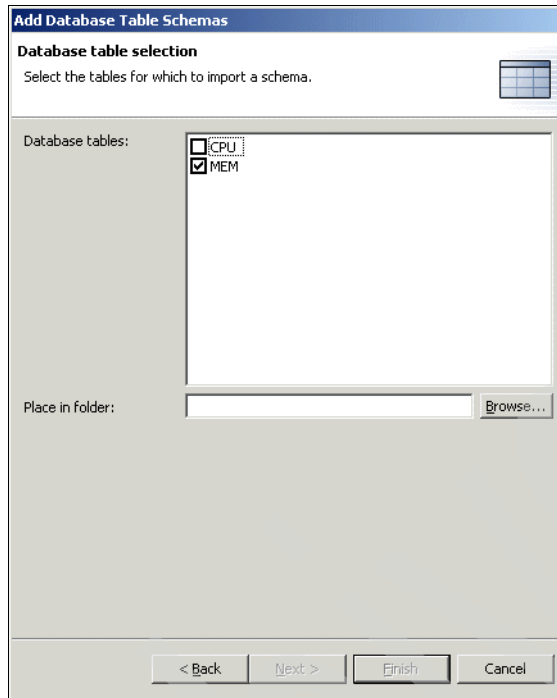
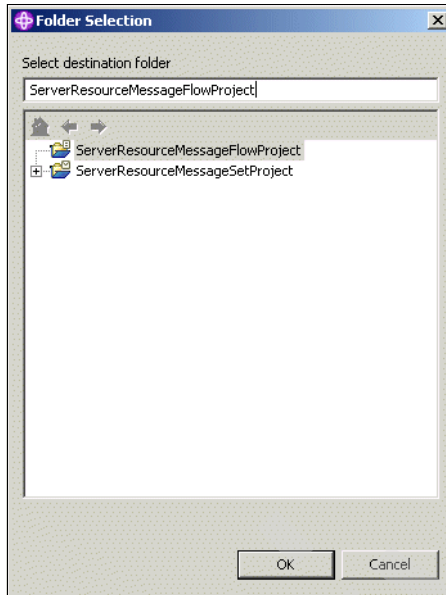


Figure 7-59 Selecting Database Table

5. The table schema information is going to be stored in a file in your workspace. Therefore, you need to indicate a location in the workspace where you want to store that information. Click **Browse** to locate a folder. See Figure 7-60 on page 266.

Select a folder name that suits and click **OK** to return to the main dialog.

**Tip:** You can create a special project fostering table schema information, or you can, for example, create a folder within the message flow project to store these database schema information files. In this example, we simply store it in the root folder of the message flow project.



*Figure 7-60 Select database location for schema information*

In the next window (Figure 7-61 on page 267), click **Next** to confirm the selection of the database table. Click **Finish** in the final step.

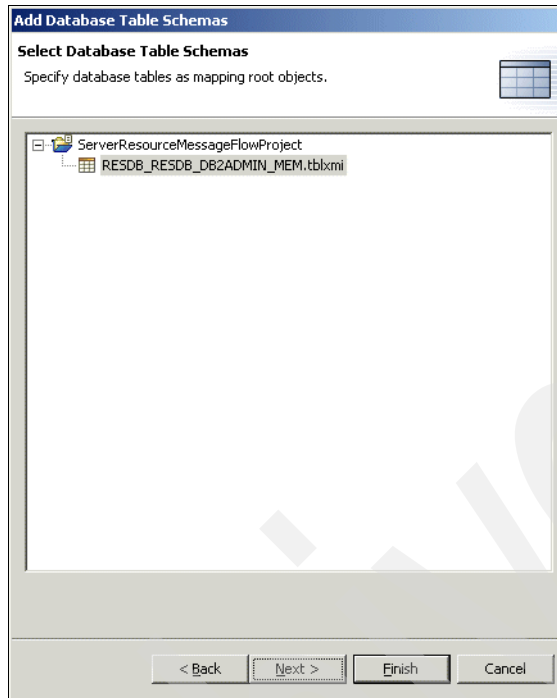


Figure 7-61 Select Database Table Schemas

6. The mapping source and target is now displayed in the mapping editor. Drag and drop each field of source to a destination. After dragging and dropping, the mapping editor should look as shown in Figure 7-62 on page 268.

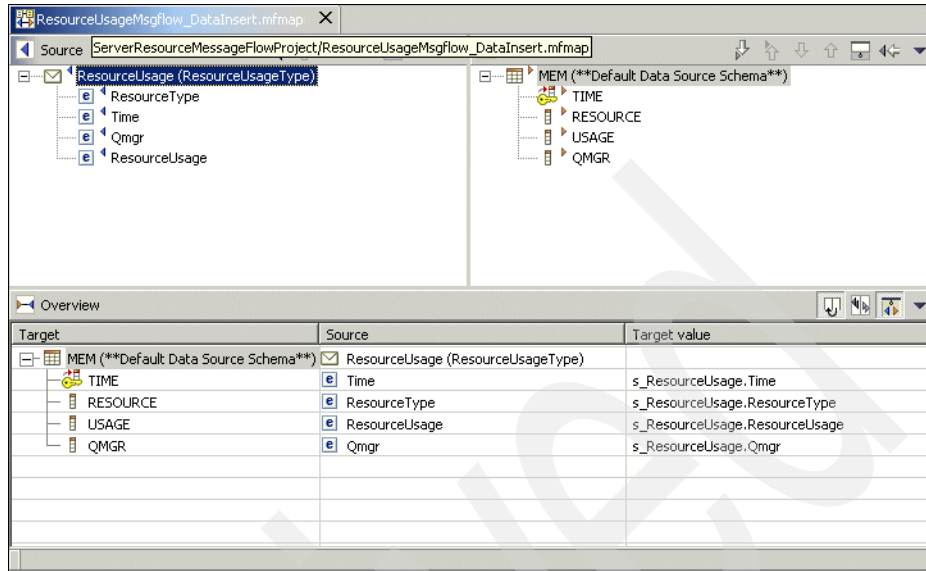


Figure 7-62 After mapping

By default, each mapping in Figure 7-62 results in a single INSERT statement for a single column. Thus, one message would result in four records. This behavior is visualized in the Outline view in Figure 7-63. This is of course not the intention and it would even result in an error, because the last three INSERTs would result in a row with a NULL value for TIME column, which is not allowed according to the table definition. See Example 7-4 on page 229.

- To combine the four INSERTs into a single INSERT for four columns, select the four mappings in the Outline view, shown in Figure 7-62. While the four mappings are selected, right-click them and select **Combine to Same Row** → **Remove selected mappings**.

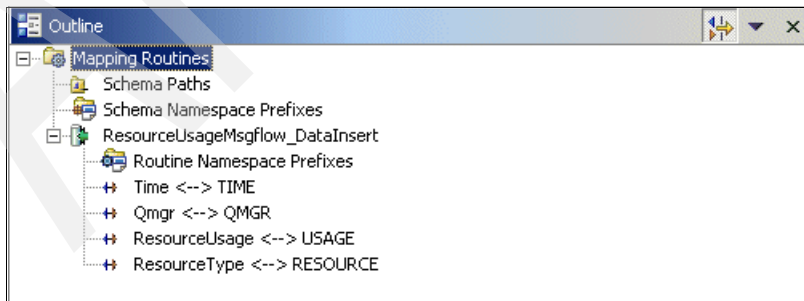


Figure 7-63 Mappings before combining

The result is shown in Figure 7-64, a single mapping that references all four message elements and also all four table columns.

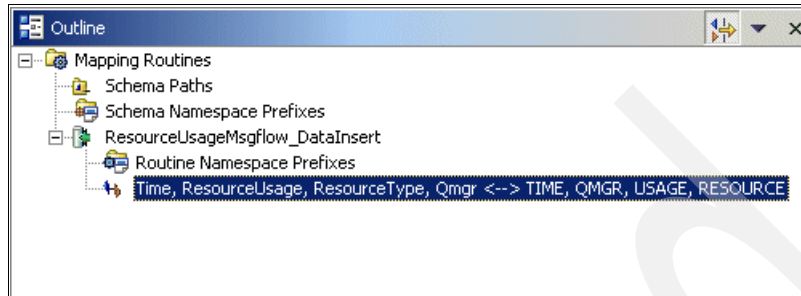


Figure 7-64 Mapping after combining

8. This completes the first DataInsert node. Repeat the same procedure to create similar mappings for the second DataInsert node. However, this time, select the table CPU.

Figure 7-65 on page 270 shows the Message Brokers Toolkit with all developed resources. The message flow editor shows the completed message flow. The Outline view in the bottom left corner shows how nodes are connected to each other. The Resource Navigator view shows the two projects. The message set project lists the single message definition that we needed for this scenario. The message flow project holds a few more resources. Within the default folder, it first lists an ESQL file, which has one child object: the ESQL module associated with the Filter node. Below that, we have the object (or better, the file) that holds the message flow itself. Further below, we have two files that hold the mapping specification for the two DataInsert nodes.

Finally, within the root folder of the message flow project, we see files associated with the database schema.

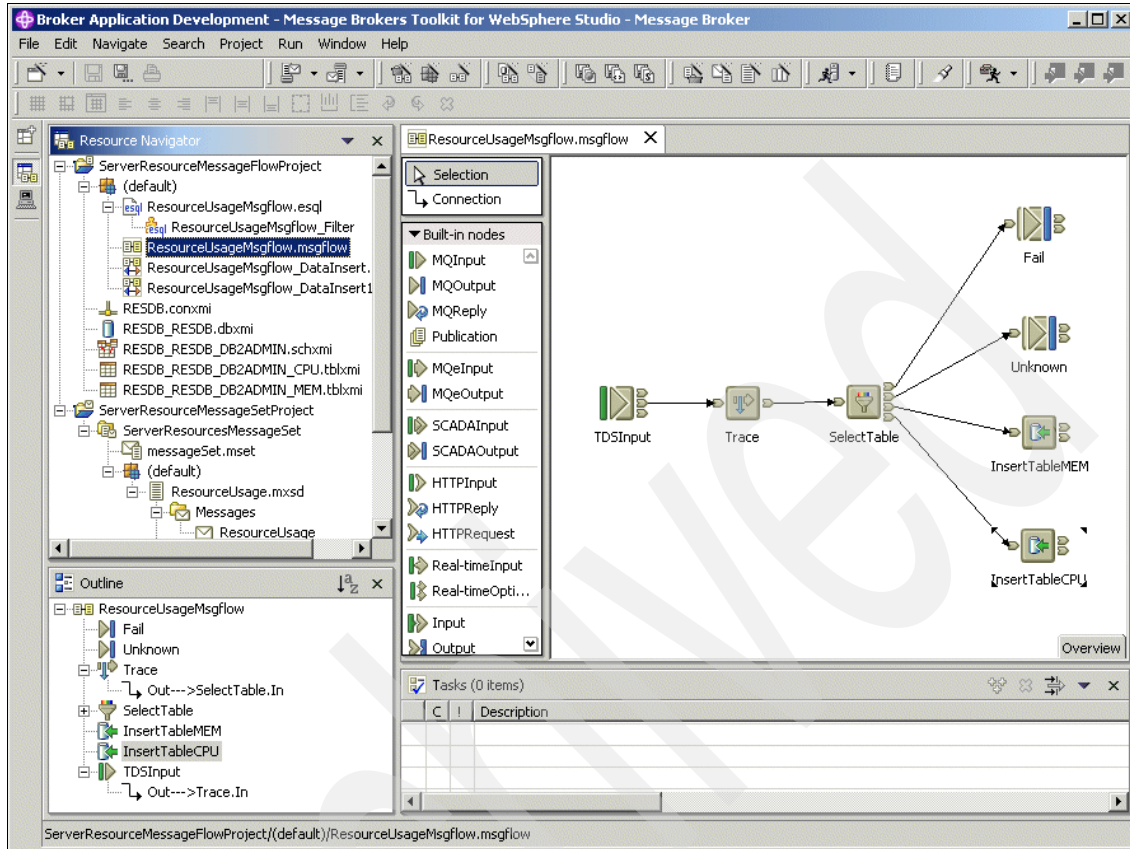


Figure 7-65 Application Development perspective showing all artefacts

## 7.8 Configuring the broker domain

The previous section discussed the development process for a message set and message flow. In this section, we return to the runtime environment on Linux and complete the configuration.

### 7.8.1 Queue manager configuration

We are planning to use two message brokers on the hosts brk1 and brk2. Both of these machines have been used before to implement MQ clusters. A broker always needs to have a queue manager and we will reuse those two queue managers qmgr and qmgr5 for the broker configuration. Review 6.6, “WebSphere

MQ cluster setup” on page 187 to understand the set-up of these two queue managers.

## 7.8.2 Creation and configuration of configuration manager

A broker domain consists of one configuration manager, a component that needs to be implemented on a Windows machine. In our lab setup we use the same Windows machine that was used to run the Message Brokers Toolkit. However, in reality, the configuration manager will likely run on a machine that is not used to develop the solution.

The communication between the Message Brokers Toolkit and the configuration manager, and between the configuration manager and the brokers is always MQ based. The configuration manager itself uses a queue manager as well. Create a new queue manager, for example `config`, on this Windows system. Configure also a listener, for example for port 3610.

1. To create a configuration manager on the Windows server, you can use the command shown in Example 7-9. This new configuration manager will use the database `WBICMDB` on the local system and the queue manager `config`. To run this service and to access the database, the configuration manager will use the user ID `db2admin`.

### *Example 7-9 Create the configuration manager*

---

```
mqsicreateconfigmgr -i db2admin -a itso4you -q config -n WBICMDB
```

---

2. To start the configuration manager, you can either use the Windows Services application or you can use the command `mqsistart configmgr`.
3. The configuration manager needs to have a working MQ communication with the queue managers `qmgr4` and `qmgr5` that are going to be associated with the message brokers. Because we have already established an MQ cluster that has `qmgr4` and `qmgr5` as its members, it becomes relatively easy to establish MQ communication between the three queue managers `config`, `qmgr4` and `qmgr5`. It is sufficient to join the queue manager `config` to that cluster. Since `qmgr5` is a full repository queue manager, we need to create a cluster sender channel pointing to this queue manager and a cluster receiver channel for queue manager `config`. These commands are shown in Example 7-10.

### *Example 7-10 Joining config queue manager to cluster*

---

```
def chl('to.config') chltype(clusrcvr) trptype(TCP) conname('win6336(3610)') cluster(ITS0)

def chl('to.qmgr5') chltype(clussdr) trptype(TCP) conname('brk2(3604)') cluster(ITS0)
```

---

### 7.8.3 Broker creation

We can now continue to create the actual message brokers on the Linux machines brk1 and brk2. Before proceeding, the assumptions are that:

- ▶ A database WBIBKDB is defined, on host was in our case.
  - ▶ A DB2 client connection is created on the hosts brk1 and brk2.
  - ▶ The database WBIBKDB is catalogued as an ODBC data source.
  - ▶ The queue managers qmgr4 and qmgr5 are defined.
1. The commands to create the brokers broker1 and broker2 are shown in Example 7-11 and Example 7-12 respectively. The command to start the brokers is shown as well. The command mqsicreatebroker has several parameters.
    - The value broker1 refers to the name of the broker. Note that a broker is named while a configuration manager is not.
    - The value db2inst1 refers to the user ID that is going to be used to run the broker. The same user ID is going to be used to access the database.
    - The value qmgr4 refers to the queue manager that the broker is going to use
    - The value WBIBKDB refers to the ODBC catalogued database

If it is required to use a different user ID to access the database, you can use the parameters `-u` and `-p` to pass a valid database user ID and password. In our set-up the user `db2inst1` is defined on the systems `brk1`, `brk2` and was and has always the same password. This results in a simplified command to create the broker.

*Example 7-11* Creating broker1 on brk1 server

---

```
mqsicreatebroker broker1 -i db2inst1 -a itso4you -q qmgr4 -n WBIBKDB  
mqsistart broker1
```

---

*Example 7-12* Creating broker2 on brk2 server

---

```
mqsicreatebroker broker2 -i db2inst1 -a itso4you -q qmgr5 -n WBIBKDB  
mqsistart broker2
```

---

### 7.8.4 Connecting brokers to the configuration manager

At this stage we have two brokers, an instance of the Message Brokers Toolkit, and a configuration manager. All of them have been created, started, and used so far without any knowledge of each other. In this section, we are first going to connect the Message Brokers Toolkit to the configuration manager, then use this link to pass information about the two brokers to the configuration manger.



1. In the Message Brokers Toolkit, switch first to the Broker Administration perspective. As you can see in Figure 7-66, we have again a Resource Navigator. However, fewer resources are now shown here. As a broker administrator, you do not need to know details about a message flow, such as the fact that it has an ESQL module or a DataInsert node.

Below the Resource Navigator view, we have the Domains view. A domain is a collection of one or more brokers and exactly one configuration manager. As a Message Brokers Toolkit user, you can have references to and work with many broker domains. For example, a development domain and a production domain, all managed from within a single Eclipse workspace.

2. To create a domain, right-click in the **Domains** view and select **New** → **Domain**.

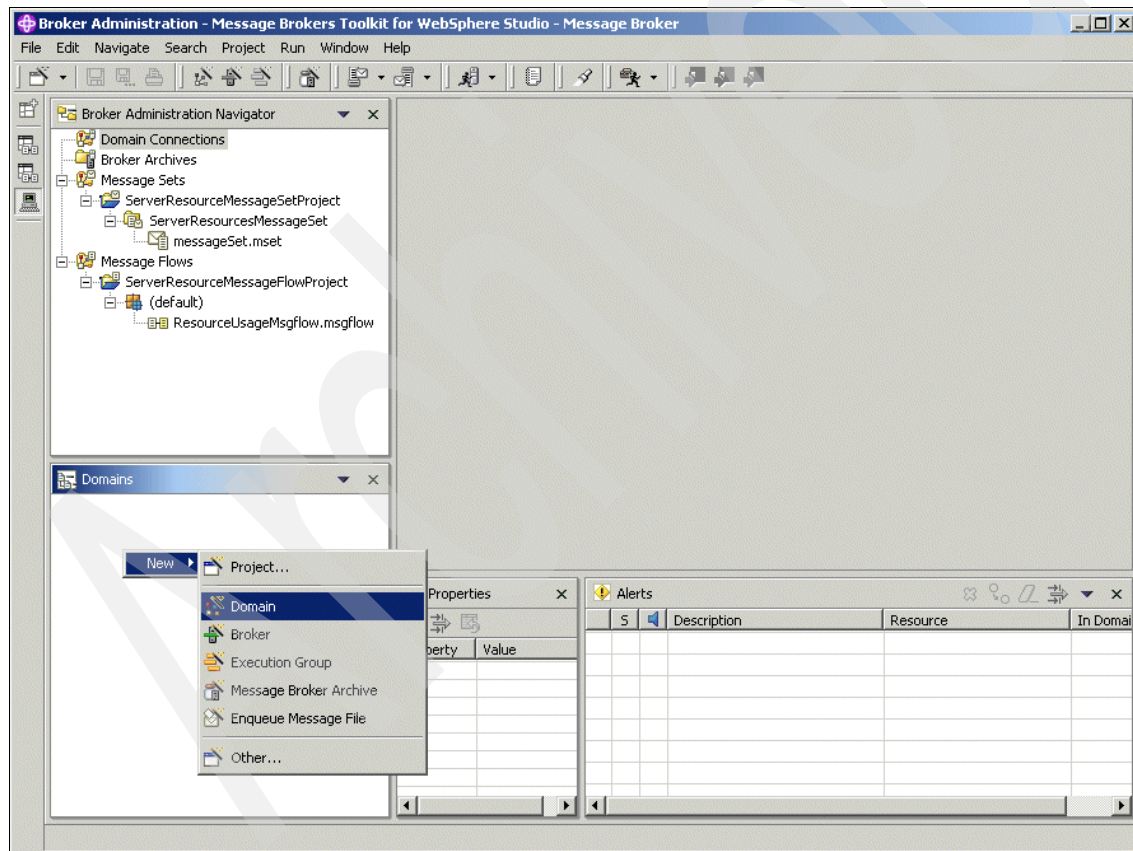
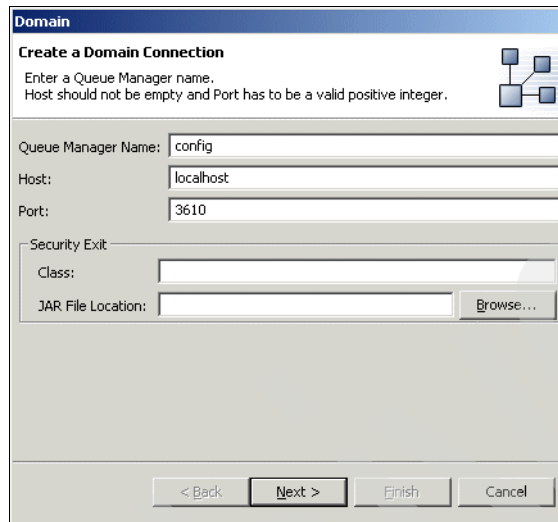


Figure 7-66 Broker Administration perspective

3. In the next step, shown in Figure 7-67, provide the name of the queue manager associated with the configuration manager, which is `config` in our case. You also need to provide the host name and port that is used by the MQ listener component. Click **Next** to proceed.



The screenshot shows a dialog box titled "Domain" with the subtitle "Create a Domain Connection". Below the subtitle, there is a small diagram of a network topology and a text instruction: "Enter a Queue Manager name. Host should not be empty and Port has to be a valid positive integer." The form contains three input fields: "Queue Manager Name:" with the value "config", "Host:" with the value "localhost", and "Port:" with the value "3610". Below these fields is a "Security Exit" section with a "Class:" field and a "JAR File Location:" field with a "Browse..." button. At the bottom of the dialog are four buttons: "< Back", "Next >", "Finish", and "Cancel".

Figure 7-67 Create a domain connection - Step 1

4. In the last step, you can provide a name to the connection information, for example `Local Connection`. Also, this connection information is stored in the workspace in a server project. By default, a new project `Servers` will be created by this wizard. However, you could have defined a server project first and then select it here in Figure 7-68 on page 275. Click **Finish**.

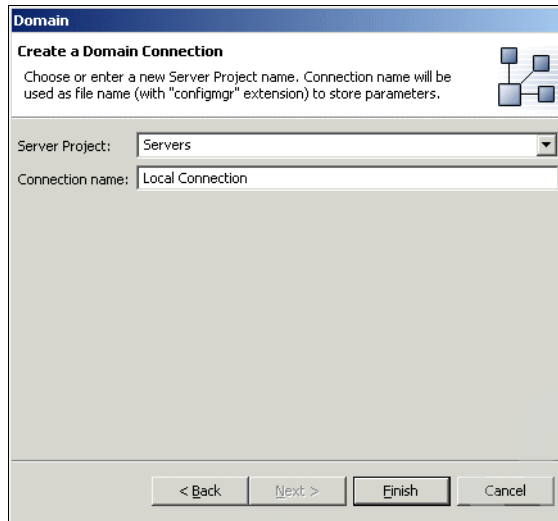


Figure 7-68 Create a domain connection - Step 2

The Message Brokers Toolkit uses this information to connect to the queue manager config and to write a request message on a system queue used by the configuration manager. As a result, this request message can only be processed if both the queue manager and the configuration manager are running. Upon successful completion of that request, your Message Brokers Toolkit should show an empty domain in the Domains view (see Figure 7-69 on page 276). Note also the addition of the Servers project and the connection file within that project. Contrast the Resource Navigator views in Figure 7-66 on page 273 and Figure 7-69 on page 276.

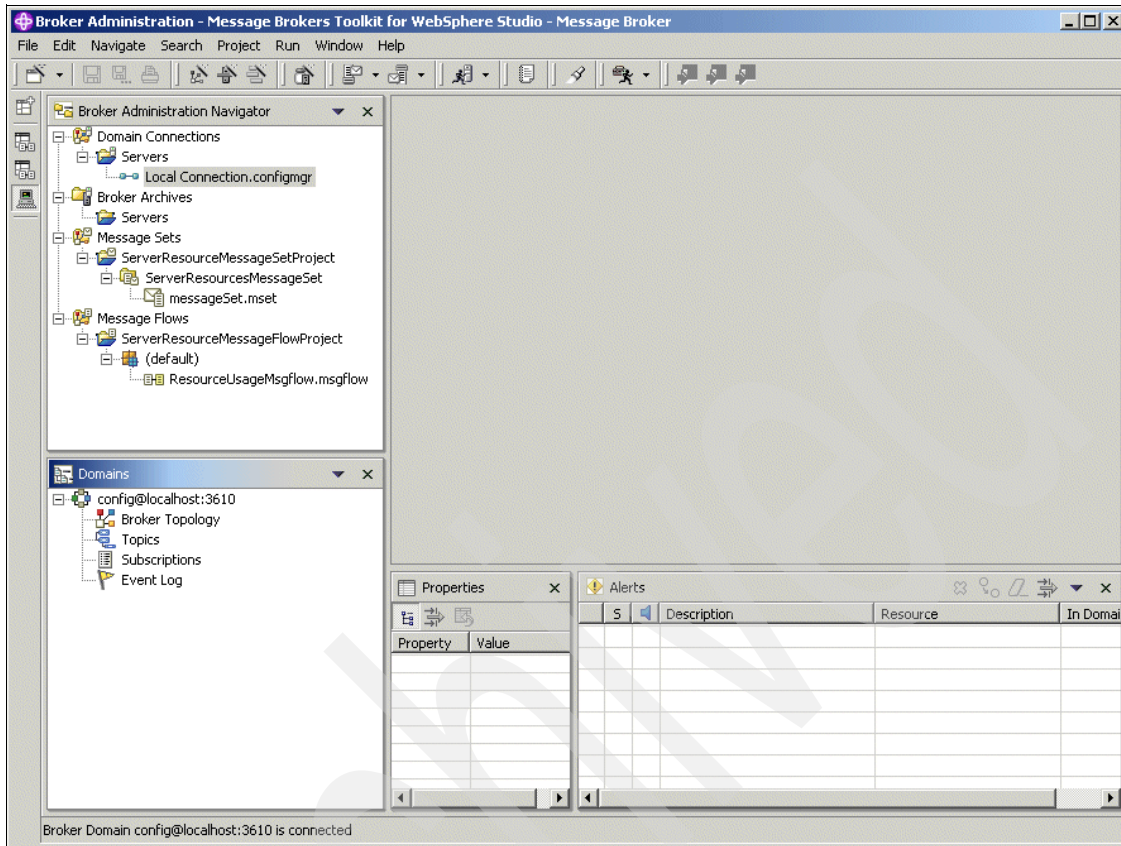


Figure 7-69 Message Brokers Toolkit with a configuration manager

Creating a domain in the Message Brokers Toolkit is nothing more than creating a reference to an existing configuration manager using the queue manager associated with that configuration manager.

5. We can now proceed by adding references to the two brokers. In the Domains view of the Broker Administration perspective, right-click and select **New** → **Broker**. Enter the name of the broker, broker1, and add the name of the queue manager associated with that broker, qmgr4 in our case. See Figure 7-70 on page 277. Click **Finish** to complete.

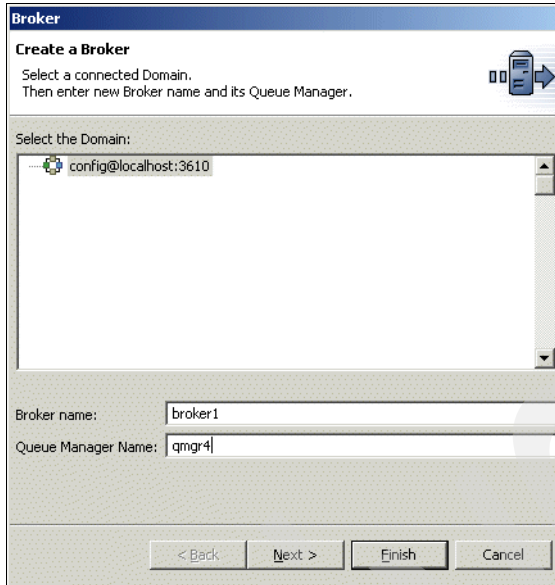


Figure 7-70 Create reference to broker1

6. Repeat this process for the second broker, broker2, with queue manager qmgr5. See Figure 7-71.

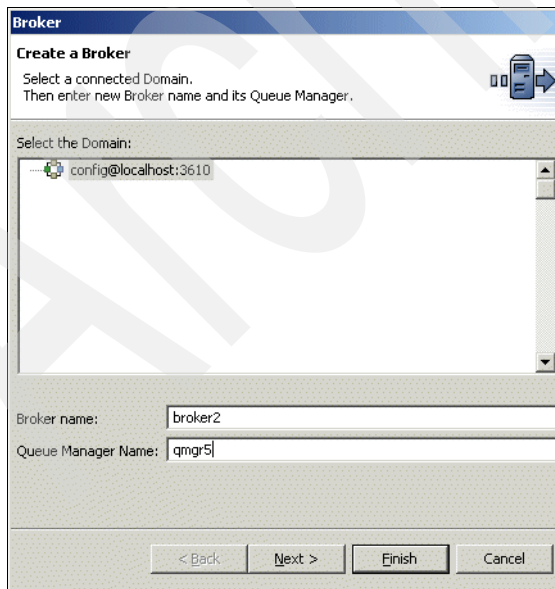


Figure 7-71 Create reference to broker2

Note that there is no need to provide host name or port information in Figure 7-70 on page 277 and Figure 7-71 on page 277. The assumption is that it is already possible to establish MQ communication between the queue managers. We achieved this earlier by adding the queue manager config to the existing cluster that consisted of qmgr4 and qmgr5.

Figure 7-72 on page 279 shows us the Message Brokers Toolkit with a single domain in the Domains view. That domain consists of two brokers. Each broker has a single execution group, called default. However, these execution groups are currently not running, because no solutions are deployed to them. This will be the task of the next section.

Figure 7-72 on page 279 also shows us the Event Log, opened by double-clicking the element Event Log in the Domains view. The Event Log provides us with information about the deployment process. It should not be confused with the Windows Event Viewer, which is used to report runtime messages by the configuration manager and any brokers defined on Windows. Our brokers run on Linux and use the syslog component to report runtime warnings and errors. The Event Log will never show that kind of messages.

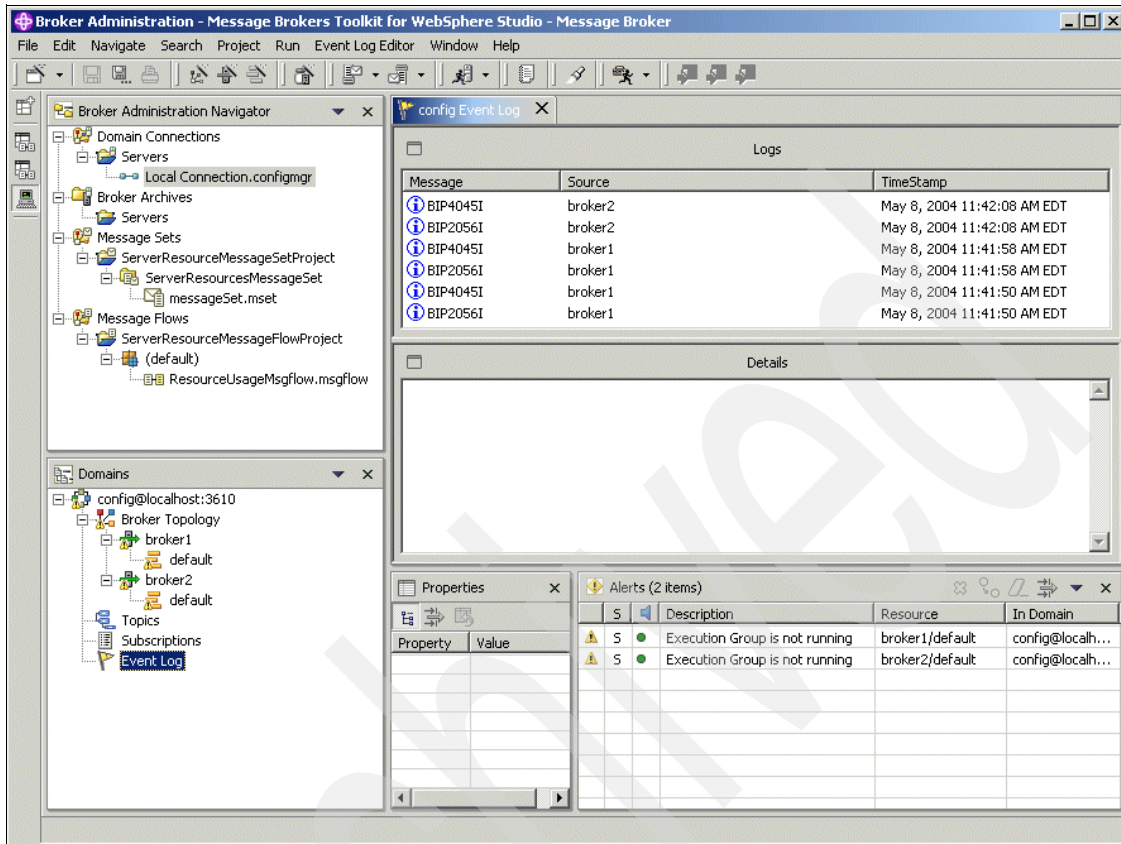


Figure 7-72 Message Brokers Toolkit showing broker domain

## 7.9 Deploying the message flow and message set

We now have a fully configured runtime environment consisting of a configuration manager and two brokers. We also have a completed broker solution consisting of a message set and a message flow. In this section, we describe the tasks required to package our solution and then deploy it to a broker. This is done by creating a *broker archive* (bar) file and deploying it to an execution group within the Linux-based brokers.

**Note:** The *broker archive* file is the unit of deployment to a broker and contains compiled message flows (cmf) and optional message sets (dictionary) resources. It also contains a deployment descriptor that contains information specific to this deployment.

A broker archive can be considered the equivalent of an enterprise archive file (EAR) in the world of J2EE applications and application servers.

## 7.9.1 Creating a message broker archive

Building a broker archive is nothing more than collecting and compiling solution resources. This file also has a configurable part for references to external resources such as queues, tables, and so on.

1. Switch to the Broker Administration perspective of the Message Brokers Toolkit.
2. Select **File** → **New** → **Message Broker Archive**.
3. A wizard starts and displays a dialog box requiring details about the broker archive file name and the associated server project. You can use the default project Servers, which will be created for you. Provide also a name for the broker archive file, for example `ResourceUsageArchive`. See Figure 7-73 on page 281.



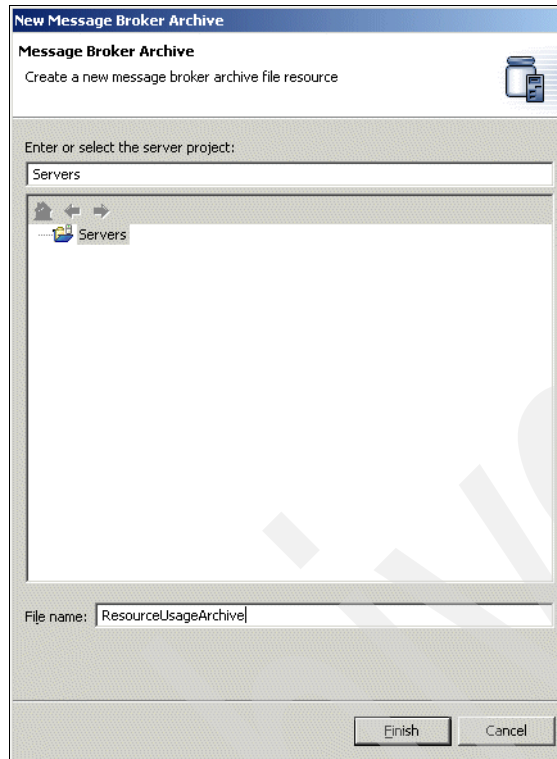


Figure 7-73 Broker Archive file

4. A new editor opens. This editor consists of two tabs, Content and Configure. The Content tab is used to add or remove solution components. The Configure tab is used to configure any external references. Click the **green icon** button, marked in Figure 7-74 on page 282, to add solution components to this archive.

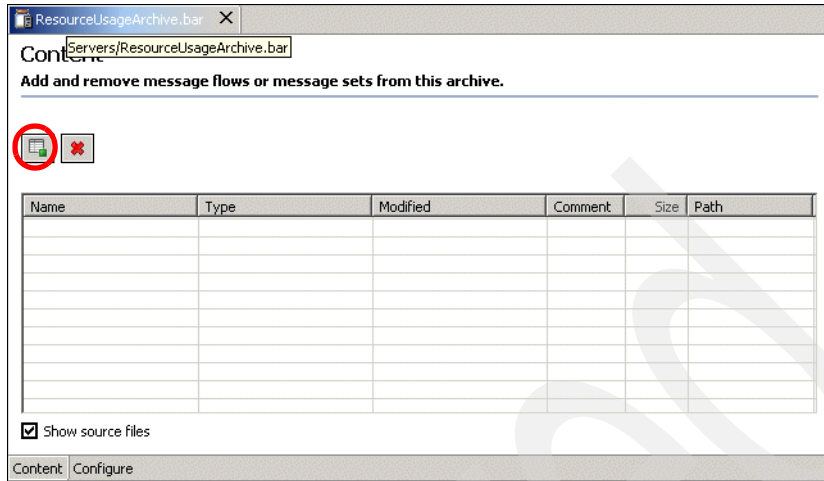


Figure 7-74 The broker archive editor

5. A new window opens, listing the message flow and message set projects in your workspace. Select the message flow and message set project that was developed for this scenario and click **OK**. See Figure 7-75.

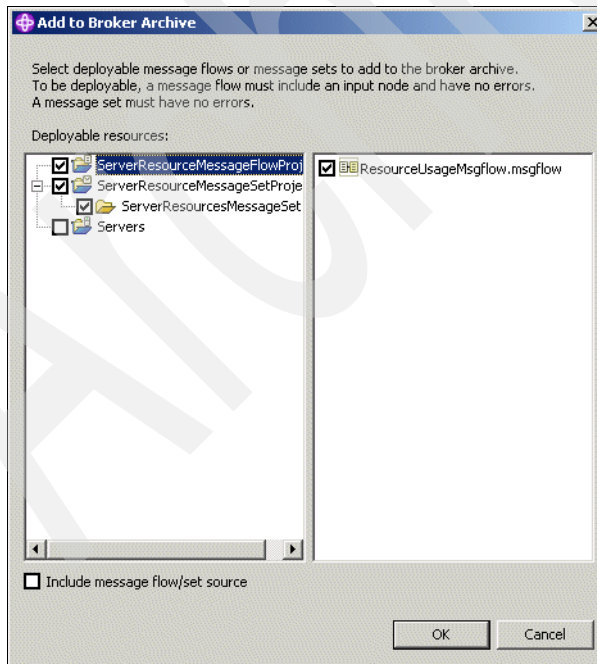


Figure 7-75 Adding message flow and message set to BAR file

- Figure 7-76 shows the archive editor. The archive now contains a compiled message flow and a dictionary, the runtime version of a message set. Select the **Configure** tab to review references in this archive to external resources.

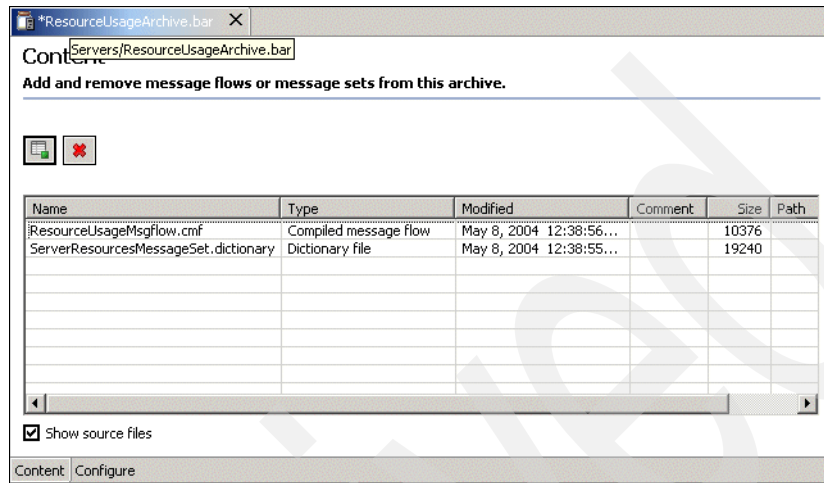


Figure 7-76 Broker archive with message flow and message set

- Figure 7-77 on page 284 shows us the references to external resources in this broker archive. The references are sorted by node. Select for example the **Trace** node. You can change the location and name of the trace file. Select for example the node **Unknown** and you can change the name of the queue to be used for messages that do not have mem or cpu as a resource type. Refer to the Filter node discussion in “Properties of the Filter node” on page 257.

Note that changing any of these references does not change the actual message flow in the workspace. The idea is that you could have a broker archive for a test environment and a broker archive for a production environment. Names of queues, queue managers, directories and databases could be different for both environments. Instead of changing the message flow before production deployment, you can simply create a different broker archive and configure the references to external resources in a different way.

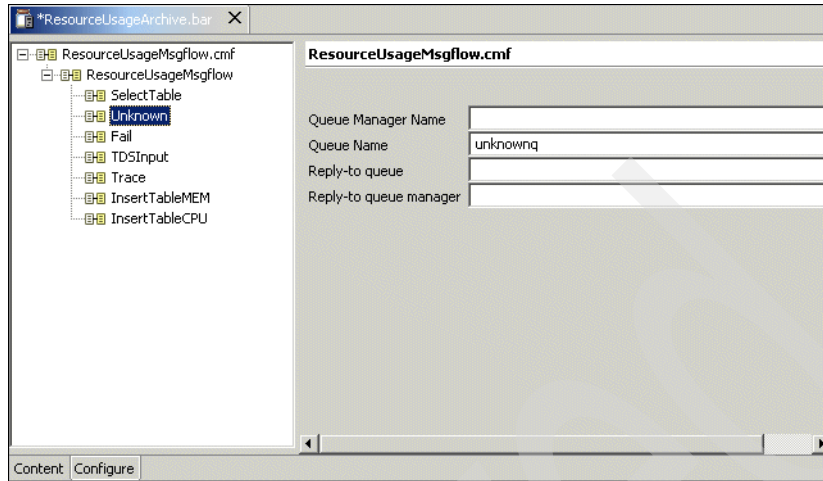


Figure 7-77 Configure the references to external resources in the broker archive

**Important:** It is particularly important to make sure that the message flows or message sets that you add to a bar file have been saved prior to being added to the bar file. Changing a message flow source after it has been compiled in a bar file requires repeating the same steps described here to add the latest version to the bar file.

A compiled message flow source is a static entity and does not automatically change when changes are made to the message flow source. While it can seem redundant to create iterations of a bar file, bear in mind that each could account for a different version of the associated message flow.

It is important to save a bar file prior to deploying. However, if you have not saved it, you will be prompted to do so prior to deployment.

The process of adding a resource to a broker archive can be compared with adding a Java class to a JAR file. If the class changes, the JAR file needs to be updated as well.

## 7.9.2 Deploying a bar file

The final step is to deploy this broker archive to the two brokers in our domain.

1. While in the Broker Administration perspective, locate the broker archive file in the navigator view. See Figure 7-78 on page 285. Right-click the broker archive file **ResourceUsageArchive** and select **Deploy File**.

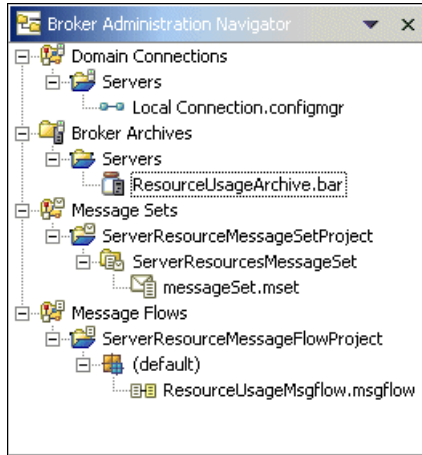


Figure 7-78 Navigator view listing the broker archive and the domain connection

2. A window (Figure 7-79) opens so that you can select an execution group belonging a broker. Select the execution group **default** belonging to broker broker1 and click **OK**.

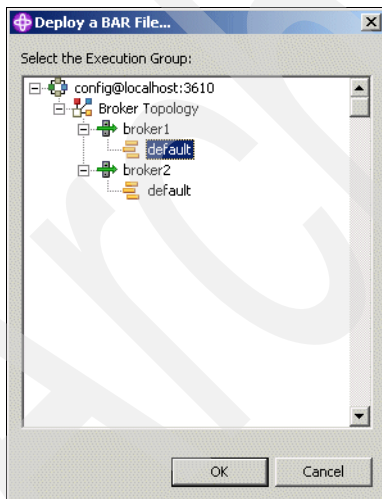


Figure 7-79 Deploy broker archive to broker1

3. Because we want to have the solution running on both brokers, right-click the archive again and select once more **Deploy File**. This time, select execution group **default** of broker broker2. See Figure 7-80 on page 286.

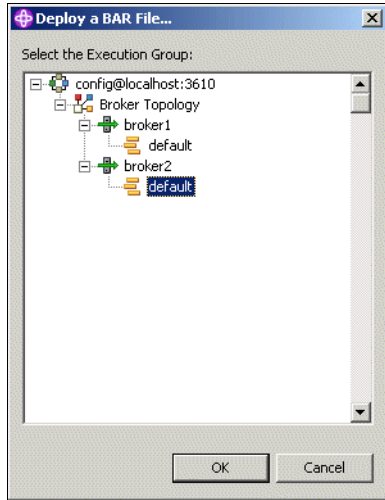


Figure 7-80 Deploy broker archive to broker1

After each deployment, you will receive a pop-up message box that says that the deployment has been initiated successfully. This means nothing more than that the configuration manager has accepted the broker archive for deployment to the selected broker. The result of the deployment, that is the answer of the broker, is not known yet at this time.

The deploy process is now initiated and messages are sent between the configuration manager and the brokers. The Alerts and Domains views will dynamically refresh as the deployment progresses. Also, the Event Log will contain information about the deployment process and about the responses from the brokers.

Figure 7-81 on page 287 shows the end result of both deployments. In the Domains view, you can now see that the message flow and the message set are assigned to the executions groups of both brokers. The messages in the Event Log (top-right) show the positive answers that returned from the broker. The Alerts (bottom-right) view no longer says that the execution groups default are not running. Compare the view Alerts in Figure 7-72 on page 279 with the view Alerts in Figure 7-81 on page 287.

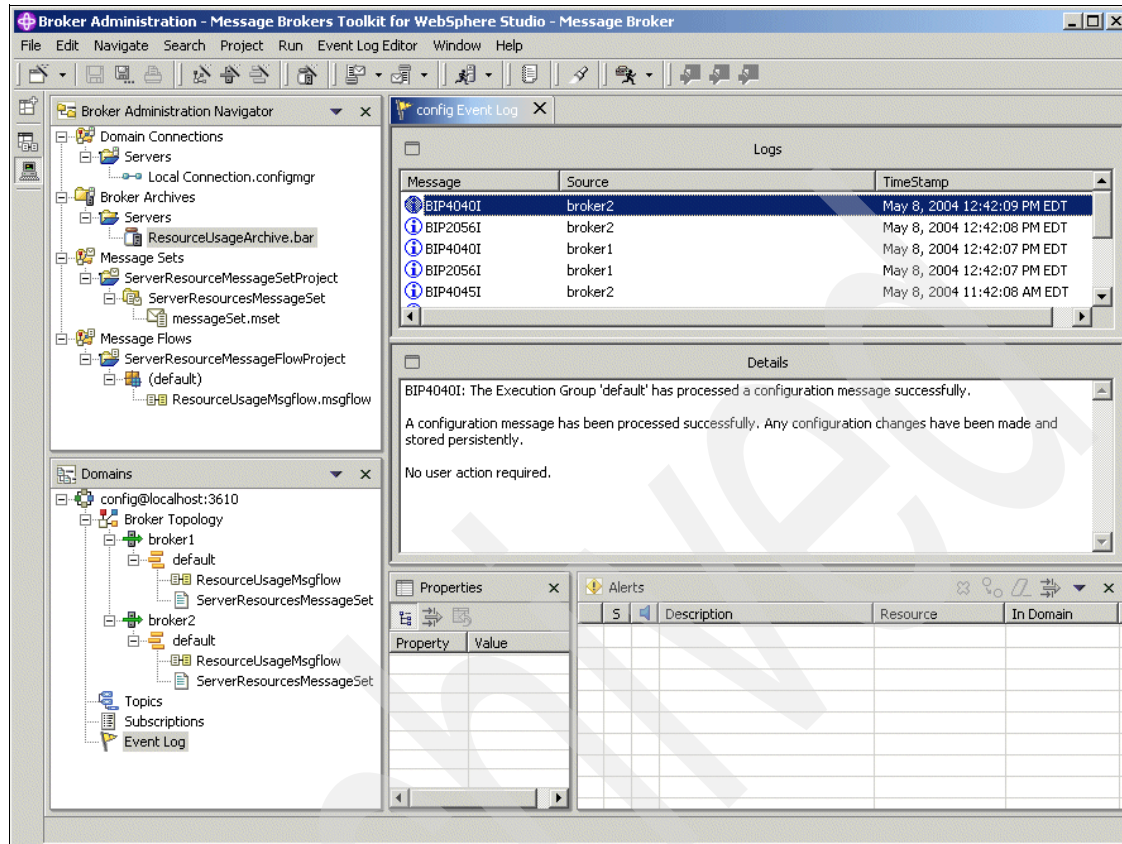


Figure 7-81 Broker Administration with deployed solutions to the brokers

## 7.10 WebSphere Application Server configuration

In this section we review the scenario of this chapter. On hosts qmgr1 and qmgr2, we have two instances of the message generation application that send messages to the brokers. On hosts brk1 and brk2, we have two brokers that are receiving and processing these messages. The processing consists of storing the message information in two tables, based on the resource type. The client side of this scenario is the same Web application that we have used in earlier chapters of this redbook.

The client side of this scenario requires only one change. We need to define and configure the JDBC resources in WebSphere Application Server so that the database RESDB can be found and accessed by the Web application.

A data source is used by the application server to access the data from a database. A data source is created for use by a JDBC provider. The provider details the specific JDBC driver implementation class that will be used by the data source. When defining a data source, we also need to provide a default user ID to access that database. Such a user ID is stored in J2C Authentication Data. Finally, the configuration of a data source typically refers to a WebSphere environment variable to locate the JDBC driver implementation class. Thus, in summary, we need to define and/or update the following:

- ▶ WebSphere environment variable
  - ▶ J2C Authentication Data
  - ▶ JDBC provider
  - ▶ Data source
1. Start the Administrative Console and select in the left pane **Security** → **JAAS Configuration** → **J2C Configuration Data**.
  2. The right pane of the Administrative Console might now show an empty list or a list of existing entries. Click **New** to add a new user entry. See Figure 7-82.
  3. Provide an alias for this user ID, for example dbUser. Provide also a valid user ID, for example db2inst1, and its password.
  4. Click **OK**.

**J2C Authentication Data Entries >**  
**New**  
 Specifies a list of userid and password for use by Java 2 Connector security. ⓘ

**Configuration**

General Properties	
Alias	* dbUser ⓘ Specifies the name of the authentication data entry.
User ID	* db2inst1 ⓘ Specifies the J2C authentication data user ID.
Password	* [masked] ⓘ Specifies the password to use for the target Enterprise Information System.
Description	ⓘ Specifies an optional description of the authentication data entry. For example, this authentication data entry is used to connect to DB2.

Apply OK Reset Cancel

Figure 7-82 Create a J2C Authentication Data entry

The next step is to review and update the environment variables in WebSphere Application Server.

1. In the left pane of the Administrative Console, select **Environment** → **Manage WebSphere Variables**.



2. A list of existing variables should open. Usually, the variable DB2UNIVERSAL\_JDBC\_DRIVER\_PATH is already defined but does not have the correct directory. Click this variable and update the directory to point to the directory that contains the Java jar and zip files for DB2 support for JDBC, for example, /home/db2inst1/sqllib/java/.

We can now create the actual JDBC Provider.

1. In the Administrative Console, select in the left pane **Resources** → **JDBC Providers**.
2. Click **New** to create a new provider. See Figure 7-83.
3. Select **DB2 Universal JDBC Driver Provider** for the drop-down list and click **Apply**. This will enable additional properties.

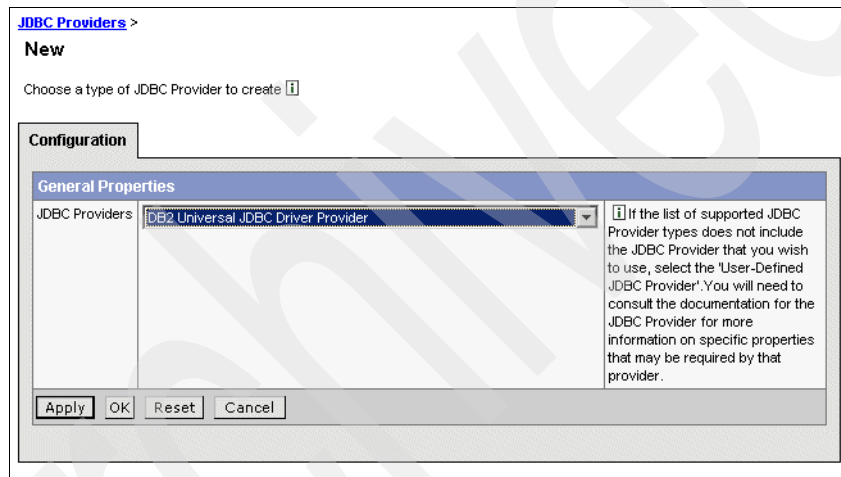



Figure 7-83 Create a new JDBC driver provider

4. Review the additional properties. You might have to review the classpath to use the correct environment variables. See Figure 7-84 on page 290.

[JDBC Providers >](#)

**New**

JDBC providers are used by the installed applications to access data from databases. 

**Configuration**





General Properties		
Scope	* cells:student:nodes:student	 The scope of the configured resource. This value indicates the configuration location for the configuration file.
Name	* DB2 Universal JDBC Driver Provider	 The name of the resource provider.
Description	Non-XA DB2 Universal JDBC Driver-compliant Provider. Datasources created under this provider support only 1-phase commit processing except in the case where driver type 2 is used under WAS z/OS. On WAS z/OS, driver type 2 uses RRS and supports 2-phase commit processing	 A text description for the resource provider.
Classpath	\$(DB2UNIVERSAL_JDBC_DRIVER_PATH)\db2jcc.jar \$(DB2UNIVERSAL_JDBC_DRIVER_PATH)\db2jcc_license_cu.jar \$(DB2UNIVERSAL_JDBC_DRIVER_PATH)\db2jcc_license_cisu.jar	 A list of paths or JAR file names which together form the location for the resource provider classes. Classpath entries are separated by using the ENTER key and must not contain path separator characters (such as '/' or '\'). Classpaths may

Figure 7-84 Additional properties of the JDBC driver provider

5. Click **Apply** once more to enable additional properties. Scroll down and click **Data Sources**.
6. An empty list of data sources is shown. Click **New**.
7. Specify the name and JNDI Name as shown in Figure 7-85 on page 291. The JNDI name has to match that specified in the application.

[JDBC Providers](#) > [DB2 Universal JDBC Driver Provider](#) > [Data Sources](#) >

**New**

Data Source is used by the application to access the data from the database. A data source is created under a JDBC provider which provides the specific JDBC driver implementation class. [i]

**Configuration**

General Properties		
Scope	* cells:student:nodes:student	[i] The scope of the configured resource. This value indicates the configuration location for the configuration file.
Name	* ResourceUsageDS	[i] The required display name for the resource.
JNDI Name	jdbc/MessagesDS	[i] The JNDI name for the resource.
Container managed persistence	<input type="checkbox"/> Use this Data Source in container managed persistence (CMP)	[i] Enable if this data source will be

Figure 7-85 Data source configuration

8. Scroll down in this page to locate the properties Component-managed Authentication Alias and Contained-managed Authentication Alias. Select for both properties the J2C Authentication Data entry **dbUser** that we created earlier.
9. Click **Apply** once more to enable custom properties.
10. Scroll to the bottom of the page, and click on **Custom Properties**. See Figure 7-86 on page 292.
11. On the next page, click the **databaseName** property.

[JDBC Providers](#) > [DB2 Universal JDBC Driver Provider](#) > [Data Sources](#) > [ResourceUsageDS](#) >

### Custom Properties

Custom properties that may be required for Resource Providers and Resource Factories. For example, most database vendors require additional custom properties for data sources that will access the database. ⓘ

Total: 17

Filter

Preferences

<input type="checkbox"/>	Name ▾	Value ▾	Description ▾	Required
<input type="checkbox"/>	<a href="#">databaseName</a>	<a href="#">RESDB</a>	This is a required property. This is an actual database name, and its not the locally catalogued database name. The Universal JDBC Driver does not rely on information catalogued in the DB2 database directory.	<a href="#">true</a>
<input type="checkbox"/>	<a href="#">driverType</a>	<a href="#">4</a>	The JDBC connectivity-type of a data source. If you want to use type 4 driver, set the value to 4. If you want to use type 2 driver, set the value to 2. On WAS 7.0.0, driverType 2 uses RRS and	<a href="#">true</a>

Figure 7-86 Custom properties of this data source

12. Set the name of the database to RESDB (Figure 7-87) and click **OK** to return to the list of properties.

[JDBC Providers](#) > [DB2 Universal JDBC Driver Provider](#) > [Data Sources](#) > [ResourceUsageDS](#) > [Custom Properties](#) >

### databaseName

Custom properties that may be required for Resource Providers and Resource Factories. For example, most database vendors require additional custom properties for data sources that will access the database. ⓘ

**Configuration**

General Properties		
Scope	* cells:student:nodes:student	ⓘ The scope of the configured resource. This value indicates the configuration location for the configuration file.
Required	true	ⓘ
Name	databaseName	ⓘ Name associated with this property (for example, PortNumber and ConnectionURL).
Value	<input type="text" value="RESDB"/>	ⓘ Value associated with this property in this property set.
Description	This is a required property. This is an actual database name, and its not the locally catalogued database name. The Universal JDBC Driver does not rely on information catalogued in the DB2 database directory.	ⓘ Text to describe any bounds or well-defined values for this property.
Type	java.lang.String	ⓘ Fully qualified Java type of this property (java.lang.Integer, java.lang.Byte).

Figure 7-87 Set the name of the database for this data source

13. Select the property **serverName** and set it to the host name of the database server, which is was in our setup. See Figure 7-88.
14. Click **OK** to return to the list of properties.

[JDBC Providers](#) > [DB2 Universal JDBC Driver Provider](#) > [Data Sources](#) > [ResourceUsageDS](#) > [Custom Properties](#) > **serverName**

Custom properties that may be required for Resource Providers and Resource Factories. For example, most database vendors require additional custom properties for data sources that will access the database. ⓘ

**Configuration**

General Properties		
Scope	* cells:student:nodes:student	ⓘ The scope of the configured resource. This value indicates the configuration location for the configuration file.
Required	false	ⓘ
Name	serverName	ⓘ Name associated with this property (for example, PortNumber and ConnectionURL).
Value	<input type="text" value="was"/>	ⓘ Value associated with this property in this property set.
Description	The TCP/IP address or host name for the DRDA server. If custom property driverType is set to 4, this property is required.	ⓘ Text to describe any bounds or well-defined values for this property.
Type	java.lang.String	ⓘ Fully qualified Java type of this property (java.lang.Integer, java.lang.Byte).

Figure 7-88 Set the name of the server for this data source

15. At the top of the page that lists the custom properties, select the link to save the changes to the master configuration. Click **Save**.
16. Before we restart the WebSphere Application Server, we can perform a test of the connection to the database. Select in the left pane **Resources** → **JDBC Provider**.
17. The right pane shows the DB2 Universal JDBC Driver Provider that we just created. Scroll down to locate the link Data Sources and select this link. You should now see a list that includes the data source ResourceUsageDS. Select the data source and click **Test Connection**.
18. The result of the test will be shown at the top of the current page. See Figure 7-89 on page 294.

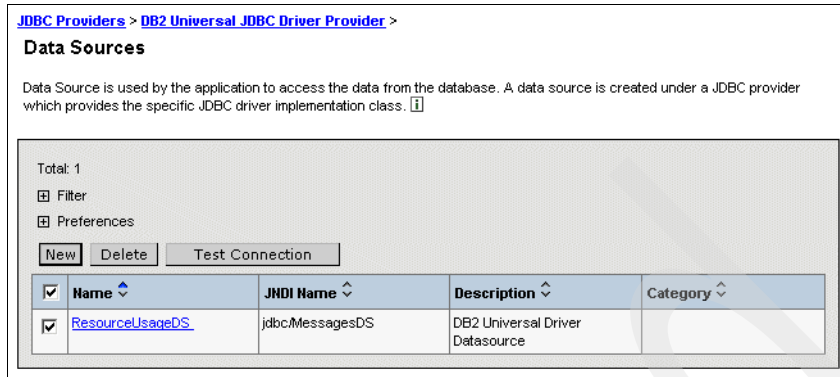


Figure 7-89 Test the connection to this data source

## 7.11 Message generation application

The message generating program used in this scenario is the same application that was used in 3.4, “Message generation application” on page 71. No changes are required to the application.

## 7.12 Message retrieval application

The message retrieval application was discussed earlier. At that time, we only discussed the retrieval of messages from a queue. This time, the Web application will retrieve messages from a database. This section discusses the additional logic needed to achieve this. We have expanded the data-retrieving, controller, and presentation model that was introduced in 3.5, “Message retrieval application” on page 73.

In Figure 7-90 on page 295, an additional JSP (selector.jsp) is introduced that collects parameters from the user. This JSP drives the same MonitorStatusServlet that got extended to support this new logic.

The class SelectMessagesBean contains all the JDBC specific code. The class MessageBean is the same as before.

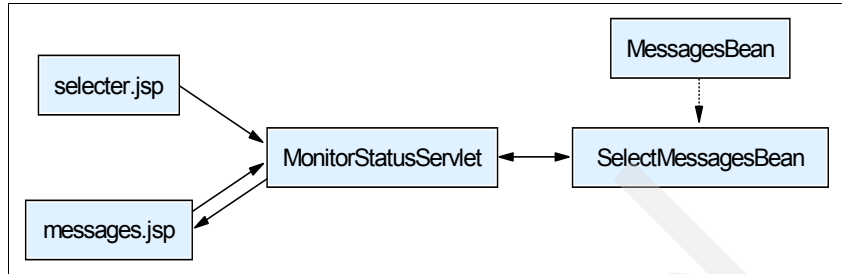


Figure 7-90 Database Web application topology

### 7.12.1 The servlet `MonitorStatusServlet.java`

The class `MonitorStatusServlet.java` is responsible for accepting requests and calling a service to fill the request. For this scenario we call the class `SelectMessagesBean.java` to retrieve messages from the database. The servlet then forwards the results to the receiver.jsp for display to the user. The same servlet is used for retrieving messages from all messaging tests because it contains no JDBC or connection management code.

The details of writing servlets are beyond the scope of this book. Our servlet is a standard servlet with only a few special methods. The method `doSelectDb()` has been added to call the `SelectMessagesBean`. The relevant portion of code is shown in Example 7-13.

Example 7-13 `doSelectDb()` for calling `SelectMessagesBean`

---

```

private ArrayList doSelectDb() {
    Utils.log(LOCATION, "Retrieving messages from select bean...");
    if (messagesBean == null) {
        messagesBean = new SelectMessagesBean();
    }
    return (ArrayList) messagesBean.getMsgs();
};
  
```

---

Notice that the method only returns an `ArrayList` that will contain a list of `String` objects and knows nothing about JDBC.

### 7.12.2 The class

The class `SelectMessagesBean.java` is responsible for creating two main objects: a data source to manage connection pooling and a connection to the database. `SelectMessagesBean.java` executes a query to retrieve the messages from the table and returns the results as a `List` of `String` objects.

## Initializing JDBC resources

The `init()` method, shown in Example 7-14, is called only the first time when the `getMsgs()` method is called or after an exception is caught, so a retry is attempted. This limits the number of times the database connection preparation is required. If no errors are encountered, the connection to the database is maintained by the application with the container (application server) for the life of the `SelectMessagesBean` instance.

*Example 7-14 `init()` for initializing the `SelectMessagesBean`*

---

```
/** The Datasource to use. */
public static final String MSGDS = "jdbc/MessageDS";

public void init() {
    initialized = false;
    try {
        Utils.log(LOCATION, "Attempting to create connection...");
        jndiContext = new InitialContext();
        ds = (DataSource)jndiContext.lookup(MSGDS);
        connection = ds.getConnection();
        initialized = true;
    }
    catch (SQLException e)
    {
        Utils.logSQLException(LOCATION, "Error creating Database connections", e);
        close();
    }
    catch (NamingException e) {
        Utils.log(LOCATION, "Error locating JNDI references", e, false);
        close();
    }
}
```

---

To review the code step by step:

1. Create a context where the JNDI name of the resource will be located, in this case the default location of the application server's JNDI repository.
2. Look up for the data source in the container using the JNDI reference `MSGDS`, defined earlier in the code as `jdbc/MessageDS`.
3. From the data source, create a connection to the database.
4. If an exception was encountered, attempt to close any opened resources with the `close()` method so that they are not orphaned.

## Getting messages from database

Now that initialization is complete, the `getMsgs()` method needs only to do a few preparation steps.



1. First, build the query to execute as seen in Example 7-15. The SQL statement is built by concatenating the required elements. The variables used in these statements are defined earlier to map to the correct column names and table names.

*Example 7-15 Creating a query*

---

```
StringBuffer qry = new StringBuffer();
qry.append("SELECT ");
qry.append(TIME_STAMP);
qry.append(",");
qry.append(KEYWORD);
qry.append(",");
qry.append(USAGE);
qry.append(",");
qry.append(QUEUE_MANAGER)
qry.append(" FROM");
qry.append("MESSAGES_TABLE");
```

---

2. From the connection, create a statement as seen in Example 7-16.

*Example 7-16 Creating a Statement from the Connection*

---

```
Statement stmt = connection.createStatement();
```

---

3. Execute the query using the statement object and obtain a ResultSet, as seen in Example 7-17.

*Example 7-17 Retrieving a ResultSet by executing a query*

---

```
ResultSet rs = stmt.executeQuery(qry.toString());
```

---

4. The next code snippet, Example 7-18, steps through the result set. For each record in the result set and for each column in a row, a getString method is executed. The output is added to the string variable msg. Between each column we had a TAG. Each instance of the variable msg is added to the array messages that is used to present the records to the user. Using the result set obtain all the messages inserted in the table.

*Example 7-18 Using the QueueReceiver to receive TextMessages*

---

```
while(rs.next())
{
    msg = new StringBuffer();
    msg.append(rs.getString(TIME_STAMP));
    msg.append(TAG);
    msg.append(rs.getString(KEYWORD));
    msg.append(TAG);
    msg.append(rs.getString(USAGE));
```

```
msg.append(TAG);
msg.append(rs.getString(Queue_Manager));
messages.add(msg);
} while (inMessage != null);
```

---

## Closing the database connection

When the select bean itself is no longer needed, the `close()` method should always be called. This is because connections to the database will not be released by the *Java Virtual Machine* (JVM) garbage collection. To keep the number of orphaned connections pinging the database to a minimum and to return the unused connection back to connection pool maintained by the container, it is also recommended that the `close()` be attempted when a `SQLException` is thrown.

*Example 7-19 Closing the database connection*

---

```
public void close() {
    try {
        if (connection != null) {
            connection.close();
        }
    } catch (SQLException e) {
        Utils.logSQLException(LOCATION, "Error closing database connection: ",
            e);
    }
}
```

---

## 7.13 Application in action

We have now completed the development and deployment of our solution. Let us review how the applications perform in a normal, failover and fallback situation.

### 7.13.1 Normal scenario

In this section we start the application normally.

1. On the machines `qmgr1` and `qmgr2`, we start the message generators using the following parameters:

```
msggen 5 cpu cq qmgr1
msggen 5 cpu cq qmgr2
```

2. Open a browser and access the main page of the monitoring application:

```
http://was:9080/LinuxHATest/index.jsp
```

3. Select this time the option to get message from the database. See Figure 7-91.

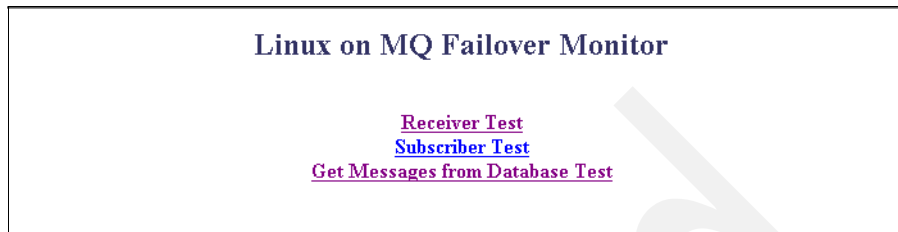


Figure 7-91 Main page of monitoring application

4. As in Figure 7-92, select an appropriate refresh interval, for example five seconds, and select a resource type, for example CPU. Click **Start Monitor**.

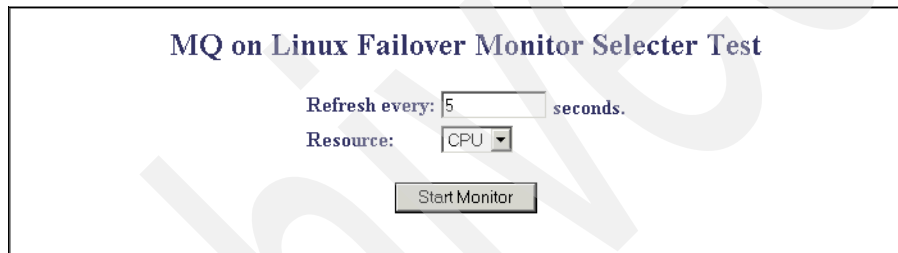


Figure 7-92 Set up the monitoring application

The results table is now shown and refreshes automatically every five seconds. As you can see in Figure 7-93 on page 300, messages from host qmgr1 and host qmgr2 reach the Web application at a constant rate. All messages are shown in the order generated. Note that there is no indication which broker has processed the messages. We know from the clustering scenario that messages are sent from qmgr1 to qmgr4 and qmgr5 using the load-balancing technique of MQ clustering. Also, messages are sent from qmgr2 to qmgr4 and qmgr5 as well. Remember that the cluster queues cq were defined with the option DEFBIND(NOTFIXED), which means that MQ can choose another instance of the cluster queue for each generated message.

Monitoring Resources	
Message Number	Message Payload
19	2004-04-08 20:06:12.0,cpu,69,qmgr2
18	2004-04-08 20:06:10.0,cpu,69,qmgr1
17	2004-04-08 20:06:07.0,cpu,25,qmgr2
16	2004-04-08 20:06:05.0,cpu,25,qmgr1
15	2004-04-08 20:06:02.0,cpu,65,qmgr2
14	2004-04-08 20:06:00.0,cpu,65,qmgr1
13	2004-04-08 20:05:57.0,cpu,9,qmgr2
12	2004-04-08 20:05:55.0,cpu,9,qmgr1
11	2004-04-08 20:05:52.0,cpu,70,qmgr2
10	2004-04-08 20:05:50.0,cpu,70,qmgr1
9	2004-04-08 20:05:47.0,cpu,2,qmgr2
8	2004-04-08 20:05:45.0,cpu,2,qmgr1
7	2004-04-08 20:05:42.0,cpu,61,qmgr2
6	2004-04-08 20:05:40.0,cpu,61,qmgr1
5	2004-04-08 20:05:37.0,cpu,36,qmgr2
4	2004-04-08 20:05:35.0,cpu,36,qmgr1

Figure 7-93 Output of the Web application in normal situations

### 7.13.2 Failover scenario

In this section, we investigate what happens when the host qmgr5 is down. We first cleared the table of the normal test and then restarted the message generators. At 20:14:45, queue manager qmgr5 was killed, which also disabled the functioning of the message broker.

Looking at the output of the Web application (see Figure 7-94 on page 301), we see that messages from qmgr1 and qmgr2 still arrive. However, the order is not any more preserved. Between the messages 7 and 8 and the messages 9 and 10, there is suddenly a gap of 10 seconds instead of the expected 5 seconds. This continues for messages 11 and 12. Starting with message 13, the gaps are getting closed. That is, messages that were on their way to brk2 have been rerouted to the working broker brk1. Note that this recovery or rerouting is completely transparent for the generation and retrieval applications.

Note also that the output shown in Figure 7-94 on page 301 seems to miss a message from qmgr2 (with time stamp 20:14:46).

## Monitoring Resources

Message Number	Message Payload
18	2004-04-08 20:15:06.0,cpu,46,qmgr2
17	2004-04-08 20:14:56.0,cpu,41,qmgr2
16	2004-04-08 20:15:11.0,cpu,8,qmgr2
15	2004-04-08 20:15:05.0,cpu,89,qmgr1
14	2004-04-08 20:14:55.0,cpu,90,qmgr1
13	2004-04-08 20:14:45.0,cpu,73,qmgr1
12	2004-04-08 20:15:01.0,cpu,30,qmgr2
11	2004-04-08 20:15:00.0,cpu,1,qmgr1
10	2004-04-08 20:14:51.0,cpu,43,qmgr2
9	2004-04-08 20:14:50.0,cpu,4,qmgr1
8	2004-04-08 20:14:41.0,cpu,89,qmgr2
7	2004-04-08 20:14:40.0,cpu,91,qmgr1
6	2004-04-08 20:14:36.0,cpu,1,qmgr2
5	2004-04-08 20:14:35.0,cpu,3,qmgr1
4	2004-04-08 20:14:31.0,cpu,90,qmgr2
3	2004-04-08 20:14:29.0,cpu,47,qmgr1
2	2004-04-08 20:11:29.0,cpu,4,qmgr2
1	2004-04-08 20:11:27.0,cpu,82,qmgr1
0	2004-04-08 20:10:44.0,cpu,78,qmgr2

Figure 7-94 Output during failure of queue manager qmgr5

### 7.13.3 Failback scenario

In this section, we restart the queue manager and see how things proceed. Note that the generators had been paused between capturing the output shown in Figure 7-94 and restarting the queue manager.

When the queue manager is restarted and the generators are resumed, the missing message is recovered as well. This demonstrates that no intervention was required to restore the load-balancing and that any work in progress on qmgr5 is recovered and processed as expected whenever the queue manager is restarted.

## Monitoring Resources

Message Number	Message Payload
25	2004-04-08 20:17:26.0,cpu,85,qmgr2
24	2004-04-08 20:17:25.0,cpu,41,qmgr1
23	2004-04-08 20:14:46.0,cpu,44,qmgr2
22	2004-04-08 20:17:21.0,cpu,78,qmgr2
21	2004-04-08 20:17:20.0,cpu,43,qmgr1
20	2004-04-08 20:15:16.0,cpu,55,qmgr2
19	2004-04-08 20:15:10.0,cpu,44,qmgr1
18	2004-04-08 20:15:06.0,cpu,46,qmgr2
17	2004-04-08 20:14:56.0,cpu,41,qmgr2
16	2004-04-08 20:15:11.0,cpu,8,qmgr2
15	2004-04-08 20:15:05.0,cpu,89,qmgr1
14	2004-04-08 20:14:55.0,cpu,90,qmgr1
13	2004-04-08 20:14:45.0,cpu,73,qmgr1
12	2004-04-08 20:15:01.0,cpu,30,qmgr2
11	2004-04-08 20:15:00.0,cpu,1,qmgr1
10	2004-04-08 20:14:51.0,cpu,43,qmgr2
9	2004-04-08 20:14:50.0,cpu,4,qmgr1
8	2004-04-08 20:14:41.0,cpu,89,qmgr2
7	2004-04-08 20:14:40.0,cpu,91,qmgr1
6	2004-04-08 20:14:36.0,cpu,1,qmgr2

Figure 7-95 Output after restarting the failed queue manager

The output shown in Figure 7-95 leads us to the following points:

1. The failover and load-balancing techniques rely on WebSphere MQ clustering. If the cluster works but one or more of the brokers are not working, then there is not going to be a failover.
2. The WebSphere MQ cluster can appear to be offline whenever one of the target queue manager is down, or the network is down to that queue manager, or the system is down. In all these cases, the messages from qmgr1 and qmgr2 will be rerouted to the still running queue manager.
3. Message order is not assured.
4. Messages can be held on the failed system until it has been restarted. New messages will find the alternative way to get a broker and thus the Web application.

## 7.14 Summary

This chapter demonstrated how two message brokers can be joined together with WebSphere MQ clustering to create a highly available broker domain with load-balancing. However, the solution can not be considered highly-available if message order and timely arrival of messages are important. The clustering technology can route messages out of order. Messages can be held on the failing server until it is restarted.

However, if this solution is combined with any of the techniques discussed in previous chapters of this redbook, we could create a highly-available solution without these disadvantages. The addition of WebSphere MQ clustering to the more hardware-related techniques gives us load-balancing and a smooth and fast failover and failback.

Archived





## Managing a distributed Linux messaging infrastructure

This chapter discusses two techniques for monitoring a messaging solution. We discuss the:

- ▶ Use of event messages that are part of WebSphere MQ
- ▶ Implementation of a monitoring tool called Monit to monitor the state of the Heartbeat software and the GPFS daemon

## 8.1 Planning monitoring

Monitoring multiple systems connected to each other with multiple technologies is never an easy problem. Also, monitoring an IT environment covers not such a few software components. It involves network monitoring, environmental monitoring (temperature, humidity and so on), hardware monitoring, and so on. The many layers in a solution should always be watched for failures, intrusions or other unexpected events.

Monitoring the availability of a system is one thing. Monitoring the overall performance is an even bigger task. It might be quite easy to detect that a process has disappeared. But how do you detect that end users are waiting a long time to get an answer from the system? If user response time is critical, you might even need to implement a simulation client that programmatically drives an interaction with the system in the same that an end-user would do.

## 8.2 Implementing WebSphere MQ event monitoring

WebSphere MQ instrumentation event messages provide information about errors, warnings, and other significant events in a queue manager. You can use these event messages to monitor the operation of queue managers, channels and queues.

An event causes a queue manager or a channel instance to put a special message, called an *event message*, on an event queue. The event message contains information about the event that we can retrieve by writing a suitable MQ application program that:

- ▶ Gets the message from the queue.
- ▶ Processes the message to extract the event data.
- ▶ Performs any action for that specific event, such as sending an e-mail.

See Figure 8-1 on page 307 for the flow of the event messages. The program can use the message in a way that fits into the business logic for example send an e-mail to a shared mail box.

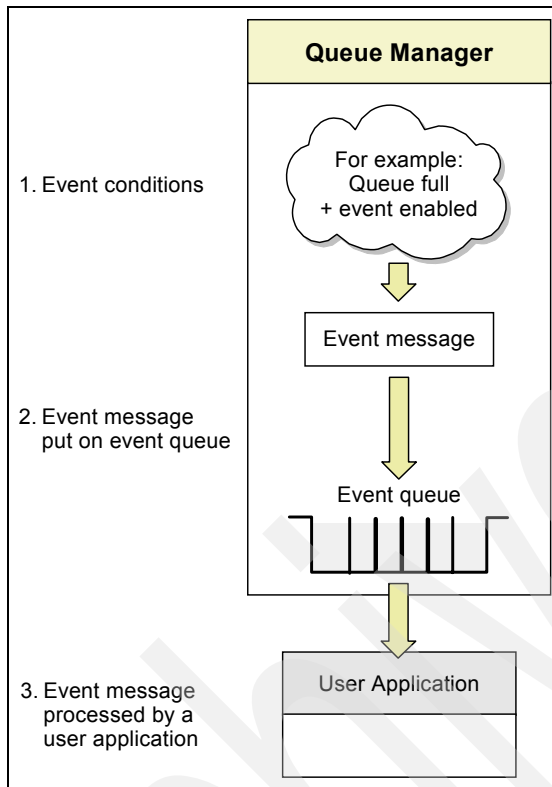


Figure 8-1 WebSphere MQ event monitoring

## 8.2.1 Queue manager events

Queue manager events are related to the use of resources within queue managers, such as an application trying to put a message to a queue that does not exist. The event messages for queue manager events are put on the `SYSTEM.ADMIN.QMGR.EVENT` queue. The following queue manager event types are supported:

- ▶ *Authority events* (AUTHOREV) report an authorization, such as an application trying to open a queue for which it does not have the required authority, or a command being issued from a user ID that does not have the required authority.
- ▶ *Inhibit events* (INHIBITEV) indicate that an MQPUT or MQGET operation has been attempted against a queue, where the queue is inhibited for puts or gets.

- ▶ *Local events* (LOCALEV) indicate that an application (or the queue manager) has not been able to access a local queue or other local object. For example, an application might try to access an object that has not been defined.
- ▶ *Remote events* (REMOTEEV) indicate that an application (or the queue manager) cannot access a (remote) queue on another queue manager. For example, the transmission queue to be used might not be correctly defined.
- ▶ *Start and stop events* (STRSTPEV) indicate that a queue manager has been started or has been requested to stop or quiesce. Stop events are not recorded unless the default message-persistence of the SYSTEM.ADMIN.QMGR.EVENT queue is defined as persistent.

### Enabling queue manager events

We enabled all the queue manager events by specifying the appropriate attribute on the MQSC command ALTER QMGR. See Example 8-1.

*Example 8-1 enable events for qmgr: qmgrevent.mqsc*

---

```
alter qmgr AUTHOREV(ENABLED)
alter qmgr INHIBTEV(ENABLED)
alter qmgr LOCALEV(ENABLED)
alter qmgr REMOTEEV(ENABLED)
alter qmgr STRSTPEV(ENABLED)
```

---

## 8.2.2 Performance events

Performance events describe conditions that can affect the performance of applications that use a specified queue. These events report that a resource has reached a threshold condition. For example, a queue depth limit might have been reached.

### Queue depth events

In WebSphere MQ applications, queues must not become full. If they do, applications can no longer put messages on the queue that they specify. Although the message is not lost if this occurs, it can be a considerable inconvenience. The number of messages can build up on a queue if the messages are being put onto the queue faster than the applications that process them can receive them.

The solution to this problem depends on the particular circumstances, but can involve:

- ▶ Diverting some messages to another queue.
- ▶ Starting new applications to take more messages off the queue.

- ▶ Stopping non-essential message traffic.
- ▶ Increasing the queue depth to overcome a transient maximum.

Clearly, having advanced warning that problems might be on their way makes it easier to take preventive action. For this purpose, the following queue depth events are provided:

- ▶ Queue Depth High events

When enabled, a Queue Depth High event is generated when a message is put on the queue, causing the queue depth to be greater than or equal to the value determined by the Queue Depth High limit.

- ▶ Queue Depth Low events

When enabled, a Queue Depth Low event is generated when a message is removed from a queue by an MQGET operation causing the queue depth to be less than or equal to the value determined by the Queue Depth Low limit.

- ▶ Queue Full events

When enabled, a Queue Full event is generated when an application is unable to put a message onto a queue because the queue is full.

**Note:** A Queue Full event is automatically enabled by a Queue Depth High or a Queue Depth Low event on the same queue. A Queue Full event automatically enables a Queue Depth Low event on the same queue

## Enabling queue depth events

By default, all queue depth events are disabled. To configure a queue for any of the queue depth events you must:

1. Enable performance events on the queue manager, using the queue manager attribute PerformanceEvent (PERFMEV in MQSC). See Example 8-2 on page 310.
2. Enable the event on the required queue by setting the following as required. See Example 8-2 on page 310.
  - QDepthHighEvent(QDPHIEV in MQSC)
  - QDepthLowEvent(QDPLOEV in MQSC)
  - QDepthMaxEvent(QDPMAXEV in MQSC)
3. If required, set the limits to the appropriate levels, expressed as a percentage of the maximum queue depth (see Example 8-2 on page 310) by setting:
  - QDepthHighLimit(QDEPTHHI in MQSC)
  - QDepthLowLimit(QDEPTHLO in MQSC).

In Example 8-2 we enable the performance events on the queue manager and enable the queue depth high, queue depth low and queue depth max. We set the threshold for the queue depth high to 80% in which case we will receive an alert to indicate a problem. When the queue depth drops to 20% we will get a new alert to indicate that the problem has been fixed.

*Example 8-2 queue depth events: qdept.mqsc*

---

```
alter qmgr PERFMEV(ENABLED)
alter ql('qmgr3') QDEPTHHI(80) QDPHIEV(ENABLED)
alter ql('qmgr3') QDEPTHLO(20) QDPLOEV(ENABLED)
alter ql('qmgr3') QDPMAXEV(ENABLED)
```

---

## Queue Service Interval events

Queue Service Interval events indicate whether a queue was serviced within a user-defined time interval called the *service interval*. Depending on the circumstances at your installation, you can use Queue Service Interval events to monitor whether messages are being taken off queues quickly enough. The following are types of queue service interval events:

- ▶ *Queue Service Interval OK* event indicates that MQGET call was performed within a user-defined time period, the service interval.
- ▶ *Queue Service Interval high* event indicates that MQGET call was not performed within a user-defined service interval.

## Enabling Queue Service Interval events

To configure a queue for Queue Service Interval events you must:

1. Enable performance events on the queue manager, using the queue manager attribute PerformanceEvent (PERFMEV in MQSC). See Example 8-3 on page 311.
2. Set the control attribute, QServiceIntervalEvent, for a Queue Service Interval High or OK event on the queue, as required (QSVCIEV in MQSC). See Example 8-3 on page 311.
3. Specify the service interval time by setting the QServiceInterval attribute for the queue to the appropriate length of time (QSVCINT in MQSC). See Example 8-3 on page 311.

In See Example 8-3 on page 311 we enable the performance events on the queue manager and enable Queue Service Interval events. We set the service interval to 1 minute, 60000 milliseconds. This will cause a queue service interval high alert to be sent when the queue has not been serviced, and no application has performed an MQGET within the specified service interval, indicating a

problem with the application reading the queue. When the situation returns to normal, we will receive an queue service interval OK alert.

*Example 8-3 Queue Service Interval events - qserver.mqsc*

---

```
alter qmgr PERFMEV(ENABLED)
alter ql('qmgr3') QSVCINT(60000) QSVCIEV(HIGH)
```

---

### 8.2.3 Channel events

Channel events are reported by channels as a result of conditions detected during their operation, such as when a channel instance is stopped. Channel events are generated:

- ▶ By a command to start or stop a channel
- ▶ When a channel instance starts or stop.
- ▶ When a channel receives a conversion error warning when getting a message
- ▶ When an attempt is made to create a channel automatically

The event is generated whether the attempt succeeds or fails.

#### Enabling channel events

Most channel events are enabled automatically and you cannot enable or disable them by command. The exceptions are the two automatic channel definition events. However, you can suppress channel events by not defining the channel events queue, or by making it put-inhibited.

**Note:** If remote event queues point to a put-inhibited channel events queue, this situation could cause a queue to fill up.

If a queue manager does not have a SYSTEM.ADMIN.CHANNEL.EVENT queue, or if this queue is put-inhibited, all channel event messages are discarded, unless they are being put by an MCA across a link to a remote queue. In this case they are put on the dead-letter queue.

### 8.2.4 Monitoring program

We created a small sample application, called monq, that monitors these queues, parses the information and creates an e-mail alert with the relevant information in the e-mail. This e-mail, see Example 8-4 on page 312, is then sent either to the administrator or a centralized system, who takes appropriate action depending on the alert. This sample program is included in the additional

materials for this redbook. Refer to Appendix D, “Additional material” on page 333 for information about obtaining this material.

The program takes the following parameters:

- ▶ `--qmgr -m <queue manager to connect to>`
- ▶ `--queue -q <queue to listen on>`
- ▶ `--from -f <source e-mail address>`
- ▶ `--to -t <destination e-mail address>`
- ▶ `--debug -d`
- ▶ `--version -V`
- ▶ `--help -?`

To make the program listen for queue manager events for qmgr5 run the following:

```
monq -m qmgr5 -q SYSTEM.ADMIN.QMGR.EVENT -f mqm@brk2 -t root@brk2
```

*Example 8-4 Alert e-mail sent as result off queue manager shutdown*

---

```
From root@brk2 Thu May6 10:06:08 2004  
Delivered-To: root@localhost
```

```
To:root@localhost  
Date: Sun, 06 May 2004 17:06:08 +0000  
Subject: WebSphere MQ event notification  
From: mqm@brk2
```

```
Queue manager: qmgr5  
Event queue: SYSTEM.ADMIN.QMGR.EVENT  
Event category: Queue manager event  
Event reason code: 2223  
Reason qualifier: queue manager stopping  
EOF
```

---

## 8.3 Implementing monitoring tool Monit

The tool Monit was introduced earlier in this redbook, see 2.3.3, “Open source offerings for Linux high availability” on page 47. In this section, we discuss a sample implementation of Monit in the lab environment used for this redbook.

### 8.3.1 Installing Monit

The Monit tool can be downloaded from:

<http://www.tildeslash.com/monit/>



1. Download the latest versions for the following packages. Create a directory `~/monitoring` and copy the tar balls to it:
  - `bison*.tar.gz`
  - `flex*.tar.gz`
  - `monit*.tar.gz`
2. Install first flex, then bison by running the following commands. Run the command:

```
tar -zxvf bison*.tar.gz
```

Once that command is completed, run the following:

```
cd bison*
./configure
make
make install
```
3. Using the same method, install Monit but add `--without-ssl` parameter to `./configure` command:

```
./configure --without-ssl
```

**Tip:** If you want to install Monit to use SSL, you also need to install OpenSSL-devel package.

4. We want Monit to start automatically and run in the background as a daemon. So we added a line in `/etc/inittab` to indicate that. See Example 8-5.

*Example 8-5 /etc/inittab*

---

```
# Run monit in standard run-levels
mo:2345:respawn:/usr/local/bin/monit -Ic /etc/monitrc -g local
```

---

## 8.3.2 Configuring Monit

The main configuration file for Monit is called `monitrc`. We created this file in location `/etc`. Example 8-6 shows this file.

*Example 8-6 /etc/monitrc*

---

```
set daemon 60
set init
set logfile syslog facility log_daemon
set statefile /var/state/monit.state
set alert root@localhost
set httpd port 2812 and user the address localhost
  allow localhost
```

---

The following subsections discuss how to include several subsystems of our scenarios in the configuration of Monit.

## Heartbeat monitoring

In the scenarios for this redbook, the heartbeat software plays a crucial role. It monitors the availability of systems in our environment. But then the question arises, what monitors this availability monitor? Monit provides a solution here.

1. Example 8-7 shows a sample script that can be used to start and stop services on a system. This script is stored on the servers brk1 and brk2.

*Example 8-7 /etc/ha.d/resource.d/monit-brk1*

---

```
#!/bin/bash
#
# sample script for starting/stopping all services on brk1
#
prog="/usr/local/bin/monit -g brk1"
start()
{
    echo -n "Starting $prog:"
    $prog start all
    echo
}
stop()
{
    echo -n "Stopping $prog:"
    $prog stop all
    echo
}
case "$1" in
    start)
        start;;
    stop)
        stop;;
    *)
        echo $"Usage: $0 {start|stop}"
        RETVAL=1
esac
exit $RETVAL
```

---

Next we need to refer to these scripts in our `/etc/ha.d/haresources` file. See Example 8-8 on page 315.

*Example 8-8 /etc/ha.d/haresources*

---

```
brk1 IPaddr::192.168.10.201/24 monit-brk1
brk2 IPaddr::192.168.10.202/24 monit-brk2
```

---

2. As the last thing we add heartbeat into Monit's configuration file. See Example 8-9. We configured heartbeat in active mode, so that Monit would try to automatically restart it, if down.

*Example 8-9 Heartbeat addition to /etc/monitrc*

---

```
check process heartbeat
  with pidfile "/var/run/heartbeat.pid"
  start program = "/etc/init.d/heartbeat start"
  stop program = "/etc/init.d/heartbeat stop"
  mode active
  GROUP local
```

---

## GPFS file system daemon monitoring

In the scenario discussed in Chapter 5, “Implementing HA queue managers: Part 3” on page 151, the availability of the data for the sample applications relied on the use of GPFS, which assured us that the disks would always be accessible from both systems in the cluster. Given the importance of GPFS for this solution, we wanted to monitor the GPFS file system daemon mmfs as without it we will not be able to access the shared fiber storage.

1. In order to do this we wrote a wrapper script, startgpfs, which creates a pid file for the daemon when started. We placed the script in /usr/local/bin/. See Example 8-10.

*Example 8-10 /usr/bin/local/startgpfs*

---

```
#!/bin/sh
/usr/lpp/mmfs/bin/mmautoload
/usr/bin/lssrc -s mmfs | tail -1 | awk '{print $3}' > /var/run/mmfsd.pid
```

---

2. We also changed the /etc/inittab to call our script startupgpfs instead of the default script mmautoload. See Example 8-11.

*Example 8-11 /etc/inittab*

---

```
# GPFS autostart
#gpfs:3:once:/usr/lpp/mmfs/bin/mmautoload >/dev/console 2>&1
gpfs:3:once:/usr/local/bin/startgpfs >/dev/console 2>&1
```

---

3. Next we added the GPFS file system daemon into Monit's configuration file /etc/monitrc. See Example 8-12 on page 316. Notice that we made the monitoring mode *passive*, meaning we only want to be notified if the file

daemon dies. This is a good idea because if the file system daemon dies, there is normally a serious underlying reason for it, such as hardware failure. In this case, a manual intervention is preferred over an automatic restart attempt.

*Example 8-12 mmfsd addition to /etc/monitrc*

---

```
check process mmfsd
  with pidfile "/var/run/mmfsd.pid"
  mode passive
  GROUP local
```

---



## Hardware and software configuration

This appendix provides details about the hardware and software configuration used for each scenario described in this redbook.

## Hardware and software used for first scenario

This scenario consisted of three machines connected to a private network. Two machines were configured for disk mirroring using the Linux-HA software. A third machine a standalone machine.

The following hardware was used:

- ▶ IBM NetFinity 5100 Server with Pentium® III processor, model 8658-51Y
- ▶ 1 GB of RAM

The mirrored machines had the following software installed:

- ▶ Red Hat Enterprise Linux AS V3.0
- ▶ Heartbeat V1.20
- ▶ DRBD V0.6.12
- ▶ WebSphere MQ V5.3 + CSD 9

The standalone machine had the following software installed:

- ▶ Red Hat Enterprise Server V2.1
- ▶ WebSphere MQ V5.3 + CSD 9
- ▶ WebSphere Application Server V5.1
- ▶ DB2 V8.2 + fixpak 3

## Hardware and software used for the second scenario

This scenario adds two more servers and an external storage unit to the network. The two additional servers have the same characteristics as the servers used for the first scenario.

The additional servers used:

- ▶ IBM Netfinity® Storage Expansion Unit EXP200 Type 3530.
- ▶ IBM ServeRAID 4M RAID controller with latest BIOS and firmware (two)

The software configuration of the additional servers is as follows:

- ▶ SUSE Linux Enterprise Server V8 Powered by United Linux V1.0 SP3
- ▶ WebSphere MQ V5.3 + CSD 9

## Hardware and software used for the third scenario

This scenario adds one more server which has the same hardware specifications as the previous scenarios with the addition of:

- ▶ A FAStT200 system

The software configuration for this machine contains:

- ▶ Windows 2000 Server + ServicePack 4
- ▶ DB2 UDB V8.2 + fixpak 3
- ▶ WebSphere MQ V5.3 + CSD 9
- ▶ WebSphere BI Message Broker V5 + CSD 4

Archived

Archived



## **Using external SCSI storage enclosure with Linux**

This appendix provides detailed information about setting up an EXP200 SCSI storage enclosure for use in a Linux environment.

## Configuring ServeRAID controllers for clustering

Use the following steps to configure ServeRAID 4M controllers for Linux High-Availability setup with EXP200 storage enclosure:

1. Before shutting down the servers, ensure that you have the latest ServeRAID driver installed on both servers. Install IBM ServeRAID manager. The Driver, BIOS and Firmware updates, plus all the ServeRAID utilities can be downloaded from:

<http://www-306.ibm.com/pc/support/site.wss/document.do?lndocid=MIGR-495PES>

2. Turn all the option switches to the off position on the EXP200, ensuring all the drives are on a single SCSI bus. See Figure B-1.

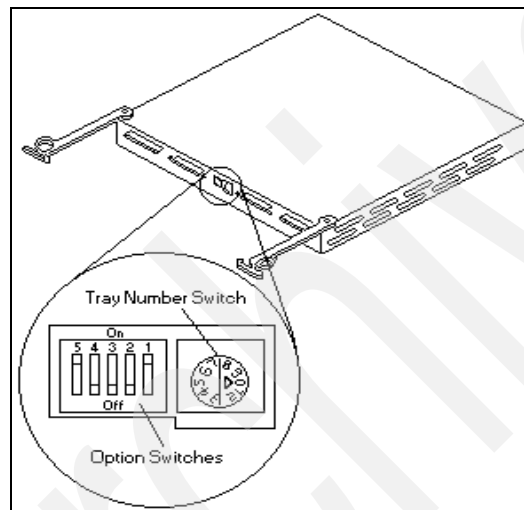


Figure B-1 Option switches on EXP200

3. Connect both ServeRAID cards to the enclosure's SCSI bus connectors.

**Restriction:** The only RAID options available for use with Linux clustering are RAID1 and RAID1e (RAID10). If you want to setup an active partition for both servers by using RAID1, you need six drives because both controllers need their own hot spare. For RAID1e setup, eight drives or more are needed. All the drives should be used either for shared or nonshared partitions.

4. Power on the EXP200 and ServerA and Insert the *IBM ServeRAID support CD* version 6.11 or later. This will automatically check if any updates are needed. If applicable, update the firmware and BIOS.

5. Ensure that the drives and the enclosure are both visible, enclosure should have SCSI ID 15. If you can't see the drives, use the scan option to locate them. Use the **Custom configuration** option to create an array of two or three ready disks and use all the available space to create a logical drive with RAID1 or RAID1e. Configure a spare.

**Tip:** Write down the SCSI IDs used for server A because these drives will also show up as ready. However, these IDs are in fact reserved and should not be used when configuring server B.

6. Use the **Configure for clustering** option to configure the controller. Select **SCSI ID 6** for all channel initiators on server A.
7. Fill in the controller name and partner name. These names are case sensitive and should correspond to the output of `uname -n` of server A and server B for clarity.
8. In Merge-group information, mark the logical drive you created as Shared and give it Merge group ID 1. See Figure B-2.

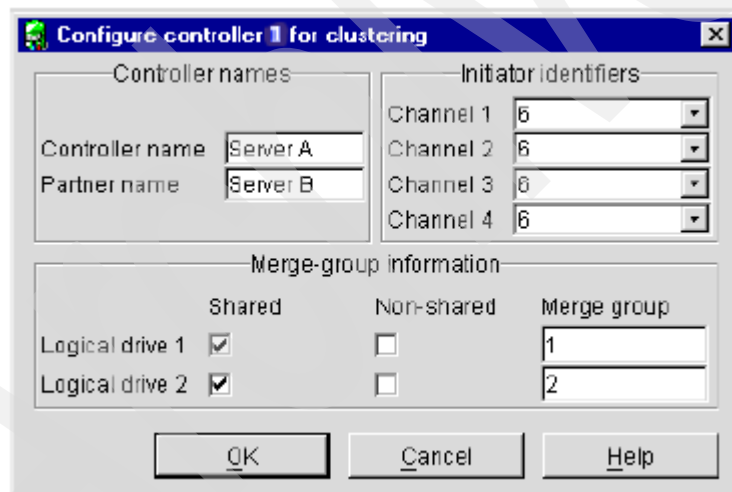


Figure B-2 Controller cluster configuration

**Note:** You can create up to eight Shared Merge groups with IDs 1-8, each can only contain one logical drive. Additionally, you can create non-shared drives. For Non-shared Merge groups you can use either ID 206 or ID 207. For example, use ID 206 for server A and 207 for server B. All non-shared drives have to be members of the same Non-shared Merge group.

9. Exit the ServeRAID manager program, eject the CD-ROM and shutdown the machine.
10. Repeat steps 4-9 for server B with the following changes:
  - a. Step 5, use the drives with SCSI ID's that have not been used for server A.
  - b. Step 6, select SCSI ID 7 for all channel initiators on server B.
  - c. Step 8, mark the logical drive you created as **Shared** and give it Merge group ID 2.
11. Boot both server A and server B.
12. On server A, create a single partition on each logical array. Each logical array will be seen as a physical drive by an operating system. If you boot off a single SCSI drive, then the ServeRAID shared drives will show up as /dev/sdb and /dev/sdc. Reboot if necessary. then format the new partitions using a journalled file system such as ext3 or JFS.
13. On server A, create mount points and mount each shared partition. See that you can successfully read and write to each drive.
14. Create mount points on server B. You will not be able to see the shared drives yet from this node. That comes next.
15. The **ipssend** utility handles the movement of the logical drives between the two nodes. The first time you pass a disk to the other controller a full synchronization is done. This will take several hours depending on your partition size. It is therefore a good idea not to initiate this before you are sure that your configuration has worked up till now. Move the disks back and forth between the servers with **ipssend** utility and check the results with ServeRAID manager. To perform a simple test do the following:
  - a. Make sure you do not have any of the shared drives mounted on either node.
  - b. Run **ipssend unmerge 1 1** on the server A. Check using the IBM ServeRAID manager that you no longer see the disk array A.
  - c. Run **ipssend merge 1 1 partner** on server B. Check using the IBM ServeRAID manager that you can see disk array A.

**Note:** At this point you will not be able to mount the disk because it needs to be added to the list of your SCSI devices in **/proc**. A reboot will fix this but for now be content if the logical array shows up in ServeRAID Manager. In Chapter 4, "Implementing HA queue managers: Part 2" on page 121, we discuss the **ServeRAID** script which handles the **ipssend** commands and update **/proc** without a reboot. It is part of the heartbeat package from linux-ha.org.

- d. Move the disk back to server A by issuing the same commands in reverse order, on server B: **ipssend unmerge 1 1** and on server A: **ipssend merge 1 1 partner**. Check that you once again have disk array A on server A.
- e. Repeat these steps for the second shared drive by passing **1 2** to the above **ipssend** commands, instead of **1 1**.

Archived

Archived



## Configuring a FAStT200 disk system for Linux

This appendix provides detailed information about setting up a FAStT200 external disk system for use in a Linux environment.

## Configuring SANbox2 to use static IP address

The following steps establish a static IP address for the SANbox2 device:

1. Connect a standard 9-pin male to 9-pin male connector to the switch and one of the servers.
2. Turn the switch on and login with the appropriate user ID and password. The following are factory defaults:
  - SanBox2 login: admin
  - Password: password
3. Type in the following and press enter:

```
admin start
set setup system
```
4. Enter the following configuration and press enter. (Accept defaults by pressing enter):
  - NetworkIPAddress: 192.168.10.40
  - NetworkIPMask: 255.255.255.0
  - GatewayIPAddress: 192.168.10.24
  - NetworkDiscovery: 1
5. Answer **yes** to saving and activating your system setup.
6. Restart the switch.

## Configuring the FAST200 disk enclosure to use a static IP-address

The following steps establish a static IP address for the FAST200 device:

1. Power up the device and check that the controller is operational.
2. Connect a serial cable that you received with your disk enclosure (RJ21 to 9-pin male) and the other end to one of the servers.
3. Start minicom
4. Power down the disk enclosure. Read all of the following instructions before executing so you are sure what to do.
  - a. Power up the disk enclosure while sending break signal, for example by pressing down **CTRL+Break**, continue pressing until you see a prompt:

Press the space bar for baud rate within 5 seconds.

- b. Press **space** to set the baud rate.



- c. Send break signal, for example press down **CTRL+Break**, until you see a prompt:

Press within 5 seconds: ESC for SHELL, BREAK for baud rate.

- d. Press **ESC** to access the controller shell.
- e. Type the password `infiniti` and press enter.
- f. Type **netCfgShow** to view to current configuration. If you want to change the configuration then type **netCfgSet**.
- g. Assign the IP address `192.168.10.30` and subnet mask `255.255.255.0`
- h. Reboot the controller.

**Note:** If you have a dual controller version, then repeat the steps for the second controller. Please make sure to use different IP addresses for the two controllers.

## Installing the Fiber Host Adapter FC2/133

A number of tasks have to be performed when installing a Fiber Host Adapter. These tasks are detailed in the following subsections.

### Updating the host adapter BIOS

To update the BIOS, perform the following steps:

1. Download the following packages from the IBM FAStT Support Web site:  
[http://www-1.ibm.com/support/docview.wss?rs=572&uid=psg1MIGR-53891&loc=en\\_US](http://www-1.ibm.com/support/docview.wss?rs=572&uid=psg1MIGR-53891&loc=en_US)
  - Latest EFI BIOS update utility for FAStT Host Adapter  
There are several available, please make sure you download the correct one.
  - IBM FAStT Host Adapter driver source code
  - IBM FAStT\_MSJ
2. Open both servers and install the cards in a 64-bit slot.
3. Connect the adapters with fiber cable to the switch and the switch to the FAStT200 storage unit.
4. Power on the switch, then the FAStT200 storage unit, wait for two minutes to allow the fiber controller to initialize.

5. Start the first server with the update diskette or CDROM.
6. The *IBM FAStT FC2 BIOS CD* contains the BIOS for the IBM FAStT FC-2 (2310), IBM FAStT FC2-133 single port (2340) and the IBM FAStT FC2-133 dual port (2342) adapters. Select the proper subdirectory for your adapter. `/efi/bios/2310`, `/efi/bios/2340`, or `/efi/bios/2342`.
7. Once you are in the proper directory, run `flasuti1 /f /1` and then `flasuti1 /u`, this will update both the adapter BIOS and NOVRAM on the adapter.

**Note:** Currently the only method to change the adapter NOVRAM settings are with FAStT\_MSJ.

8. If you are booting from the diskette, run `flasuti1 /f /1` and then `flasuti1 /u` from the default directory.
9. Reboot the server. Press **CTRL-Q** when prompted.
10. After the Fast!Util program loads, the display will depend on whether there are multiple IBM FAStT adapters installed. If there are multiple IBM FAStT adapters, a list of addresses occupied by those host adapters will appear. Using the arrow keys, select the desired adapter and press Enter.
11. Select **Host adapter settings** and change the loop reset delay to 8.
12. Select **Advanced Adapter settings** and change:
  - LUNs per target: 0
  - Enable Target Reset: **Yes**
  - Port down retry count: 12
13. Reboot the server once more.
14. Repeat the steps 5-13 for the second server brk2.

## Install the host adapter drivers

These instructions assume that you have an FC2/133 or FC2 adapter. If you have the older FAStT host adapter, you need to use the `qla2200.o` driver. Perform all the steps on both servers.

1. Make sure that the kernel-headers and kernel-sources RPM files for the supported kernel are installed.
2. Create an install directory similar to `~/i2x00-v6.06.60-fo`. Copy the file containing the drivers into this directory.
3. Unpack the driver in that directory by issuing the following command:

```
tar -zxvf *.tgz
```
4. Change to the kernel source directory for example:

```
cd /usr/src/linux
```

5. Run `make menuconfig` and verify your configuration.

**Note:** For SLES8/United Linux 1.0, you might need to run `make cloneconfig` to pull in your current kernel configuration prior to running `make menuconfig`.

6. Exit and save your kernel configuration.
7. Rebuild the dependencies to the kernel by running `make dep`.

**Note:** It is necessary to complete the step above so your source tree is correct and the drivers will load.

8. Build the driver `qla2200.o` and `qla2300.o` from the driver source code by typing:

- To make UP version of ISP2200 and ISP2300 drivers:

```
make all install
```

- To make SMP version of ISP2200 and ISP2300 drivers:

```
make all SMP=1 install
```

Using the `install` option with `make` renames the old drivers and copies your new drivers to `/lib/modules/<kernel_version>/kernel/drivers/scsi`.

**Note:** For SuSE/United Linux versions you will add `OSVER=linux` to the command line to build the device drivers, making the command:

```
make all SMP=1 OSVER=linux install
```

9. This step depends on which distribution you are running.

- Redhat

Modify `/etc/modules.conf` adding the following lines:

```
alias scsi_hostadapter0 qla2200_conf
alias scsi_hostadapter1 qla2200
```

- SUSE/United Linux

You will need to modify the `/etc/sysconfig/kernel` file to specify which modules will be added during `initrd` creation. The actual line will depend on your existing disk drivers, for example:

```
INITRD_MODULES="aic7xxx qla2300_conf qla2300 qla2200_conf qla2200"
```

**Note:** Ensure the conf module is listed before the actual driver module.

10. Change the line in `/etc/hba.conf` from:

```
gla2x00 /usr/lib/libqlsdp.so
```

To:

```
gla2300 /usr/lib/libqlsdp.so
```

11. Test the module by typing and executing the `modprobe gla2300.o` command.

12. Add the following line to the `/etc/modules.conf` file to scan for multiple LUNs at each boot:

```
options scsi_mod max_scsi_luns=128
```

## Install the FAST\_MSJ

This tool can be used to configure the host adapter configuration and change the NOVRAM settings. Perform all the steps on both servers.

1. Create a directory, `~/fc2`, and copy the tar file containing the tool into it.
2. Run `tar -zxvf *.tgz`.
3. Run `./FASTMSJ_install_linux_rev46.bin`.
4. Install the GUI and Linux Agent.
5. To run the program, run `qlremote` to start the agent and run `/usr/FAST_MSJ` to launch the agent. The default password is `config`.

## Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

### Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG246336>

Alternatively, you can go to the IBM Redbooks Web site at:

[ibm.com/redbooks](http://ibm.com/redbooks)

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG246336.

### Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
<b>comp.sh</b>	sample shell script to compile an MQ C++ program

<b>createmqm</b>	sample script to create a queue manager and move it to a network mount point
<b>imqput.cpp</b>	C++ source code for the message generation program
<b>LinuxHATestEAR.ear</b>	Enterprise archive for the Web application, including source code
<b>monq.c</b>	Sample C program to analyze MQ event messages
<b>mqsMod.pl</b>	sample perl script to edit the file mqs.ini
<b>mqstart</b>	Sample shell script to start MQ components on a system
<b>mqstatus</b>	Sample shell script to obtain MQ status information
<b>mqstop</b>	Sample shell script to stop MQ resources on a system

## System requirements for downloading the Web material

The following system configuration is recommended:

<b>Hard disk space:</b>	2 MB
<b>Operating System:</b>	Red Hat Enterprise Linux

## How to use the Web material

To use any element of the Web material, review the chapter where that element was discussed.

# Abbreviations and acronyms

<b>AIX</b>	Advanced Interactive Executive	<b>EAR</b>	Enterprise Archive
<b>AMI</b>	Application Messaging Interface	<b>ECT</b>	Enhanced Cluster Tools
<b>API</b>	Application Programming Interface	<b>EOF</b>	End Of File
<b>AS</b>	Advanced Server	<b>ERP</b>	Enterprise Resource Planning
<b>BAR</b>	Broker Archive	<b>ESCON</b>	Enterprise Systems Connection
<b>BIOS</b>	Basic Input Output System	<b>ESQL</b>	Extended (or Enhanced) SQL
<b>BIOS</b>	Basic Input Output System	<b>FTP</b>	File Transfer Protocol
<b>BSD</b>	Berkeley Software Distribution	<b>GNOME</b>	GNU Network Object Model Environment
<b>CLI</b>	Call Level Interface	<b>GNU</b>	GNU Not UNIX
<b>CMVC</b>	Configuration Management Version Control	<b>GPFS</b>	General Parallel File System
<b>COBOL</b>	Common Business Oriented Language	<b>GPL</b>	GNU General Public License
<b>CPAN</b>	Comprehensive Perl Archive Network	<b>GUI</b>	Graphical User Interface
<b>CPU</b>	Central Processing Unit	<b>HA</b>	High Availability
<b>CRC</b>	Cyclic Redundancy Check	<b>HACMP™</b>	High Availability Cluster Multi-processing
<b>CRC</b>	Cyclic Redundancy Check	<b>HPC</b>	High Performance Cluster
<b>CSD</b>	Corrective Service Diskette	<b>HPFS</b>	High Performance File System
<b>CSM</b>		<b>HTML</b>	Hypertext Markup Language
<b>DHCP</b>	Dynamic Host Configuration Protocol	<b>HTTP</b>	Hypertext Transfer Protocol
<b>DMA</b>	Direct Memory Access	<b>HTTPS</b>	Hypertext Transfer Protocol Secure
<b>DNS</b>	Domain Name System	<b>IBM</b>	International Business Machines Corporation
<b>DOS</b>	Disk Operation System	<b>ICMP</b>	Internet Control Message Protocol
<b>DPKG</b>	Debian's Package Manager	<b>ID</b>	Identifier
<b>DRBD</b>		<b>IDE</b>	Integrated Drive Electronics
<b>DTD</b>	Document Type Definition	<b>IMAP</b>	Internet Mail Access Protocol
<b>EAI</b>	Enterprise Application Integration	<b>IMAP</b>	Internet Mail Access Protocol
		<b>IO</b>	Input Output

<b>IP</b>	Internet Protocol	<b>ODBC</b>	Open Database Connectivity
<b>ITSO</b>	International Technical Support Organization	<b>OGFS</b>	Open Global File System
<b>JAAS</b>	Java Authentication and Authorization Service	<b>ONC</b>	Open Network Computing
<b>JDBC</b>	Java Database Connectivity	<b>OSCAR</b>	Open Source Cluster Application Resources
<b>JFS</b>	Journaled File System	<b>PCF</b>	Programmable Command Format
<b>JMS</b>	Java Messaging Service	<b>PCI</b>	Peripheral Component Interconnect
<b>JNDI</b>	Java Naming and Directory Interface	<b>POP</b>	Post Office Protocol
<b>JSP</b>	Javaser Page	<b>POSIX</b>	Portable Operating System Interface
<b>JVM</b>	Java Virtual Machine	<b>POST</b>	Power-on Self-test
<b>KDE</b>	Kool Desktop Environment	<b>PPP</b>	Point-to-point Protocol
<b>KRUD</b>	Kevin's Red Hat Uber Distribution	<b>PSSP</b>	Parallel System Support Programs (AIX)
<b>LAN</b>	Local Area Network	<b>PXE</b>	Preboot eXecution Environment
<b>LUI</b>	Linux Utility for Installation	<b>QCF</b>	Queue Connection Factory
<b>LUN</b>	Logical Unit Number	<b>RAD</b>	Rapid Application Development
<b>LVM</b>	Logical Volume Manager	<b>RAID</b>	Redundant Array Of Independent Disks
<b>LVS</b>	Linux Virtual Server	<b>RAM</b>	Random Access Memory
<b>LWP</b>	Light-Weight Process	<b>RAS</b>	reliability, availability, and serviceability
<b>MCA</b>	Micro-channel Architecture	<b>RDB</b>	Relational Database
<b>MINIX</b>	Mini UNIX	<b>RFH</b>	Rules and Formats Header
<b>MOM</b>	Message Oriented Middleware	<b>RPC</b>	Remote Procedure Call
<b>MQI</b>	Message Queuing Interface	<b>RPM</b>	Red Hat Package Manager
<b>MQSC</b>	MQ Scripting Command	<b>RSCT</b>	Reliable Scalable Cluster Technology
<b>MRM</b>	Message Repository Manager	<b>RVM</b>	Remote Volume Mirroring
<b>NAS</b>	Network Attached Storage	<b>SAN</b>	Storage Area Network
<b>NOVRAM</b>	Non Volatile Random Access Memory	<b>SCO</b>	Santa Cruz Operation
<b>NPTL</b>	Native POSIX Thread Library	<b>SCSI</b>	Small Computer System Interface
<b>NSD</b>	Network Shared Disk	<b>SMP</b>	Symmetric Multiprocessing
<b>NTP</b>	Network Time Protocol		
<b>NVRAM</b>	Non Volatile Random Access Memory		
<b>OAM</b>	Object Authority Manager		



<b>SNA</b>	Systems Network Architecture
<b>SNMP</b>	Simple Network Management Protocol
<b>SQL</b>	Structured Query Language
<b>SRC</b>	System Resource Controller
<b>SSA</b>	Serial Storage Architecture
<b>SSL</b>	Secure Sockets Layer
<b>SUSE</b>	Software und System Entwicklung
<b>TCP</b>	Transmission Control Protocol
<b>TDS</b>	Tagged/Delimited String Format
<b>UDB</b>	Universal Database
<b>UDP</b>	User Datagram Protocol
<b>UL</b>	United Linux
<b>UPS</b>	Uninterruptible Power Supply
<b>URL</b>	Universal Resource Locator
<b>USB</b>	Universal Serial Bus
<b>VM</b>	Virtual Machine
<b>VMS</b>	Virtual Memory System
<b>WS</b>	Workstation
<b>WSDL</b>	Web Services Description Language
<b>XML</b>	Extensible Markup Language
<b>XSL</b>	Extensible Syle Language

Archived

# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

## IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 340. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Linux Handbook A Guide to IBM Linux Solutions and Resources*, SG24-7000
- ▶ *Introduction to Storage Area Networks*, SG24-5470
- ▶ *WebSphere Application Server V5 and WebSphere MQ Family Integration*, SG24-6878-00
- ▶ *WebSphere Application Server V5 Architecture*, REDP-3721
- ▶ *WebSphere Product Overview*, REDP-3740

## Other publications

These publications are also relevant as further information sources:

- ▶ *Selecting the most appropriate JMS provider for your applications*, G325-2318-00

## Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ The GNU General Public License (GPL)  
<http://www.gnu.org/copyleft/gpl.html>
- ▶ Linux kernel archives  
<http://www.kernel.org>
- ▶ Linux High Availability  
<http://www.linux-ha.org/>

## How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)

# Index

## A

### add

- J2C user 288
- local element to message 241
- message 240
- nodes to message flow 250
- physical format to message set 233
- resources to a broker archive 282
- resources to heartbeat 63
- administration tool for a broker 218
- application vertical scaling 34
- architecture of message broker 23, 203
- asynchronous messaging
  - BIND transport type 17
- authority event 307
- availability
  - high 35

## B

### BAR file 28

- add resources 282
- create 280
- deploy 285

### Beowulf

- see OSCAR 41

### binding JMS resources at runtime 100

### binding of a database 229

### bindings transport type 17

### broker 204

- add to domain 276
- administration tool 218
- architecture 203
- create 272
- development tool 219
- error handling in flows 211
- execution group 205
- programming language 208
- start 272

### broker archive

- add resources 282
- configure 284
- create 280
- deploy 285

- file 28
- broker domain
  - create 273
  - role of configuration manager 221
- Broker Toolkit
  - definition of 25
- building message flow 207

## C

### channel

- event 311
- status 92
- client transport type 18
- close database connection 298
- cluster
  - channel 185
  - queue
    - definition of 188
  - receiver channel 185
    - definition of 187
  - sender channel 185
    - definition of 188

### Cluster Systems Management 40

### command

- dspmq 99
- dspmqver 70
- ifconfig 66
- mqscreatebroker 28, 272
- mqscreateconfigmgr 27, 271
- mqslist 227
- mqsistart 271
- runmqsc 70
- setmqcap 70
- start listener 91

### compile

- DRDB 59
- Linux kernel 58

### complex

- element 215
- type 216
- configuration manager 25–26, 221
  - create 271
  - start 271

- configure
  - broker archive 284
  - database client 230
  - DRDB 60–61
  - heartbeat 56, 63
  - Monit 313
  - TCP/IP for Linux 8
- connect
  - nodes in a message flow 251
  - to configuration manager 274
  - to JDBC resource 296
  - to remote database 230
- create
  - broker archive 280
  - broker domain 273
  - broker in domain 276
  - configuration manager 271
  - data source 290
  - database 229
  - database table 229
  - file system 62
  - JDBC Driver Provider 289
  - local queue 70
  - logical structure of message 239
  - message broker 272
  - message definition file 235
  - message flow 249
  - message flow project 247
  - message set project 232
  - queue manager 70
- CSM 40
- custom kernel 4
- customize broker archive 284

## D

- data source
  - in a message flow 259
  - in WebSphere Application Server 287
- database
  - bind 229
  - create 229
  - create table 229
  - interaction nodes 210
- DataInsert node 259
- debug a message flow 220
- decision making nodes 211
- define
  - cluster channel 271

- cluster queue 188
- cluster receiver channel 187
- cluster sender channel 188
- data source 289
- JDBC resources 289
- JMS resources 96–97
- local queue 70, 91
- queue connection factory 96
- queue destination 98
- receiver channel 91
- remote queue 88
- sender channel 88
- transmission queue 88

- delete
  - queue 200
  - queue manager 71
- deploy broker archive 285
- deployment process 286
- design of Linux 3
- development
  - of a message flow 207
  - tool for the broker 219
- display
  - broker resources 228
  - channel status 92, 190
  - queue depth 197
  - queue manager status 99
- DRBD
  - compile 59
  - definition of 48
- DRDB
  - compile 59
  - configure 60–61
  - monitor 67

## E

- ECT 41
- element
  - of a message 215
  - separation 245
- enable
  - channel events 311
  - performance events 310
  - queue depth event messages 309
  - queue manager event messages 308
  - queue service interval event 311
- end
  - queue manager 71

- enhanced cluster tools 41
- environment variables in WebSphere 289
- error handling nodes 211
- event message 306
  - authority 307
  - channel event 311
  - performance 309
  - process 312
  - queue depth 308
  - queue manager 308
  - queue service interval 310
  - types of event 307
- execute query 297
- execution group 24, 205
- ext2 46
- ext3 46
- Extreme Cluster Administration Toolkit 39

## F

- Fake 47
- Fibre Channel 37
- full repository 185

## G

- General Parallel File System 43
- General Public License 2
- global element 216
- GPFS 43, 49
  - monitor 315
- GPL 2

## H

- heartbeat 47, 53
  - configure 63
  - log files 66
  - monitor 314
- high availability 35
- horizontal scalability 35

## I

- inhibit event 307
- initialize JDBC resources 296
- input of message flow 206
- install
  - Monit 313
  - software from source 8
  - WebSphere BI Message Broker 222

- WebSphere MQ 70
- instrumentation event message 306
- ipfail 56

## J

- J2EE 29
  - definition of 29
- Java 2 Platform, Enterprise Edition 29
- Java 2 Standard Edition (J2SE) 29
- JDBC Driver Provider 289
- JDBC resource
  - initialize 296
- JFS 42
- JMS
  - API trace 103
  - resources in JNDI 96
  - tracing in WebSphere Application Server 103
- JNDI
  - data source 290
  - look-up data source 296
  - name for data source 291
  - name for JMS resource 96
  - obtain context 296
  - journaling file system
    - creating a system 62
    - definition of 42

## K

- kernel compilation 8, 58

## L

- layout of a message flow 253
- light-weight process 5
- Linux
  - custom kernel 4
  - distributions 9
  - installing software from source 8
  - kernel 3
    - archives 4
    - compile 8, 58
    - threading model 5
- Linux Virtual Server Project 41
- Linux-HA 47, 53
  - Fake 47
  - heartbeat 53
  - heartbeat code 47
- LinuxThreads 5

- local
  - element 216
  - queue
    - current depth 197
    - definition of 70
- local event 308
- log files
  - drdb 67
  - for heartbeat 66
  - for WebSphere Application Server 101
  - view 66
- logical properties of an element 242
- look-up resources in JNDI 296
- LVS
  - see Linux Virtual Server Project 41

## M

- mapping in a message flow 261
- message broker
  - development tool 219
- message
  - definition of 214
  - manipulation nodes 210
  - set 24, 214
    - add physical format 233
    - properties 234
  - set project 216
    - creating 232
  - transformation 203
- message broker 21, 27, 204
  - add to domain 276
  - administration tool 218
  - architecture 23, 203
  - built-in nodes 206
  - create 272
  - error handling in flows 211
  - execution group 205
  - programming language 208
  - requirements 28
  - start 272
- Message Broker Toolkit
  - illustration 24
- message definition
  - element types 215
  - types 216
- message definition file 215
  - add local element 241
  - add message 240

- create 235
- message flow 24, 205
  - create 249
  - default message format 255
  - development process 207
  - error handling 211
  - input and output nodes 206
  - layout 253
  - mapping 261
  - outline view 253
  - programming language 208
  - subflow 209
  - tuning 213
  - use a data source 259
  - wiring nodes 209, 251
- Message Oriented Middleware 12
- MINIX 2
- MOM 12
- Monit 48
- monitor
  - DRDB 67
  - event messages 311
  - GPFS daemon 315
  - heartbeat 66, 314
- MQInput node 255
- MQOutput node 258

## N

- Native POSIX Thread Library 5
- network failure 65
- network shared disk 45
- network-attached storage 38
- node
  - connect to each other 209
  - DataInsert 259
  - Filter 257
  - in a message flow 205
  - message manipulation 210
  - MQInput 255
  - MQOutput 258
  - properties 208
  - rename 254
  - terminal 205
  - Trace 256
- NSD 45

## O

- obtain JNDI context 296



ODBC  
  use in a message flow 259  
OGFS 46  
Open Global File System 46  
OpenGFS 46  
OSCAR 41  
output of a message flow 206

## P

partial repository 185  
performance events 308  
  enable 310  
  types of events 309  
physical properties  
  of a message set 235  
  of a type 245  
  of an element 243  
point-to-point messaging 13  
POSIX thread model 5  
Programmable Command Format 16  
properties  
  DataInsert node 259  
  Filter node 257  
  MQInput node 255  
  MQOutput node 258  
  of an element 242  
  Trace node 256  
publication 13  
publish/subscribe messaging 13  
publisher 13  
purpose of message broker 21

## Q

queue  
  current depth 197  
  define 88  
  delete 200  
queue connection factory  
  binding at runtime 100  
  definition of 96  
queue depth events 308  
  enable 309  
queue destination  
  binding at runtime 100  
  definition of 98  
queue manager  
  alter 308  
  create 70

  delete 71  
  display status 99  
  enable events 308  
  end 71  
  repository 185  
    alter 188  
queue manager events 307  
queue service interval events 310

## R

read a message from a queue 92  
receiver channel  
  definition of 91  
  status 93  
Red Hat Package Manager 70  
Redbooks Web site 340  
  Contact us xiv  
reference projects 248  
remote event 308  
remote queue  
  define 88  
  delete 200  
rename nodes in a message flow 254  
requirements for message broker 28  
RPM 70

## S

sample programs 91  
SAN  
  definition of 37  
  switch 44  
scalability  
  horizontal 35  
scalability  
  definition of 34  
  vertical 34  
scaling  
  vertical  
    application 34  
scenario one overview 52  
sender channel  
  definition of 88  
  status 92  
set capacity  
  WebSphere BI Message Broker 227  
  WebSphere MQ 70  
simple  
  element 215

- type 216
- simulate network failure 65
- start
  - configuration manager 271
  - listener 91
  - message broker 272
  - queue manager 70
- start event message 308
- status of a queue manager 99
- stop event message 308
- Storage Area Network
  - definition of 37
- subflow 209
- subscriber 13

## T

- test connection to data source 293
- Trace node 256
- trace specification for JMS 103
- transmission queue
  - define 88
- triggering channels 88
- troubleshooting WebSphere Application Server 101
- tuning of message flows 213
- type of an element 242

## U

- update IP tables 65
- user name server 25, 221
- using delimiters 245

## V

- vertical scalability 34
- view log files 66

## W

- WebSphere Application Server
  - add a J2C user 288
  - binding JMS resources at runtime 100
  - create data source 290
  - define JDBC resources 290
  - define JMS resources 96–98
  - environment variables 289
  - logging and tracing 101
  - test connection to data source 293
  - tracing JMS 103

- WebSphere BI Message Broker 24
- WebSphere MQ 14, 52
  - cluster benefits 182
  - cluster channel 185
  - define
    - channels 91
    - cluster channels 187
    - cluster queue 188
    - queues 88
    - transmission queue 88
  - display
    - channel status 92
    - queue depth 197
    - queue manager status 99
  - enable
    - channel event messages 311
    - performance events 310
    - queue manager event messages 308
  - event message 306
    - types of event 307
  - install 70
  - installed version 70
  - MQ bindings 52
  - MQ client connections 52
  - repository 185
  - set capacity 70
  - start listener 91
  - triggering channels 88
  - use a JMS provider 97
- WebSphere MQ cluster benefits 182
- WebSphere MQ Mobile Transport 213
- WebSphere MQ Script Command 16
- wire nodes 209
- write a message on a queue 91

## X

- xCAT 39



## Messaging Solutions in a Linux Environment

(0.5" spine)  
0.475" <-> 0.875"  
250 <-> 459 pages







# Messaging Solutions in a Linux Environment



**Redbooks**

**Implement  
WebSphere MQ on  
Linux for Intel**

**Implement  
WebSphere BI  
Message Broker on  
Linux**

**Learn HA for  
WebSphere MQ on  
Linux**

Recently, the adoption of Linux as a platform for business-critical applications has increased significantly. This evolution has also introduced requirements and technologies that were previously not seen in a Linux environment. Messaging and integration products, such as WebSphere MQ and WebSphere BI Message Broker, can now be used on Linux to solve integration problems that include Linux platforms.

This IBM Redbook starts with an introduction of messaging and integration technologies to Linux users. It also introduces the world of Linux to people that are familiar with messaging and integration technologies on other platforms.

This book ends with a discussion of a few monitoring techniques that can be implemented in a Linux environment, so that a messaging solution can be monitored.

**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](http://ibm.com/redbooks)**