

WebSphere Application Server V6.1:

System Management and Configuration

**Learn about WebSphere Application
Server**

**Configure and administer a
WebSphere system**

Deploy applications



**Carla Sadtler, Fabio Albertoni,
Bernardo Fagalde, Thiago
Kleinubing, Henrik Sjostrand, Ken
Worland, Lars Bek Laursen, Martin
Phillips, Martin Smithson,
Kwan-Ming Wan**



International Technical Support Organization

**WebSphere Application Server V6.1:
System Management and Configuration**

November 2006

Archived

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

First Edition (November 2006)

This edition applies to IBM WebSphere Application Server V6.1, IBM WebSphere Application Server Network Deployment V6.1, and IBM WebSphere Application Server for z/OS V6.1.

© Copyright International Business Machines Corporation 2006. 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

Notices	xv
Trademarks	xvi
Preface	xvii
The team that wrote this redbook	xvii
Become a published author	xxi
Comments welcome	xxii
Part 1. The basics	1
Chapter 1. WebSphere Application Server	3
1.1 Product overview	4
1.2 WebSphere Application Server	5
1.3 Packaging	7
1.4 Supported platforms and software	10
1.4.1 Operating systems	10
1.4.2 Web servers	11
1.4.3 Database servers	12
1.4.4 Directory servers	12
Chapter 2. System management: A technical overview	15
2.1 System management overview	16
2.1.1 System management in a stand-alone server environment	16
2.1.2 System management in a distributed server environment	17
2.2 Java Management Extensions (JMX)	19
2.2.1 JMX architecture	19
2.2.2 JMX distributed administration	22
2.2.3 JMX MBeans	24
2.2.4 JMX usage scenarios	24
2.2.5 J2EE management	25
2.3 Distributed administration	26
2.3.1 Distributed process discovery	28
2.3.2 Centralized changes to configuration and application data	31
2.3.3 File synchronization	32
2.4 Configuration and application data repository	39
2.4.1 Repository directory structure	39
2.4.2 Variable scoped files	42
2.4.3 Application data files	42

Chapter 3. Getting started with profiles	47
3.1 Understanding profiles	48
3.1.1 Types of profiles	50
3.1.2 Directory structure and default profiles	51
3.2 Building a system using profiles	54
3.2.1 Stand-alone server environment	55
3.2.2 Distributed server environment	55
3.3 Creating profiles on distributed systems (non z/OS)	57
3.3.1 Creating a deployment manager profile	59
3.3.2 Creating an application server profile	67
3.3.3 Creating a cell profile	78
3.3.4 Creating a custom profile	79
3.3.5 Federating a custom node to a cell	86
3.3.6 Creating a new application server on an existing node	88
3.3.7 Federating an application server profile to a cell	91
3.4 Creating profiles on z/OS systems	95
3.5 Managing profiles	123
3.5.1 Using the manageprofiles command	123
3.5.2 Creating a profile	124
3.5.3 Deleting profiles	126
3.6 Managing the processes	127
3.6.1 Starting a distributed server environment	128
3.6.2 Stopping the distributed server environment	129
3.6.3 Enabling process restart on failure	130
Chapter 4. Administration basics	137
4.1 Introducing the WebSphere administrative console	138
4.1.1 Starting the administrative console	138
4.1.2 Logging in to the administrative console	140
4.1.3 Changing the administrative console session timeout	142
4.1.4 The graphical interface	143
4.1.5 Finding an item in the console	147
4.1.6 Updating existing items	151
4.1.7 Adding new items	153
4.1.8 Removing items	154
4.1.9 Starting and stopping items	154
4.1.10 Using variables	156
4.1.11 Saving work	157
4.1.12 Getting help	158
4.2 Securing the administrative console	159
4.3 Working with the deployment manager	162
4.3.1 Deployment manager configuration settings	162
4.3.2 Starting and stopping the deployment manager	166

4.4 Working with application servers	170
4.4.1 Creating an application server	171
4.4.2 Viewing the status of an application server	175
4.4.3 Starting an application server	178
4.4.4 Stopping an application server	181
4.4.5 Viewing run time attributes of an application server	185
4.4.6 Customizing application servers	188
4.5 Working with nodes	201
4.5.1 Adding (federating) a node	201
4.5.2 Removing a node	209
4.5.3 Renaming a node	212
4.5.4 Node agent synchronization	213
4.5.5 Starting and stopping nodes	215
4.5.6 Node groups	219
4.6 Working with clusters	222
4.6.1 Creating clusters	222
4.6.2 Viewing cluster topology	226
4.6.3 Managing clusters	226
4.7 Working with virtual hosts	227
4.7.1 Creating a virtual host	228
4.8 Managing applications	229
4.8.1 Using the administrative console to manage applications	230
4.8.2 Installing an enterprise application	231
4.8.3 Uninstalling an enterprise application	233
4.8.4 Exporting an enterprise application	233
4.8.5 Starting an enterprise application	234
4.8.6 Stopping an enterprise application	234
4.8.7 Preventing an enterprise application from starting on a server	234
4.8.8 Viewing application details	235
4.8.9 Finding a URL for a servlet or JSP	238
4.9 Managing your configuration files	242
4.9.1 Backing up a profile	243
4.9.2 Restoring a profile	244
4.9.3 Exporting and importing profiles	246
Chapter 5. Administration with scripting	249
5.1 Overview of WebSphere scripting	250
5.2 Using wsadmin	250
5.2.1 Jacl versus Jython	250
5.2.2 Launching wsadmin	251
5.2.3 Configuring wsadmin	252
5.2.4 Command and script invocation	254
5.2.5 Overview of wsadmin objects	257

5.2.6 Management using wsadmin objects	259
5.3 Common operational tasks using wsadmin	278
5.3.1 General approach for operational tasks	278
5.3.2 Examples of common administrative tasks	279
5.3.3 Managing the deployment manager	279
5.3.4 Managing nodes	280
5.3.5 Managing application servers	281
5.3.6 Managing enterprise applications	283
5.3.7 Managing clusters	285
5.3.8 Generating the Web server plug-in configuration	286
5.3.9 Enabling tracing for WebSphere components	286
5.4 Common configuration tasks	288
5.4.1 General approach for configuration tasks	288
5.4.2 Specific examples of WebSphere configuration tasks	288
5.5 Help creating wsadmin scripts	300
5.6 Using Java for administration	301
Online resources	302
Chapter 6. Configuring WebSphere resources	303
6.1 WebSphere resources	304
6.2 JDBC resources	305
6.2.1 What are JDBC providers and data sources?	306
6.2.2 WebSphere support for data sources	307
6.2.3 Creating a data source	311
6.2.4 Creating a JDBC provider	311
6.2.5 Creating JDBC data source	317
6.3 JCA resources	329
6.3.1 WebSphere Application Server JCA support	331
6.3.2 Installing and configuring resource adapters	333
6.3.3 Configuring J2C connection factories	337
6.3.4 Using resource adapters from an application	340
6.4 JavaMail resources	342
6.4.1 JavaMail sessions	343
6.4.2 Configuring the mail provider	343
6.4.3 Configuring JavaMail sessions	347
6.4.4 Example code	350
6.5 URL providers	351
6.5.1 Configuring URL providers	351
6.5.2 Configuring URLs	353
6.5.3 URL provider sample	354
6.6 Resource environment providers	355
6.6.1 Resource environment references	356
6.6.2 Configuring the resource environment provider	357

6.7 Resource authentication	361
6.8 More information	363
Chapter 7. Managing Web servers.	365
7.1 Web server support overview	366
7.1.1 Request routing using the plug-in	366
7.1.2 Web server and plug-in management	367
7.2 Working with Web servers	371
7.2.1 Defining nodes and Web servers	371
7.2.2 Viewing the status of a Web server	376
7.2.3 Starting and stopping a Web server	376
7.2.4 IBM HTTP Server remote administration	378
7.2.5 Mapping modules to servers	383
7.3 Working with the plug-in configuration file	385
7.3.1 Regenerating the plug-in configuration file	386
7.3.2 Propagating the plug-in configuration file	392
7.3.3 Modifying the plug-in request routing options	393
Part 2. Messaging with WebSphere.	397
Chapter 8. Asynchronous messaging	399
8.1 Messaging concepts	400
8.1.1 Loose coupling	400
8.1.2 Messaging types	401
8.1.3 Destinations	401
8.1.4 Messaging models	402
8.1.5 Messaging patterns	404
8.2 Java Message Service	406
8.2.1 JMS API history	406
8.2.2 JMS providers	407
8.2.3 JMS domains	407
8.2.4 JMS administered objects	408
8.2.5 JMS and JNDI	409
8.2.6 JMS Connections	411
8.2.7 JMS sessions	412
8.2.8 JMS messages	413
8.2.9 JMS message producers	415
8.2.10 JMS message consumers	415
8.2.11 JMS exception handling	419
8.2.12 Application Server Facilities	421
8.2.13 JMS and J2EE	422
8.3 Messaging in the J2EE Connector Architecture	422
8.3.1 Message endpoints	425
8.3.2 MessageEndpointFactory	425

8.3.3	Resource adapters	425
8.3.4	JMS ActivationSpec JavaBean	428
8.3.5	Message endpoint deployment	430
8.3.6	Message endpoint activation	431
8.3.7	Message delivery	432
8.3.8	Administered objects	433
8.4	Message-driven beans	434
8.4.1	Message-driven bean types	434
8.4.2	Client view of a message-driven bean	435
8.4.3	Message-driven bean implementation	435
8.4.4	Message-driven bean life cycle	437
8.4.5	Message-driven beans and transactions	439
8.4.6	Message-driven bean activation configuration properties	443
8.4.7	Associating a message-driven bean with a destination	445
8.4.8	Message-driven bean best practices	448
8.5	Managing WebSphere JMS providers	451
8.5.1	Managing the default messaging JMS provider	451
8.5.2	Managing the WebSphere MQ JMS provider	455
8.5.3	Managing a generic JMS provider	457
8.6	Configuring WebSphere JMS administered objects	461
8.6.1	Common administration properties	462
8.6.2	Configuring the default messaging JMS provider	462
8.6.3	Configuring the WebSphere MQ JMS provider	491
8.6.4	Configuring listener ports	511
8.6.5	Configuring a generic JMS provider	515
8.7	Connecting to a service integration bus	520
8.7.1	JMS client run time environment	521
8.7.2	Controlling messaging engine selection	524
8.7.3	Load balancing bootstrapped clients	534
8.8	References and resources	536
Chapter 9	Default messaging provider	539
9.1	Concepts and architecture	540
9.1.1	Buses	540
9.1.2	Bus members	541
9.1.3	Messaging engines	541
9.1.4	Message stores	547
9.1.5	Destinations	549
9.1.6	Mediations	555
9.1.7	Foreign buses	555
9.2	Run time components	561
9.2.1	SIB service	561
9.2.2	Service integration bus transport chains	563

9.2.3	Message stores	568
9.2.4	Exception destinations	579
9.2.5	Service integration bus links	580
9.2.6	WebSphere MQ links	584
9.2.7	WebSphere MQ Servers	592
9.3	High availability and workload management	594
9.3.1	Cluster bus members for high availability	594
9.3.2	Cluster bus members for workload management	594
9.3.3	Partitioned queues	595
9.3.4	JMS clients connecting into a cluster of messaging engines	596
9.3.5	Preferred servers and core group policies	597
9.3.6	Best practices	600
9.4	Service integration bus topologies	601
9.4.1	One server in the cell is a member of one bus	601
9.4.2	Every server in the cell is a member of the same bus	602
9.4.3	A single cluster bus member and one messaging engine	602
9.4.4	A cluster bus member with multiple messaging engines	603
9.4.5	Mixture of cluster and server bus members	603
9.4.6	Multiple buses in a cell	604
9.5	Service integration bus and message-driven beans	605
9.5.1	Message-driven beans connecting to the bus	605
9.5.2	MDBs and clusters	607
9.6	Service integration bus security	608
9.7	Problem determination	610
9.8	Configuration and management	612
9.8.1	SIB service configuration	613
9.8.2	Creating a bus	616
9.8.3	Configuring bus properties	616
9.8.4	Enabling bus security	618
9.8.5	Adding a bus member	622
9.8.6	Creating a queue destination	637
9.8.7	Creating a topic space destination	639
9.8.8	Creating an alias destination	640
9.8.9	Adding messaging engines to a cluster	642
9.8.10	Setting up preferred servers	643
9.8.11	Setting up a foreign bus link to a service integration bus	650
9.8.12	Setting up a foreign bus link to an MQ queue manager	656
9.8.13	Creating a foreign destination	666
Part 3	Working with applications	669
Chapter 10	Session management	671
10.1	HTTP session management	672

10.2	Session manager configuration	672
10.2.1	Session management properties	672
10.2.2	Accessing session management properties	673
10.3	Session scope	674
10.4	Session identifiers	676
10.4.1	Choosing a session tracking mechanism	676
10.4.2	SSL ID tracking	678
10.4.3	Cookies	679
10.4.4	URL rewriting	682
10.5	Local sessions	683
10.6	General properties for session management	685
10.7	Session affinity	688
10.7.1	Session affinity and failover	690
10.8	Persistent session management	692
10.8.1	Enabling database persistence	694
10.8.2	Memory-to-memory replication	698
10.8.3	Session management tuning	708
10.8.4	Persistent sessions and non-serializable J2EE objects	715
10.8.5	Larger DB2 page sizes and database persistence	716
10.8.6	Single and multi-row schemas (database persistence)	717
10.8.7	Contents written to the persistent store using a database	719
10.9	Invalidating sessions	723
10.9.1	Session listeners	723
10.10	Session security	725
10.11	Session performance considerations	726
10.11.1	Session size	727
10.11.2	Reducing persistent store I/O	730
10.11.3	Multirow persistent sessions: Database persistence	731
10.11.4	Managing your session database connection pool	732
10.11.5	Session database tuning	733
10.12	Stateful session bean failover	734
10.12.1	Enabling stateful session bean failover	734
10.12.2	Stateful session bean failover considerations	737
Chapter 11.	WebSphere naming implementation	741
11.1	Features	742
11.2	WebSphere naming architecture	743
11.2.1	Components	743
11.2.2	JNDI support	744
11.2.3	JNDI bindings	745
11.2.4	Federated name space	746
11.2.5	Local name space structure	749
11.3	Interoperable Naming Service (INS)	757

11.3.1 Bootstrap ports	757
11.3.2 CORBA URLs	757
11.4 Distributed CosNaming	759
11.5 Configured bindings	760
11.5.1 Types of objects	761
11.5.2 Types of binding references	762
11.6 Initial contexts	763
11.6.1 Setting initial root context	766
11.7 Federation of name spaces	769
11.8 Foreign cell bindings	770
11.9 Interoperability	771
11.9.1 WebSphere V4.0 EJB clients	772
11.9.2 WebSphere V4.0 server	773
11.9.3 EJB clients hosted by non-WebSphere environment	773
11.10 Examples	774
11.10.1 Single server	775
11.10.2 Two single servers on the same box	776
11.10.3 Network Deployment application servers on the same box	778
11.10.4 WebSphere Application Server V4 client	780
11.11 Naming tools	782
11.11.1 dumpNameSpace	782
11.12 Configuration	784
11.12.1 Name space bindings	785
11.12.2 Foreign cell bindings	788
11.12.3 CORBA naming service users and groups	789
Chapter 12. Understanding class loaders	795
12.1 A brief introduction to Java class loaders	796
12.2 WebSphere class loaders overview	800
12.2.1 WebSphere extensions class loader	801
12.2.2 Application and Web module class loaders	802
12.2.3 Handling JNI code	803
12.3 Configuring WebSphere for class loaders	804
12.3.1 Class loader policies	804
12.3.2 Class loading/delegation mode	807
12.3.3 Shared libraries	809
12.4 Class loader viewer	810
12.5 Learning class loaders by example	811
12.5.1 Step 1: Simple Web module packaging	812
12.5.2 Step 2: Adding an EJB module and Utility jar	817
12.5.3 Step 3: Changing the WAR class loader delegation mode	818
12.5.4 Step 4: Sharing utility JARs using shared libraries	820
12.6 Additional class loader diagnostics	827

Chapter 13. Packaging applications	829
13.1 Plants by WebSphere sample application	830
13.1.1 Plants by WebSphere resources used	831
13.2 Packaging using the Application Server Toolkit	832
13.2.1 Import source code	832
13.2.2 Working with deployment descriptors	838
13.3 Setting application bindings	842
13.3.1 Defining EJB JNDI names	842
13.3.2 Binding EJB and resource references	844
13.3.3 Defining data sources for entity beans	846
13.3.4 Setting the context root for Web modules	853
13.4 IBM EJB extensions: EJB caching options	854
13.4.1 EJB container caching option for entity beans	854
13.4.2 EJB container caching option for stateful session beans	858
13.4.3 Stateful EJB timeout option	859
13.5 IBM EJB extensions: EJB access intents	860
13.5.1 Transaction isolation levels overview	860
13.5.2 Concurrency control	862
13.5.3 Using EJB 2.x access intents	863
13.5.4 Using read-ahead hints	868
13.5.5 Tracing access intents behavior	871
13.6 IBM EJB extensions: inheritance relationships	871
13.7 IBM Web module extensions	872
13.7.1 File serving servlet	872
13.7.2 Web application auto reload	873
13.7.3 Serve servlets by class name	873
13.7.4 Default error page	874
13.7.5 Directory browsing	874
13.7.6 JSP attributes	874
13.7.7 Automatic HTTP request and response encoding	875
13.8 IBM EAR extensions: Sharing session context	876
13.9 Exporting the PlantsByWebSphere EAR file	877
13.10 WebSphere Enhanced EAR	877
13.10.1 Configuring a WebSphere Enhanced EAR	879
13.11 Packaging recommendations	891
Chapter 14. Deploying applications	893
14.1 Preparing the environment	894
14.1.1 Creating the Plants by WebSphere DB2 database	894
14.1.2 Creating an environment variable	895
14.1.3 Creating the Plants by WebSphere application server	896
14.1.4 Defining the Plants by WebSphere virtual host	901
14.1.5 Creating the virtual host for IBM HTTP Server and Apache	902

14.1.6 Creating a DB2 JDBC provider and data source	904
14.2 Generating deployment code	911
14.2.1 Using EJBDeploy command-line tool	911
14.3 Deploying the application	913
14.3.1 Using a bindings file	919
14.4 Deploying application clients	920
14.4.1 Defining application client bindings	924
14.4.2 Launching the J2EE client	925
14.5 Updating applications	928
14.5.1 Replacing an entire application EAR file	928
14.5.2 Replacing or adding an application module	929
14.5.3 Replacing or adding single files in an application or module	930
14.5.4 Removing application content	931
14.5.5 Performing multiple updates to an application or module	931
14.5.6 Rolling out application updates to a cluster	934
14.5.7 Hot deployment and dynamic reloading	938
Related publications	941
IBM Redbooks	941
Other publications	942
Online resources	942
How to get IBM Redbooks	944
Help from IBM	944
Index	945

Notices

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

Trademarks

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

AIX 5L™	ibm.com®	Redbooks™
AIX®	IBM®	Redbooks (logo)  ™
alphaWorks®	IMS™	RACF®
Cloudscape™	Informix®	SupportPac™
CICS®	iSeries™	System z™
Domino®	Lotus®	Tivoli®
DB2 Connect™	MVS™	WebSphere®
DB2 Universal Database™	MVS/ESA™	Workplace™
DB2®	OS/400®	z/OS®
e-business on demand®	Power PC®	zSeries®
i5/OS®	Rational®	

The following terms are trademarks of other companies:

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

iPlanet, Enterprise JavaBeans, EJB, Java, Java Naming and Directory Interface, Javadoc, JavaBeans, JavaMail, JavaServer, JavaServer Pages, JDBC, JDK, JMX, JRE, JSP, JVM, J2EE, J2SE, Solaris, Streamline, Sun, Sun Java, Sun Microsystems, Sun ONE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Active Directory, ActiveX, Microsoft, Visual Basic, Windows NT, Windows Server, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

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

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

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

Preface

This IBM® Redbook provides system administrators, developers, and architects with the knowledge to configure a WebSphere Application Server V6.1 run time environment, to package and deploy Web applications, and to perform ongoing management of the WebSphere® environment.

One in a series of handbooks, the entire series is designed to give you in-depth information about the entire range of WebSphere Application Server products. In this IBM Redbook, we provide a detailed exploration of the WebSphere Application Server V6.1 run time environments and administration process.

The IBM Redbook includes configuration and administration information for WebSphere Application Server V6.1 and WebSphere Application Server Network Deployment V6.1 on distributed platforms (excluding iSeries™) and WebSphere Application Server for z/OS® V6.1.

The following are considered companion pieces to this IBM Redbook:

- ▶ *WebSphere Application Server V6.1: Technical Overview*, REDP-4191 at:
<http://www.redbooks.ibm.com/redpieces/abstracts/redp4191.html>
- ▶ *WebSphere Application Server V6.1: Planning and Design*, SG24-7305 at:
<http://www.redbooks.ibm.com/redpieces/abstracts/sg247305.html>

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, Poughkeepsie Center.

Carla Sadtler is a certified IT Specialist at the ITSO, Raleigh Center. She writes extensively about the WebSphere and Patterns for e-business areas. Before joining the ITSO in 1985, Carla worked in the Raleigh branch office as a Program Support Representative. She holds a degree in mathematics from the University of North Carolina at Greensboro.

Fabio Albertoni is a Senior IT Specialist working from Integrated Technology Delivery SSO, on Hortolandia, Brazil. He has nine years of experience work in IT Industry and Banks, he has spent last five years developing and implementing integrated solutions using WebSphere Application Server and MQ-Series. He hold a degree in Data Process from FATEC University of Ourinhos and a Master degree on Computer Engineer from Instituto de Pesquisas Tecnologicas of Sao Paulo, Brazil.

Bernardo Fagalde is an IT Architect at IBM Uruguay, working for IBM since 2000. During his time at IBM, he has had many positions, including database administrator, system administrator, developer, designer, application server administrator, and as a technical lead for e-business projects. He has been working with WebSphere Application Server since V3.5 and mostly designs e-business solutions focused on using the WebSphere product family. He is currently the lead IT Architect on a large J2EE™ project. Bernardo holds a Computing Engineer degree from the Uruguayan main University (Universidad de la República Oriental del Uruguay).

Thiago Kleinubing is an IT Specialist in Brazil and has over nine years of experience in the IT field. He has been working at IBM for the last six years and is currently a Team Leader for the IBM Global Business Services Organization - Total Workplace™ Experience Center of Excellence. His areas of expertise include the architecture, design, and development of J2EE applications. He is also an expert in IBM WebSphere Application Server, performance tuning, and problem determination. Thiago holds a degree in computer science and is certified in Rational® Application Developer and WebSphere Studio V5.

Henrik Sjostrand is a Senior IT Specialist and has worked for IBM Sweden for 12 years. He is currently working as a technical consultant for the Nordic IBM Software Services for WebSphere team. For the last six years, he has focused on J2EE application development and WebSphere Application Server architecture, deployment, performance tuning, and troubleshooting. He is certified in WebSphere Application Server V4, V5, and V6, WebSphere Studio V5, and Rational Application Developer V6. Henrik holds a Master of Science in Electrical Engineering from Chalmers University of Technology in Gothenburg, Sweden, where he lives.

Ken Worland is a senior IT Specialist based in Melbourne, Australia. He specializes in Web services and messaging solutions and has over 15 years of experience in the IT field. His areas of expertise include WebSphere Application Server, WebSphere MQ, DB2®, Oracle, and much more, having worked as a UNIX® system administrator and database administrator on occasion. Ken holds a Bachelors degree in Computer Science from LaTrobe University in Melbourne.

Special thanks to the authors of the previous book, *WebSphere Application Server V6 System Management & Configuration Handbook*, SG24-6451:

Lars Bek Laursen was an Advisory IT Specialist at the Integrated Technology Services division of IBM Global Services in Lyngby, Denmark while working on this project. He has eight years of Java™ experience, from developing Java-based systems management solutions to designing and implementing enterprise application server environments. For the last five years, Lars has worked extensively as a WebSphere Application Server consultant, advising on problem solving, tuning, and implementation of fail-safe run time environments. Lars holds a Master of Science in Engineering degree from the Technical University of Denmark. Lars has since left IBM.

Martin Phillips is a tester for the WebSphere Messaging and Transaction Technology team in the Hursley Laboratories in the UK. He has worked for IBM UK for five years as a tester for WebSphere Application Server. His areas of expertise include the service integration bus, about which he writes extensively in this book. Martin holds a Master of Science in Information Technology specializing in Software and Systems from the University of Glasgow.

Martin Smithson is a Senior IT Specialist working for IBM Software Group in Hursley, England. He has nine years of experience working in the IT Industry and has spent the last four years working as a technical consultant for the EMEA IBM Software Services for WebSphere team. He is certified in WebSphere Application Server V3.5, V4 and V5 and WebSphere Studio Application Developer V4.0.3 and V5. His areas of expertise include the architecture, design, and development of J2EE applications. He is also an expert on IBM WebSphere Application Server. He has written extensively on asynchronous messaging and the service integration bus. He holds a degree in Software Engineering from City University in London, UK.

Kwan-Ming Wan is a Consulting IT Specialist working for the IBM Software Group in London, England. He has over fifteen years of experience in the IT industry and has been working as a consulting professional throughout his career. For the past five years, he has been working as a WebSphere consultant with a focus on performance tuning, problem determination, and architecture design. He holds a Master of Science degree in Information Technology from the University of Nottingham, England.

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

Margaret Ticknor
International Technical Support Organization, Raleigh Center

Rich Conway
International Technical Support Organization, Poughkeepsie Center

Mollie Tucker
IBM Intern from North Carolina State University

Daniel Tishman
IBM Intern from Penn State University

Sam Cleveland
WebSphere Application Server Samples Development

The following members of the WebSphere Messaging and Transaction
Technologies Team, IBM Hursley:

Malcolm Ayres
David Currie
Sarah Hemmings
Graham Hopkins
Geraint Jones
Adrian Preston
Anne Redwood
Matthew Vaughton
Graham Wallis



Figure 1 Authors: (from left to right) Fabio Albertoni, Carla Sadtler, Thiago Kleinubing, Bernardo Fagalde, Ken Worland, Henrik Sjostrand

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 have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- Send your comments in an e-mail to:

redbooks@us.ibm.com

- Mail your comments to:

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



Part 1

The basics

This part introduces you to WebSphere Application Server V6.1. It includes information about the run time architecture, administration tools, and the basics of configuring and managing the run time environment.

This part includes the following:

- ▶ Chapter 1, “WebSphere Application Server” on page 3
- ▶ Chapter 2, “System management: A technical overview” on page 15
- ▶ Chapter 3, “Getting started with profiles” on page 47
- ▶ Chapter 4, “Administration basics” on page 137
- ▶ Chapter 5, “Administration with scripting” on page 249
- ▶ Chapter 6, “Configuring WebSphere resources” on page 303
- ▶ Chapter 7, “Managing Web servers” on page 365

WebSphere Application Server

IBM WebSphere is the leading software platform for e-business on demand®. Providing comprehensive e-business leadership, WebSphere is evolving to meet the demands of companies faced with challenging business requirements, such as the need for increasing operational efficiencies, strengthening client loyalty, and integrating disparate systems. WebSphere provides answers in today's challenging business environments.

IBM WebSphere is architected to enable you to build business-critical applications for the Web. WebSphere includes a wide range of products that help you develop and serve Web applications. They are designed to make it easier for clients to build, deploy, and manage dynamic Web sites more productively.

In this chapter, we take a look at the new WebSphere Application Server V6.1 for distributed platforms and WebSphere Application Server for z/OS.

1.1 Product overview

WebSphere is the IBM brand of software products designed to work together to help deliver dynamic e-business quickly. It provides solutions for connecting people, systems, and applications with internal and external resources. WebSphere is based on infrastructure software, or middleware, designed for dynamic e-business. It delivers a proven, secure, and reliable software portfolio that can provide an excellent return on investment.

The technology that powers WebSphere products is Java. Over the years, many software vendors have collaborated on a set of server-side application programming technologies that help build Web accessible, distributed, and platform-neutral 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, with which most clients are familiar. J2SE supports the development of client-side applications with rich graphical user interfaces (GUIs). The J2EE platform is built on top of the J2SE platform. J2EE 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 clients is tremendous. Among the benefits of J2EE are:

- ▶ 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, such as security and transaction support. This improves productivity and shortens development cycles.
- ▶ Industry standard technologies allow clients 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.

Another exciting opportunity for IT is Web services. Web services allow for the definition of functions or services within an enterprise that can be accessed using industry standard protocols that most businesses already use today, such as HTTP and XML. This allows for easy integration of both intra- and inter-business applications that can lead to increased productivity, expense reduction, and quicker time to market.

1.2 WebSphere Application Server

WebSphere Application Server provides the environment to run your Web-enabled e-business applications. An application server functions 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 (for example, DB2 UDB for iSeries) and the business logic (for example, traditional business applications, such as order processing). The middle tier is WebSphere Application Server, which provides a framework for a consistent and architected link between the HTTP requests and the business data and logic.

WebSphere Application Server is available on a wide range of platforms and in multiple packages to meet specific business needs. It also serves as the base for other WebSphere products, such as WebSphere Enterprise Service Bus and WebSphere Process Server, by providing the application server that is required to run these specialized applications.

Figure 1-1 illustrates a product overview of WebSphere Application Server.

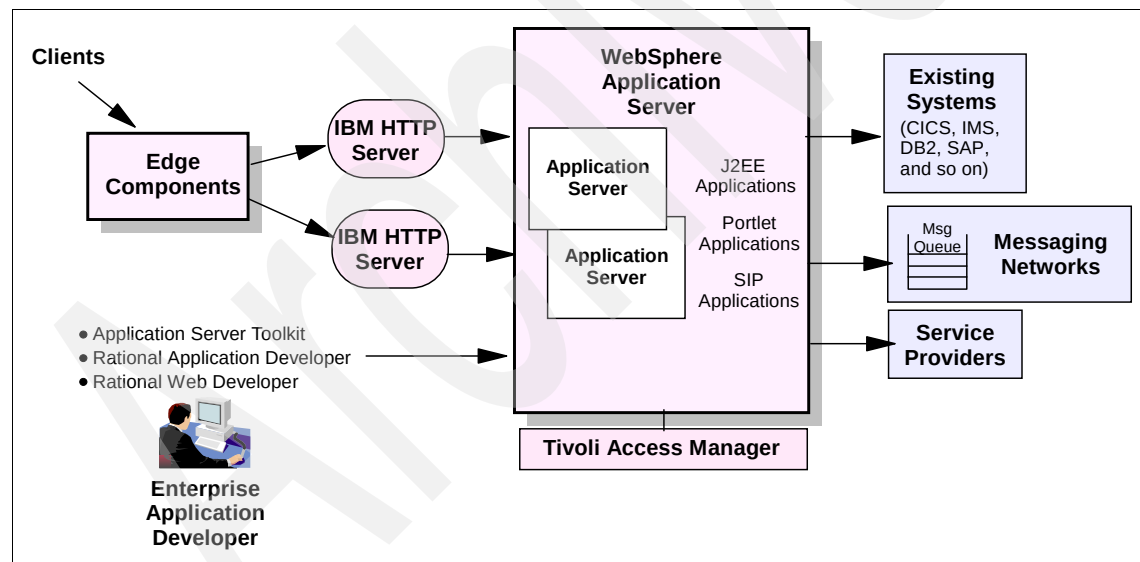


Figure 1-1 WebSphere Application Server product overview

The application server is the key component of WebSphere Application Server, providing the run time environment for applications that conform to the J2EE 1.2, 1.3, and 1.4 specifications. Clients access these applications through standard interfaces and APIs. The applications, in turn, have access to a wide variety of external sources, such as existing systems, databases, Web services, and

messaging resources that can be used to process the client requests. V6.1 extends the application server to allow it to run JSR 168 compliant portlets and Session Initiation Protocol (SIP) applications written to the JSR 116 specification.

With the Base and Express packages, you are limited to single application server environments. The Network Deployment package allows you to extend this environment to include multiple application servers that are administered from a single point of control and can be clustered to provide scalability and high availability environments.

WebSphere Application Server supports asynchronous messaging through the use of a JMS provider and its related messaging system. WebSphere Application Server includes a fully integrated JMS 1.1 provider called the default messaging provider. This messaging provider complements and extends WebSphere MQ and application server. It is suitable for messaging among application servers and for providing messaging capability between WebSphere Application Server and an existing WebSphere MQ backbone.

WebSphere Application Server provides authentication and authorization capabilities to secure administrative functions and applications. Your choice of user registries include the operating system user registry, an LDAP registry (for example, Tivoli® Directory Server), custom registries, file-based registries, or federated repositories. In addition to the default authentication and authorization capabilities, you have the option of using an external Java Authorization Contract for Containers (JACC) compliant authorization provider for application security. The IBM Tivoli Access Manager client embedded in WebSphere Application Server is JACC-compliant and can be used to secure your WebSphere Application Server-managed resources. This client technology is designed to be used with the Tivoli Access Manager Server (shipped with Network Deployment).

WebSphere Application Server works with a Web server (such as the IBM HTTP Server) to route requests from browsers to the applications that run in WebSphere Application Server. Web server plug-ins are provided for installation with supported Web browsers. The plug-ins direct requests to the appropriate application server and perform workload balancing among servers in a cluster.

WebSphere Application Server Network Deployment includes the Caching Proxy and Load Balancer components of Edge Component for use in highly available, high volume environments. Using these components can reduce Web server congestion, increase content availability, and improve Web server performance.

1.3 Packaging

Because varying e-business application scenarios require different levels of application server capabilities, WebSphere Application Server is available in multiple packaging options. Although they share a common foundation, each provides unique benefits to meet the needs of applications and the infrastructure that supports them. At least one WebSphere Application Server product fulfills the requirements of any particular project and its supporting infrastructure. As your business grows, the WebSphere Application Server family provides a migration path to more complex configurations.

WebSphere Application Server - Express V6.0

The Express package is geared to those who need to get started quickly with e-business. It is specifically targeted at medium-sized businesses or departments of a large corporation, and is focused on providing ease of use and ease of application development. It contains full J2EE 1.4 support but is limited to a single-server environment.

WebSphere Application Server - Express is unique from the other packages in that it is bundled with an application development tool. Although there are WebSphere Studio and Rational Developer products designed to support each WebSphere Application Server package, normally they are ordered independent of the server. WebSphere Application Server - Express includes the Rational Web Developer application development tool. It provides a development environment geared toward Web developers and includes support for most J2EE 1.4 features with the exception of Enterprise JavaBeans™ (EJB™) and J2EE Connector Architecture (JCA) development environments. However, keep in mind that WebSphere Application Server - Express V6 does contain full support for EJB and JCA, so you can deploy applications that use these technologies.

WebSphere Application Server V6.1

The WebSphere Application Server package is the next level of server infrastructure in the WebSphere Application Server family. Though the WebSphere Application Server is functionally equivalent to that shipped with Express, this package differs slightly in packaging and licensing.

This package includes two tools for application development and assembly:

- The Application Server Toolkit, which has been expanded in V6.1 to include a full set of development tools. The toolkit is suitable for J2EE 1.4 application development as well as the assembly and deployment of J2EE applications. It also supports Java 5 development.

In addition, the toolkit provides tools for the development, assembly, and deployment of JSR 116 SIP and JSR 168 portlet applications.

- ▶ This package also includes a trial version of Rational Application Developer, which supports the development, assembly, and deployment of J2EE 1.4 applications.

To avoid confusion with the Express package in this IBM Redbook, we refer to this as the Base package.

WebSphere Application Server Network Deployment V6

WebSphere Application Server Network Deployment is an even higher level of server infrastructure in the WebSphere Application Server family. It extends the WebSphere Application Server base package to include clustering capabilities, Edge components, and high availability for distributed configurations. These features become more important at larger enterprises, where applications tend to service a larger client base, and more elaborate performance and availability requirements are in place.

Application servers in a cluster can reside on the same or multiple machines. A Web server plug-in installed in the Web server can distribute work among clustered application servers. In turn, Web containers running servlets and Java ServerPages (JSPs) can distribute requests for EJBs among EJB containers in a cluster.

The addition of Edge components provides high performance and high availability features. For example:

- ▶ The Caching Proxy intercepts data requests from a client, retrieves the requested information from the application servers, and delivers that content back to the client. It stores cachable content in a local cache before delivering it to the client. Subsequent requests for the same content are served from the local cache, which is much faster and reduces the network and application server load.
- ▶ The Load Balancer provides horizontal scalability by dispatching HTTP requests among several, identically configured Web server or application server nodes.

WebSphere Application Server V6.1 for z/OS

IBM WebSphere Application Server for z/OS is a full-function version of the Network Deployment product. WebSphere Application Server for z/OS can support e-business on any scale.

Packaging summary

Table 1-1 shows the features included with each WebSphere Application Server packaging option.

Table 1-1 WebSphere Application Server packaging

Features included	Express V6.0 ¹	Base V6.1	Network Deployment V6.1	V6.1 for z/OS
WebSphere Application Server	Yes	Yes	Yes	Yes
Deployment manager	No	No	Yes	Yes
Web server plug-ins	Yes	Yes	Yes	Yes
IBM HTTP Server	Yes	Yes	Yes	Yes
Application Client (not available on Linux® for zSeries®)	Yes	Yes	Yes	Yes
Application Server Toolkit	Yes	Yes	Yes	Yes
DataDirect Technologies JDBC™ Drivers for WebSphere Application Server	Yes	Yes	Yes	Yes (for Windows® only)
Rational Development tools	Rational Web Developer (single use license)	Rational Application Developer Trial	Rational Application Developer Trial	Rational Application Developer Trial (non-z/OS platforms)
Database	IBM DB2 Universal Database™ Express V8.2	IBM DB2 Universal Database Express V8.2 (development use only)	IBM DB2 UDB Enterprise Server Edition V8.2 for WebSphere Application Server Network Deployment	No
Production ready applications	IBM Business Solutions	No	No	No
Tivoli Directory Server for WebSphere Application Server (LDAP server)	No	No	Yes	No

Features included	Express V6.0 ¹	Base V6.1	Network Deployment V6.1	V6.1 for z/OS
Tivoli Access Manager Servers for WebSphere Application Server	No	No	Yes	Yes (non-z/OS platforms)
Edge Components	No	No	Yes	Yes (non-z/OS platforms)
1. Express is limited to a maximum of two CPUs.				

Note: Not all features are available on all platforms. See the System Requirements Web page for each WebSphere Application Server package for more information.

1.4 Supported platforms and software

The following tables illustrate the platforms, software, and versions that WebSphere Application Server V6 supports at the time of the writing of this document. For the most up-to-date operating system levels and requirements, refer to the WebSphere Application Server system requirements at:

<http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html>

1.4.1 Operating systems

Table 1-2 shows the supported operating systems and versions for WebSphere Application Server V6.1.

Table 1-2 Supported operating systems and versions

Operating Systems	Versions
Windows	▶ Windows 2000 Advanced Server with SP4 ▶ Windows 2000 Server with SP4 ▶ Windows 2000 Professional Server with SP4 ▶ Microsoft® Windows Server® 2003 (Datacenter with SP1) ▶ Microsoft Windows Server 2003 (Enterprise with SP1) ▶ Microsoft Windows Server 2003 (Standard with SP1) ▶ Microsoft Windows XP Professional with SP2 ▶ Microsoft Windows Server 2003 x64 Editions
AIX®	▶ AIX 5L™ V5.2 Maintenance Level 5200-07 ▶ AIX 5L V5.3 with Service Pack 5300-04-01

Operating Systems	Versions
Sun™ Solaris™	▶ Solaris 9 with the latest patch Cluster ▶ Solaris 10 with the latest patch Cluster
HP-UX	▶ HP-UX 11iv2 (11.23) with the latest Quality Pack
Linux (Intel®)	▶ Red Hat Linux Enterprise AS, ES, WS V3 with Update 5 or 6 ▶ Red Hat Linux Enterprise AS, ES, WS V4 with Update 2 ▶ SUSE Linux Enterprise Server, V9 with SP2 or 3
Linux (Power PC®)	▶ Red Hat Enterprise Linux AS V3 with Update 5 or 6 ▶ Red Hat Enterprise Linux AS V4 with Update 2 ▶ SUSE Linux Enterprise Server V9 with SP2 or 3
Linux on IBM System z™ (Supported for WebSphere Application Server Network Deployment only)	▶ Red Hat Enterprise Linux AS V3 with Update 5 or 6 ▶ Red Hat Enterprise Linux AS V4 with Update 2 ▶ SUSE Linux Enterprise Server V9 with SP2 or 3
i5/OS® and OS/400®	▶ i5/OS and OS/400, V5R3 ▶ i5/OS V5R4
z/OS (Supported for WebSphere Application Server Network Deployment only)	▶ z/OS 1.6 or later ▶ z/OS.e 1.6 or later

1.4.2 Web servers

The following Web servers are supported by WebSphere Application Server V6.1 on all available platforms:

- ▶ Apache HTTP Server 2.0.54
- ▶ IBM HTTP Server for WebSphere Application Server V6.0.2
- ▶ IBM HTTP Server for WebSphere Application Server V6.1
- ▶ Internet Information Services 5.0
- ▶ Internet Information Services 6.0
- ▶ Lotus® Domino® Enterprise Server 6.5.4 or 7.0
- ▶ Sun Java™ System Web Server 6.0 SP9
- ▶ Sun Java System Web Server 6.1 SP3

1.4.3 Database servers

Table 1-3 shows the database servers that WebSphere Application Server V6.1 supports.

Table 1-3 Supported database servers and versions

Databases	Versions
IBM DB2	DB2 for iSeries 5.2, 5.3, or 5.4 DB2 for z/OS V7 or V8 DB2 Enterprise Server Edition 8.2 FP4 DB2 Express 8.2 FP4 DB2 Workgroup Server Edition 8.2 FP4
Cloudscape™	Cloudscape 10.1
Oracle	Oracle 9i Standard/Enterprise Release 2 - 9.2.0.7 Oracle 10g Standard/Enterprise Release 1 - 10.1.0.4 Oracle 10g Standard/Enterprise Release 2 - 10.2.0.1 or 10.2.0.2
Sybase	Sybase Adaptive Server Enterprise 12.5.2 or 15.0
Microsoft SQL Server	Microsoft SQL Server Enterprise 2000 SP4 Microsoft SQL Server Enterprise 2005
Informix®	Informix Dynamic Server 9.4C7W1 or 10.00C4
IMS™	IMS V8 or V9
WebSphere Information Integrator	WebSphere Information Integrator 8.2 FP4

1.4.4 Directory servers

Table 1-4 shows the LDAP servers that WebSphere Application Server V6.1 supports.

Table 1-4 Supported Directory servers and versions

Directory Server	Versions
IBM Tivoli Directory Server	5.2 and 6.0
z/OS Security Server	1.6 and 1.7
z/OS.e Security Server	1.6 and 1.7
Lotus Domino Enterprise Server	6.5.4 and 7.0
Sun ONE™ Directory Server	5.1 SP4 and 5.2

Directory Server	Versions
Windows Active Directory®	2003 and 2000
Novell eDirectory	8.7.3 and 8.8

Archived

System management: A technical overview

This chapter describes in detail the system management functionality of WebSphere Application Server. This information will help you understand how system administration occurs. It is particularly useful in a multi-server environment to understand the distributed administration and synchronization topics.

This chapter includes the following topics:

- ▶ System management overview
- ▶ Java Management Extensions (JMX)
- ▶ Distributed administration
- ▶ Configuration and application data repository

2.1 System management overview

At first glance, system management concepts in WebSphere Application Server might seem complex. However, the fact that the system management architecture is based on JMX™, and the fact that WebSphere Application Server provides easy-to-use administration tools makes it fairly simple to use and understand.

Terminology: There are differences in how WebSphere Application Server handles administration depending on the environment you have set up. You will see us refer to the following when explaining these differences:

- ▶ *Stand-alone server environment* refers to a single stand-alone server that is not managed as part of a cell. With the Base and Express packages, this is your only option. You can also create a stand-alone server with the Network Deployment package.
- ▶ *Distributed server environment* refers to the situation where you have multiple servers managed from a single deployment manager in the cell. We also refer to these as *managed servers*. This is only valid with the Network Deployment package.
- ▶ *Managed processes* refer to the deployment manager, nodes (node agents), and application servers.

2.1.1 System management in a stand-alone server environment

Each managed process has an administrative service that interacts with administration clients. In a stand-alone server environment, both the administrative console application and the administrative service runs on the application server. The configuration repository consists of one set of configuration files managed by the administrative service. System management is simplified in the sense that the changes made by the administrator are applied directly to the configuration files used by the server.

Figure 2-1 on page 17 shows the management of a single-server installation.

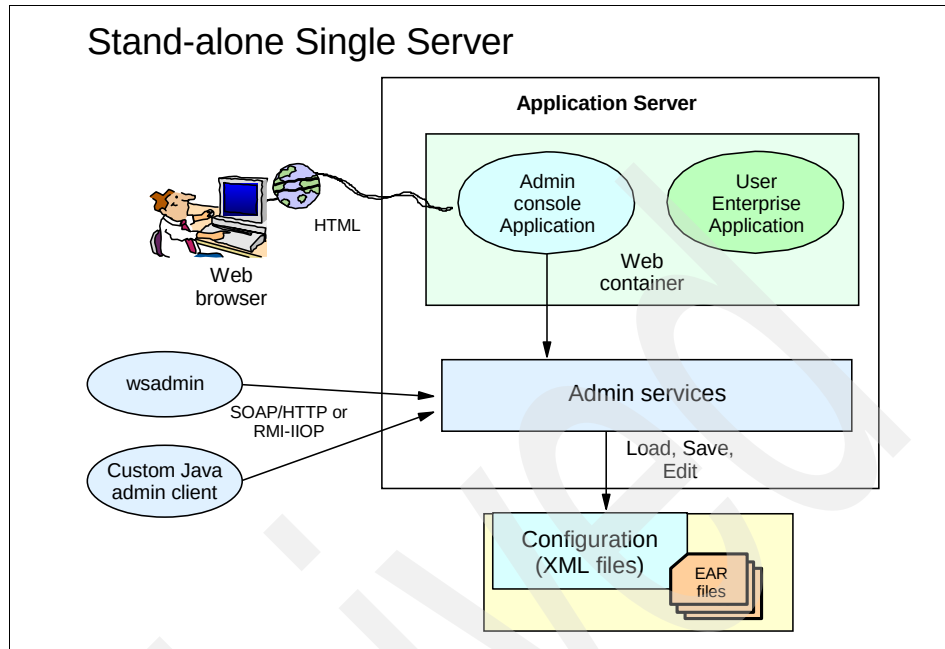


Figure 2-1 Managing a single-server installation

The administrative console will contain a subset of options that you see in the administrative console for a distributed server environment. The options you will not see are related to the workload management and high availability features.

2.1.2 System management in a distributed server environment

In a distributed server environment, administration tasks and configuration files are distributed among the nodes, reducing the reliance on a central repository or administration server for basic functions and bring-up. The administrative services and the administrative console are hosted on the deployment manager.

Managed application servers are installed on nodes. Each node has a node agent that interacts with the deployment manager to maintain and manage the processes on that node.

Multiple sets of the configuration files exist. The master configuration is maintained on the deployment manager node and pushed out, synchronized, to the nodes. Each managed process starts with its own configuration file.

Figure 2-2 shows manager a multi-server installation.

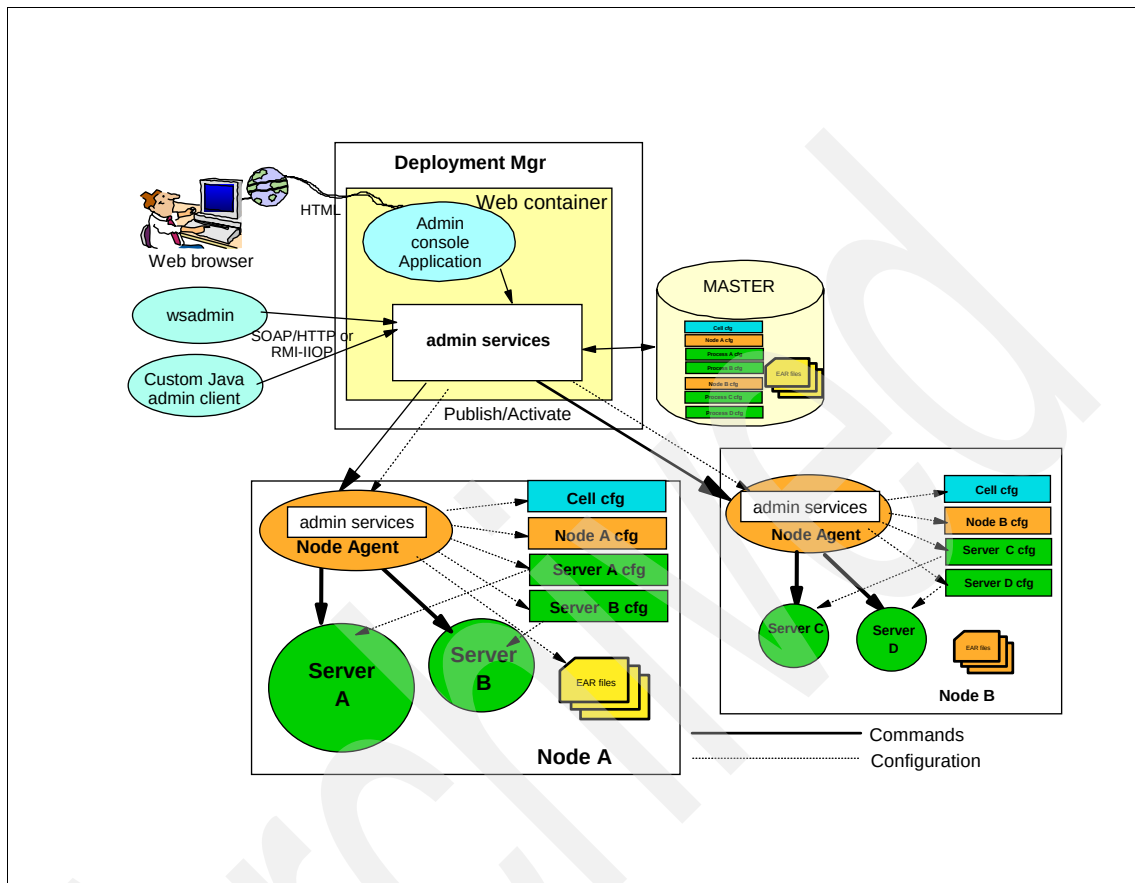


Figure 2-2 Managing a multi-server installation

Configuration should always be done at the deployment manager and synchronized out to the nodes. Although it is theoretically possible to configure nodes locally using **wsadmin**, it is not recommended and any changes made will be overwritten at the next synchronization.

However, operational commands can be directed at the deployment manager, node agent, or server.

2.2 Java Management Extensions (JMX)

The system management functionality of WebSphere Application Server is based on the use of Java Management Extensions (JMX). JMX is a framework that provides a standard way of exposing Java resources, for example application servers, to a system management infrastructure. The JMX framework allows a provider to implement functions, such as listing the configuration settings, and allows users to edit the settings. It also includes a notification layer that can be used by management applications to monitor events, such as the startup of an application server.

The use of JMX opens the door to third-party management tool providers. Users of WebSphere are no longer restricted to IBM-supplied management tools.

JMX is a Java specification (JSR-003) that is part of J2SE 1.5. A separate specification defines the J2EE management API (JSR-77) for managing a J2EE conforming application server. The J2EE 1.4 specification requires that all J2EE products support the Enterprise Edition management API. WebSphere Application Server provides *managed objects (MOs)* as defined in the JSR-77 specification and hence is manageable from third-party management products that delivers J2EE management capabilities.

IBM WebSphere Application Server V6.x implements JMX 1.2, while Version 5.x implements JMX 1.1. Due to the evolution of the JMX specification, the serialization format for JMX objects, such as `javax.management.ObjectName`, differs between the two specifications.

The WebSphere Application Server V6.1 JMX run time has been enhanced to be aware of the version of the client with which it is communicating. It makes appropriate transformations on these incompatible serialized formats so as to allow the different version run times to communicate with each other. This makes it possible for a V5.x administrative client to call a V6.1 deployment manager, node, or server. Similarly, a V6.1 administrative client can call a V5.x node or server.

2.2.1 JMX architecture

The JMX architecture is structured into three layers:

- Instrumentation layer

The instrumentation layer dictates how resources can be wrapped within special Java beans called Management Beans (MBeans).

► Agent layer

The agent layer consists of the MBean server and agents, which provide a management infrastructure. Services implemented include:

- Monitoring
- Event notification
- Timers

► Management layer

The management layer defines how external management applications can interact with the underlying layers in terms of protocols, APIs, and so on.

The layered architecture of JMX is summarized in Figure 2-3.

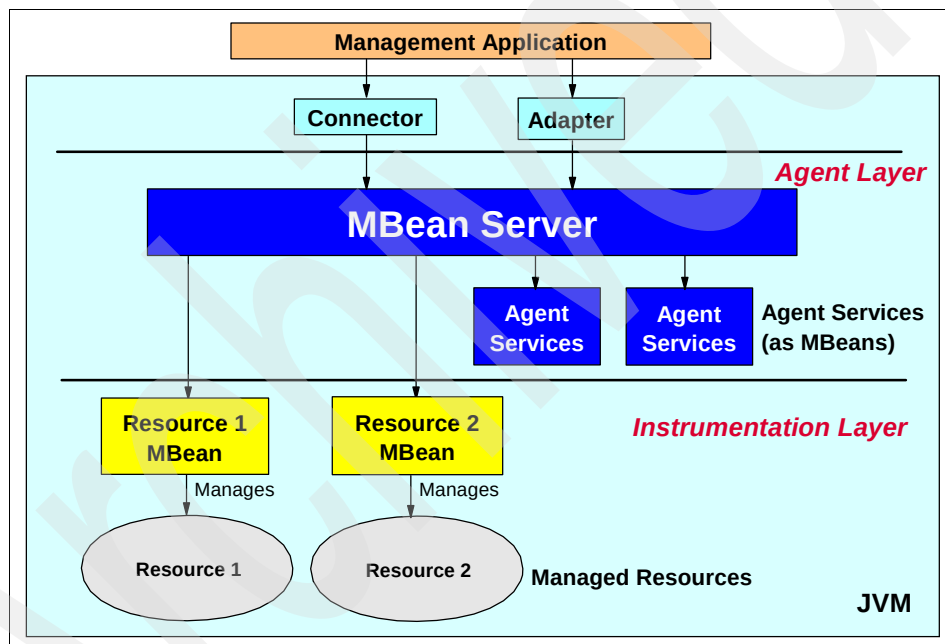


Figure 2-3 JMX architecture

How does JMX work?

Resources are managed by JMX MBeans. These are not EJBs, but simple Java beans that need to conform to certain design patterns outlined in the JMX specification.

Providers that want to instrument their systems with JMX need to provide a series of MBeans. Each MBean is meant to wrap, or represent, a certain run time resource. For example, in order to expose an application server as a manageable resource, WebSphere needs to provide an application server MBean.

External applications can interact with the MBeans through the use of JMX connectors and protocol adapters. Connectors are used to connect an agent with a remote JMX-enabled management application. This form of communication involves a connector in the JMX agent and a connector client in the management application.

The key features of JMX connectors are:

- ▶ Connectors are oriented to the transport mechanism. For example, a provider can provide an RMI connector that allows Java applications to interact remotely with the MBeans.
- ▶ The connector translates Java beans calls to a protocol stream.
- ▶ There is a 1:1 mapping between client method invocations and MBean operations.
- ▶ This is the low-level API for accessing MBeans.

Protocol adapters

Protocol adapters provide a management view of the JMX agent through a given protocol. Management applications that connect to a protocol adapter are usually specific to the given protocol.

The key features of JMX protocol adapters are:

- ▶ Protocol adapters adapt operations of MBeans and the MBean server into a representation in the given protocol, and possibly into a different information model, for example, SNMP or HTTP.
- ▶ There is not a 1:1 mapping between client method invocations and MBean operations.
- ▶ This is the high-level API for accessing MBeans.

MBean server

Each JMX enabled JVM™ contains an MBean server that registers all the MBeans in the system. It is the MBean server that provides access to all of its registered MBeans. There is only one MBean server per JVM.

Both connectors and protocol adapters use the services of the MBean server in order to apply the management operation they receive to the MBeans, and in order to forward notifications to the management system. Connector and protocol adapter communication is summarized in Figure 2-4.

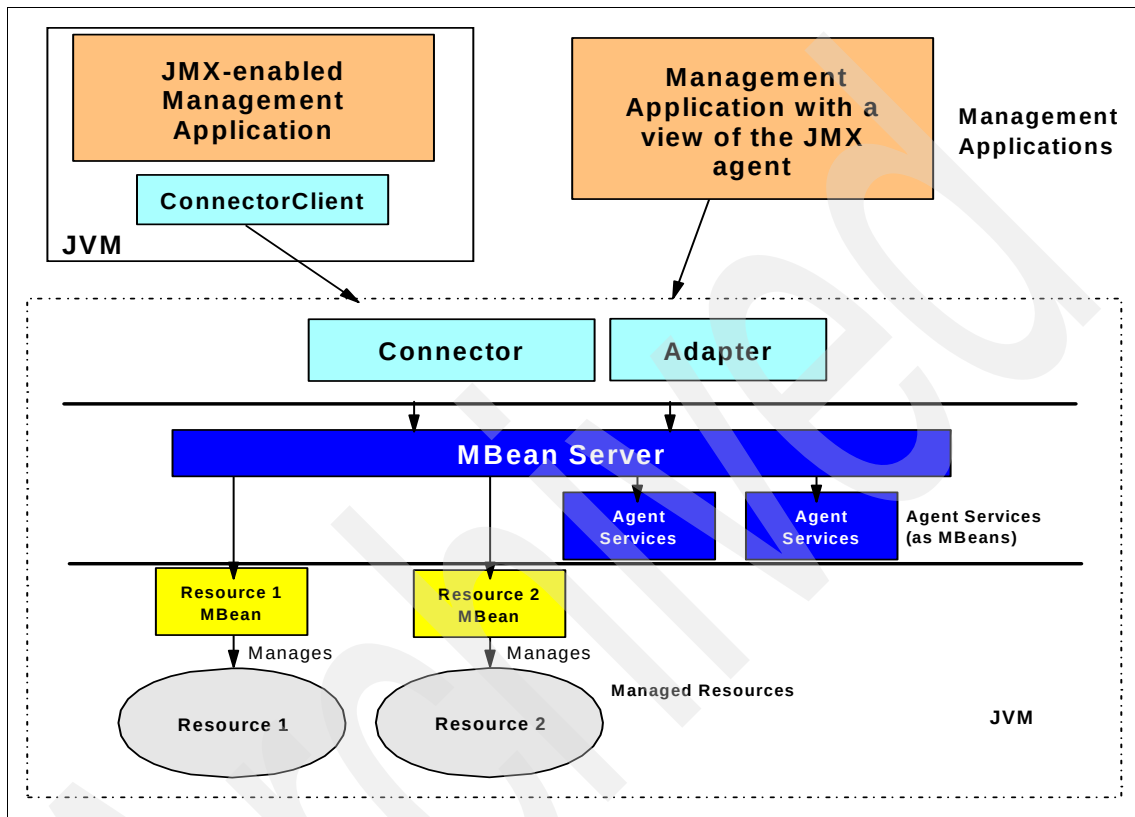


Figure 2-4 JMX connectors and adapters

2.2.2 JMX distributed administration

Figure 2-5 on page 23 shows how the JMX architecture fits into the overall distributed administration topology of a distributed server environment.

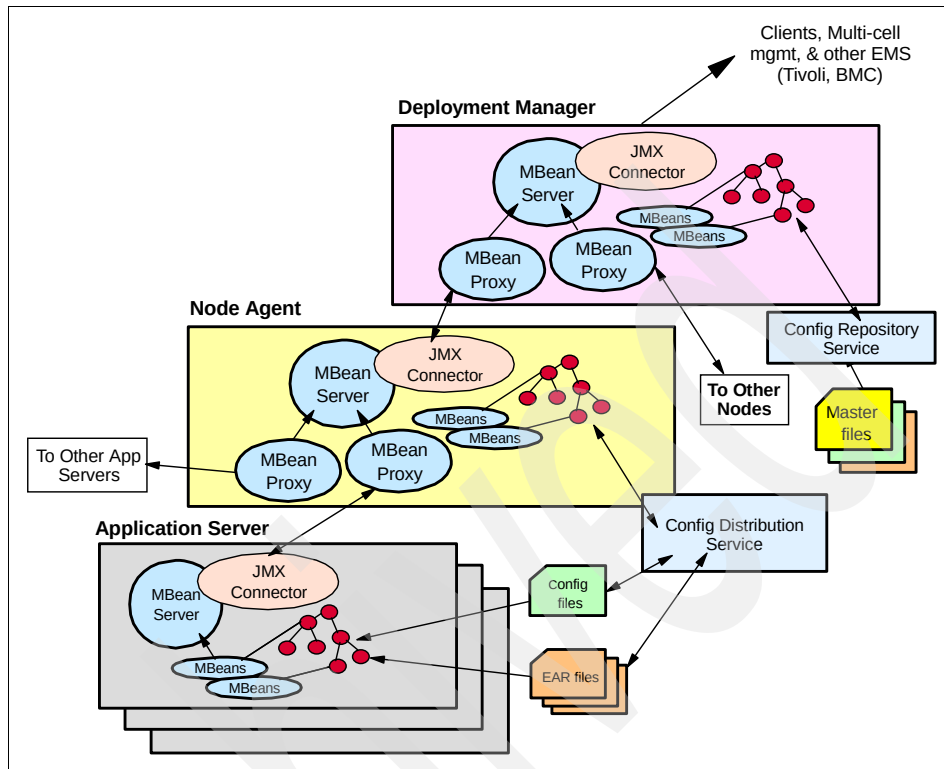


Figure 2-5 JMX distributed administration

The key points of this distributed administration architecture are:

- ▶ Internal MBeans local to the JVM register with the local MBean server.
- ▶ External MBeans have a local proxy to their MBean server. The proxy registers with the local MBean server. The MBean proxy allows the local MBean server to pass the message to an external MBean server located on:
 - Another server
 - Node agent
 - Deployment manager
- ▶ A node agent has an MBean proxy for all servers within its node. However, MBean proxies for other nodes are not used.
- ▶ The deployment manager has MBean proxies for all node agents in the cell.

The configuration of MBean proxies is shown in Figure 2-6 on page 24.

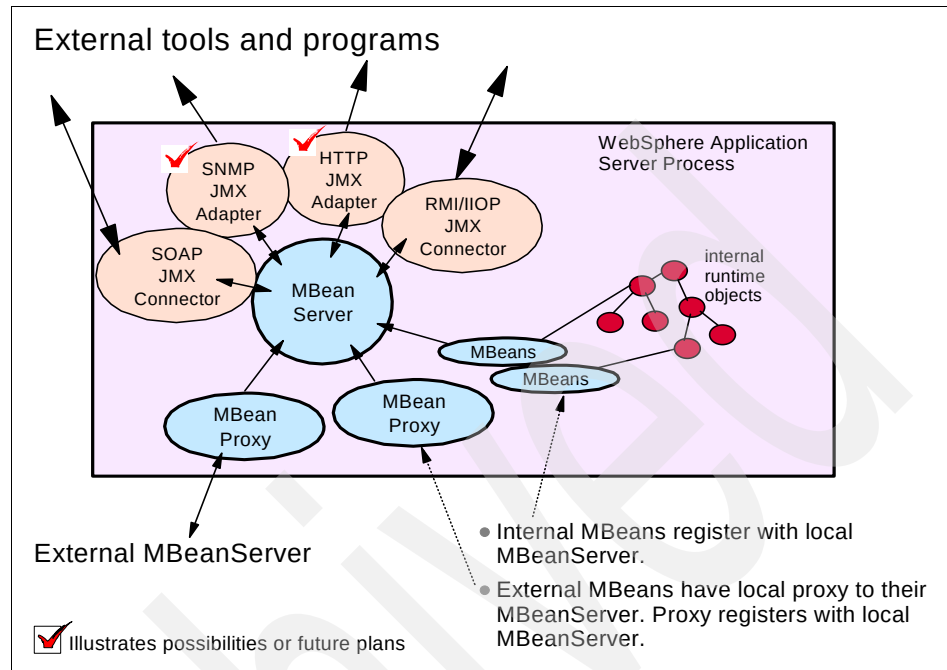


Figure 2-6 JMX architecture

2.2.3 JMX MBeans

WebSphere Application Server provides a number of MBeans, each of which can have different functions and operations available. For example:

- ▶ An application server MBean might expose operations such as start and stop.
- ▶ An application MBean might expose operations such as install and uninstall.

2.2.4 JMX usage scenarios

Some of the more common JMX usage scenarios you will encounter are:

- ▶ Internal product usage:

All WebSphere Application Server administration clients use JMX:

- WebSphere administrative console
- **wsadmin** scripting client
- Admin client Java API

► External programmatic administration

In general, most external users will not be exposed to the use of JMX. Instead, they will access administration functions through the standard WebSphere Application Server administration clients.

However, external users would need to access JMX in the following scenarios:

- External programs written to control the WebSphere Application Server run time and its resources by programmatically accessing the JMX API.
- Third-party applications that include custom JMX MBeans as part of their deployed code, allowing the applications components and resources to be managed through the JMX API.

2.2.5 J2EE management

The J2EE management specification dictates the existence of certain Managed Objects (MOs) that can be used to manage the available application server resources. The specification does not require that managed objects be implemented by means of JMX MBeans, but the required interface makes MBeans a natural choice for MOs.

In WebSphere Application Server, the management standard MOs are essentially provided by mappings to existing WebSphere JMX MBeans. For example, the specification requires a J2EEServer managed object that is equivalent to the Server MBean in WebSphere. The management standard introduces a set of required key properties, part of a new `ObjectName` method, a number of attributes, and three optional interfaces: `EventProvider`, `StateManageable`, and `StatisticsProvider`. These required and optional parts have all been added to the relevant WebSphere MBeans (see the Information Center section *Administrative programs for multiple Java 2 Platform, Enterprise Edition application servers* for a detailed description of the available objects and attributes).

A major requirement by the standard that does not easily map into the existing WebSphere architecture is the ability to interoperate with management objects representing resources that have not been started in the WebSphere run time environment. Consequently, a proxy mechanism has been introduced that runs in every application server in a stand-alone server environment, or as part of the deployment manager in a distributed server environment. With this proxy implementation, all the required managed objects, methods, and attributes can be interfaced regardless of whether the WebSphere JMX MBean is running or not.

Be aware that the J2EE management standard defines a common set of objects and operations for J2EE application servers and hence does not provide management capabilities for specific WebSphere Application Server features.

We recommend that WebSphere-only management clients operate directly on the WebSphere JMX MBeans to avoid the overhead of the proxy object and to take advantage of the full management capabilities of the WebSphere product.

2.3 Distributed administration

Administration in a distributed server environment is by necessity more complex than administration in a stand-alone server environment. In a distributed server environment, multiple WebSphere Application Server nodes are managed from a single central location. This distributed administration of components is brought about by three *tiers*, or layers, of administration services, as shown in Figure 2-7.

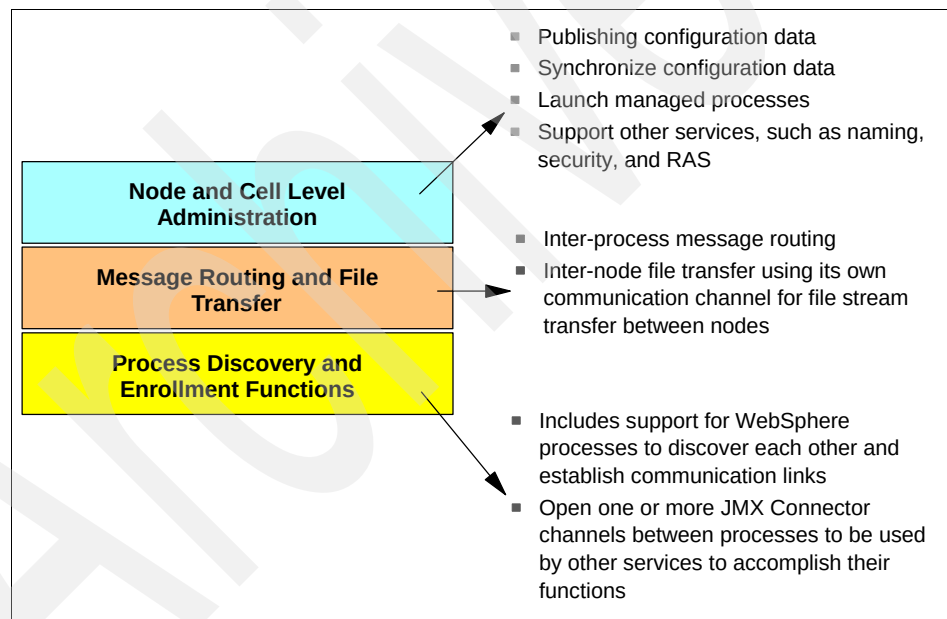


Figure 2-7 Layers of distributed administration services

Between these tiers, communication is used to distribute configuration and application data updates from the deployment manager to the node agent, and in turn to the server instances.

The routing of administration messages between components makes use of the JMX ObjectName that identifies the target managed resource within the

administrative cell. The ObjectName contains all of the information necessary to route a request targeted at the resource, to the appropriate node where the resource is executing.

An example is shown in Figure 2-8, where an operation on Node Y invokes a management method on a management bean (MBean) located on another node, Node X.

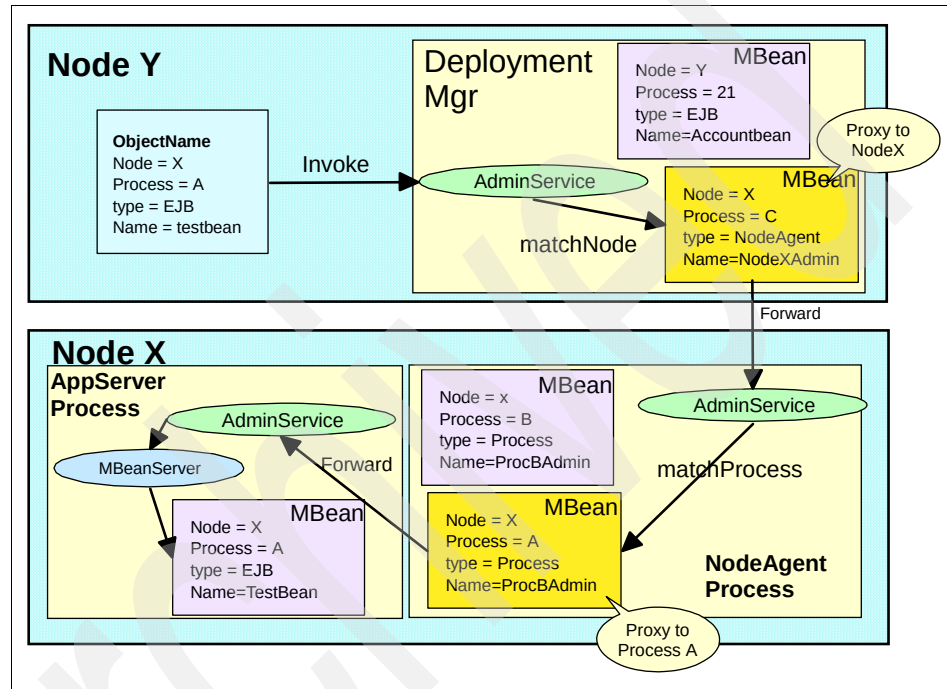


Figure 2-8 Distributed administration message routing

Where:

1. An object running on server A of Node Y sends an operation request to the deployment manager AdminService located on the same machine.
2. The deployment manager AdminService determines which node hosts the requested service (Node X) and passes the request to the MBean acting as the proxy of the node's node agent.
3. The proxy MBean forwards the request to the AdminService of the Node X node agent.
4. On Node X, the node agent AdminService receives the request and determines which managed server (process) the requested service is hosted on (process A).

5. The AdminService passes the request to the MBean acting as the proxy of the managed server.
6. The proxy MBean forwards the request to the AdminService of the managed server.
7. The managed server AdminService invokes the requested service via the local MBeanServer, which is responsible for all direct communication with MBeans hosted in that JVM.

2.3.1 Distributed process discovery

When a managed server begins its startup, it sends a discovery request message that allows other processes to discover its existence and establish communication channels with the process.

Figure 2-9 shows an example of the distributed discovery process for a topology containing two nodes that are located on different machines. Note that both node agents in the figure use ports 7272 and 5000. This assumes they reside on separate physical machines. If nodes are located on the same machine, they must be configured to use non-conflicting IP ports.

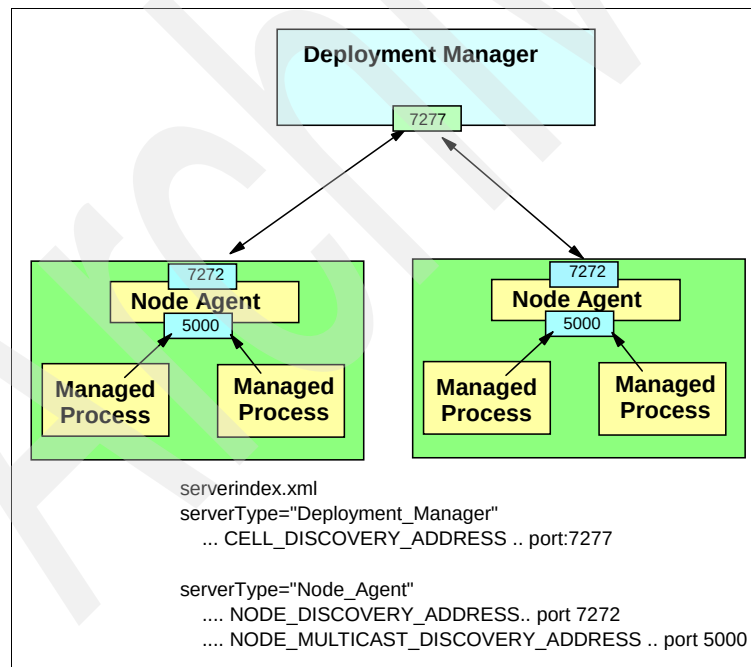


Figure 2-9 Distributed discovery process

Each node agent and deployment manager maintains status and configuration information by using discovery addresses, or *ports*. On startup, processes discover other running components, and create communication channels between them, through the discovery addresses:

- The master repository located on the deployment manager installation contains the `serverindex.xml` file for each node. The deployment manager reads this file on startup to determine the host name and IP port of each node agent's `NODE_DISCOVERY_ADDRESS`.

The default port for the `NODE_DISCOVERY_ADDRESS` is 7272. You can verify this by looking at the `NODE_AGENT` stanza in the `serverindex.xml` file of each node located at:

```
<dmgr_profile_home>/config/cells/<cell>/nodes/<node>/serverindex.xml
```

You can also display this port from the administrative console by selecting **System Administration** → **Node agents**. Select each node agent and expand **Ports** under the Additional Properties section.

- The copy of the configuration repository located on each node contains the `serverindex.xml` file for the deployment manager. The node agent reads this file on startup to determine the host name and IP port of the deployment manager's `CELL_DISCOVERY_ADDRESS`.

The default port for the `CELL_DISCOVERY_ADDRESS` is port 7277. You can verify this by looking at the `DEPLOYMENT_MANAGER` stanza in the `serverindex.xml` file for the deployment manager node located at:

```
<profile_home>/config/cells/<cell>/nodes/<DM_node>/serverindex.xml
```

You can also display this port from the administrative console by selecting **System Administration** → **Deployment manager**. Expand **Ports** under the Additional Properties section.

- The copy of the configuration repository located on each node also contains the `serverindex.xml` file for the node. Each managed server reads this file on startup to determine the host name and IP port of the node agent's `NODE_MULTICAST_DISCOVERY_ADDRESS`.

A multicast address is used to prevent the usage of a large number of IP ports for managed server to node agent discovery requests. Using multicast, a node agent can listen on a single IP port for any number of local servers.

The default port for the `NODE_MULTICAST_DISCOVERY_ADDRESS` is 5000. You can verify this by looking at the `NODE_AGENT` stanza in the `serverindex.xml` file of the node located at:

```
<profile_home>/config/cells/<cell>/nodes/<node>/serverindex.xml
```

You can also display this port from the administrative console by selecting **System Administration** → **Node agents**. Select the node agent and expand **Ports** under the Additional Properties section.

Important: Keep the following in mind:

- ▶ The discovery service uses the `InetAddress.getLocalHost()` call to retrieve the IP address for the local machine's host name. The network configuration of each machine must be configured so that `getLocalHost()` does not return the loopback address (127.0.0.1). It must return the real IP address of the correctly chosen NIC.
- ▶ A multicast address is a logical address. Therefore, it is not bound to a real, physical network interface, and will not be the same as the host name (or IP address) of the host on which the node agent is executed.
- ▶ Multicast host addresses must be within a special range (224.0.0.0 to 239.255.255.255) defined by the IP standards and must never be a host name value. The default for WebSphere node agents is 232.133.104.73.

Each server has its own copy of the configuration and application data necessary for startup of the run time and the installed applications.

Rules for process startup

The order of process startup needs to adhere to the following rules:

- ▶ A node agent can be running while the deployment manager is not, and vice versa. When the stopped process is started, discovery will occur automatically.
- ▶ The deployment manager can be running while a managed server is not, and vice versa. The execution of a managed server is not dependent on the presence of a running deployment manager. The deployment manager is only required for permanent configuration changes written to the master repository.
- ▶ The node agent should be started before any application servers on that node. The node agent contains the Location Service Daemon (LSD) in which each application server registers on startup.
- ▶ The node agent is purely an administrative agent and is not involved in application serving functions. Each managed server has the data necessary to start itself.

Example discovery scenarios

Situation: The node agent is not running and the deployment manager starts:

1. The deployment manager tries to determine if the node agent is running. The process fails.
2. When the node agent is started, it contacts the deployment manager, creates a communication channel, and synchronizes data.

Situation: The node agent starts but no managed servers are started:

1. The node agent knows all about its managed servers and checks whether they are started. If so, it creates communication channels to these processes.
2. When a managed server starts, it checks whether the node agent is started and then creates a communication channel to it.

2.3.2 Centralized changes to configuration and application data

In a distributed server environment, you have a master repository of configuration and application data for the cell. Administrative clients are used to provide centralized functionality for:

- ▶ Modification of configuration settings in the master repository.
- ▶ Installation, update, and uninstallation of applications on application server(s) in the cell. In the process, the Enterprise Application Archive (EAR) files and deployment descriptors are also stored in the master repository.

Each node contains a separate copy of the repository containing only the files required for that node, including:

- ▶ Cell and node-level configuration files necessary for node and managed server operation, for example, the `serverindex.xml` file for each node in the cell.
- ▶ Application server configuration files for the application servers on that node.
- ▶ EAR files for the applications hosted by servers on that node.
- ▶ Application deployment descriptors for the applications hosted by servers on that node. These deployment descriptors contain the settings specified when the application was deployed.

When an administrator makes changes to the configuration using an administration tool and saves these changes to the master repository, they are available for use. The next step is to synchronize the changes out to the nodes of the cell.

2.3.3 File synchronization

The file synchronization service is the administrative service responsible for keeping up to date the configuration and application data files that are distributed across the cell. The service runs in the deployment manager and node agents, and ensures that changes made to the master repository will be propagated out to the nodes, as necessary. The file transfer system application is used for the synchronization process. File synchronization can be forced from an administration client, or can be scheduled to happen automatically.

During the synchronization operation, the node agent checks with the deployment manager to see if any files that apply to the node have been updated in the master repository. New or updated files are sent to the node, while any deleted files are also deleted from the node.

Synchronization is one-way. The changes are sent from the deployment manager to the node agent. No changes are sent from the node agent back to the deployment manager.

How files are identified for synchronization

When synchronization occurs, WebSphere must be able to identify the files that have changed and therefore need to be synchronized. To do this, WebSphere uses the following scheme:

- ▶ A calculated digest is kept by both the node agent and the deployment manager for each file in the configuration they manage. These digest values are stored in memory. If the digest for a file is recalculated and it does not match the digest stored in memory, this indicates the file has changed.
- ▶ An *epoch* for each folder in the repository and one for the overall repository is also stored in memory. These epochs are used to determine whether any files in the directory have changed. When a configuration file is altered through one of the WebSphere administration interfaces, then the overall repository epoch and the epoch for the folder in which that file resides is modified.

Note that manually updating a configuration file does not cause the digest to change. Only files updated with administration clients will be marked as changed. Manually updating the files is not recommended, but if you do, a forced synchronization will include manually updated files.

- ▶ During configuration synchronization operations, if the repository epoch has changed since the previous synchronize operation, then individual folder epochs are compared. If the epochs for corresponding node and cell directories do not match, then the digests for all files in the directory are recalculated, including that changed file.

Synchronization scheduling

The scheduling of file synchronization is configured using an administrative client. The available options are:

- ▶ Automatic synchronization

Synchronization can be made to operate automatically by configuring the file synchronization service of the node agent. These settings allow you to:

- Enable periodic synchronization to occur at a specified time interval

By default, this option is enabled with a time interval of one minute.

- Enable synchronization at server startup

The synchronization will occur before the node agent starts a server. Note that if you start a server using the **startServer** command, this setting has no effect.

- ▶ Explicit/forced synchronization

Synchronization can be explicitly forced at anytime via use of an administrative client.

Tip: In a production environment, the automatic synchronization interval should be increased from the one minute default so that processing and network overhead is reduced.

Ensuring manual changes are synchronized

Important: Although it is technically possible to edit configuration files manually, it should not be done unless absolutely necessary. Manual editing has several drawbacks, including:

- ▶ When using `wsadmin` and the administrative console, you have the benefit of a validation process before the changes are applied. With manual editing, you have no such failsafe.
- ▶ Updates made manually are not marked for synchronization and will be lost at the next synchronization process unless you make them in the master repository and manually force synchronization.

Manual editing might be appropriate in problem determination scenarios. For example, if you enable WebSphere security, but have not set it up properly, you might not be able to start WebSphere and, thus, have no access to admin clients. In this instance, being able to turn off security manually so you can start WebSphere and review your configuration is very helpful.

The *Configuration Document Descriptions* topic in the Information Center lists several configuration files that have settings not exposed in the administration clients. In the event you find it necessary to edit a file manually, this topic will help make sure you do not lose your changes.

If a change to a configuration file is made by editing the file, then the digest for the file is not recalculated, because the epochs for the directories continue to match and the synchronization process will not recognize that the files have changed.

However, manual edits of configuration files in the master cell repository can be picked up if the repository is reset so that it re-reads all the files and recalculates all of the digests. You can reset either the master cell repository epoch or the node repository epoch.

- ▶ Resetting the master cell repository causes any manual changes made in the master configuration repository to be replicated to the nodes where the file is applicable.
- ▶ Resetting the node repository causes any manual changes to the local node files to be overwritten by whatever is in the master cell repository, regardless of whether the cell repository was changed or not. Any manual changes in the master repository will be picked up and brought down to the node.

The main difference between cell reset and node reset is that cell reset is likely to impact the entire cell, not just one node.

This holds true for changes to installed applications as well. They are treated the same as other configuration files in the repository. For each installed application, there is an EAR file in the repository and also some configuration files associated with the deployment of the application.

If you manually change the EAR file and reset the master cell repository, the changed EAR file will be replicated out to the nodes where it is configured to be served and will be expanded in the appropriate location on that node for the application server to find it. The application on that node will be stopped and restarted automatically so that whatever is changed is picked up and made available in the application server.

Important: Manually changing the EAR file is best performed by advanced users. Otherwise, unpredictable results can occur.

If you manually edit one of the deployment configuration files for the application and reset the repository, that change will be replicated to the applicable nodes and will be picked up the next time the application on that node is restarted.

Resetting the master cell repository

Note: The use of `wsadmin` is covered in Chapter 5, “Administration with scripting” on page 249. The only thing you might need to know about `wsadmin` to complete these tasks is to start `wsadmin` on the SOAP connector port of the process on which you want to run the commands. The default is to start to port 8879. If the process you are connecting to has a different port number specified, start `wsadmin` with the `-port` argument.

To perform a reset of the master cell repository, do the following:

1. Open a command prompt and change to the `<dmgr_profile_home>/bin` directory and start a `wsadmin` session. Note that the deployment manager must be running. Use the following command:

```
cd <install_root>\profiles\Dmgr01\bin
wsadmin
```

2. Enter the following:

```
wsadmin>set config [$AdminControl queryNames
*:*,type=ConfigRepository,process=dmgr]
```

```
wsadmin>$AdminControl invoke $config refreshRepositoryEpoch
```

You will see a number returned by the refreshRepositoryEpoch operation, for example, 1047961605195, as shown in Example 2-1.

Example 2-1 Resetting the master cell repository

```
<install_root>\profiles\Dmgr01\bin>wsadmin
WASX7209I: Connected to process "dmgr" on node DmgrNode using SOAP connector;
The type of process is: DeploymentManager
WASX7029I: For help, enter: "$Help help"

wsadmin>set config [$AdminControl queryNames
*:*,type=ConfigRepository,process=dmgr]

WebSphere:platform=common,cell=DmgrCell,version=6.1.0.0,name=repository,mbeanId
entifier=repository,type=ConfigRepository,node=DmgrNode,process=dmgr

wsadmin>$AdminControl invoke $config refreshRepositoryEpoch
1098317369266
wsadmin>
```

This resets the entire cell repository digest set. On the next synchronize operation, all files in the master cell repository will have their digests recalculated. Any manual changes will be replicated to the applicable nodes.

Resetting the node repository

There are multiple ways to reset a node repository for synchronization:

- In a **wsadmin** session connected to the deployment manager or node agent, enter the following:

```
wsadmin>set config [$AdminControl queryNames
*:*,type=ConfigRepository,process=nodeagent]
```

```
wsadmin>$AdminControl invoke $config refreshRepositoryEpoch
```

This resets the node digest set. Any file that does not match what is in the repository is overwritten.

Example 2-2 gives an overview of resetting the node repository.

Example 2-2 Resetting the node repository

```
<install_root>\profiles\<server_name>\bin>wsadmin -port 8883
```

```
WASX7209I: Connected to process "nodeagent" on node AppSrvrNode using  
SOAP connector; The type of process is: NodeAgent
```

```
WASX7029I: For help, enter: "$Help help"
```

```
wsadmin>set config [$AdminControl queryNames  
*:*,type=ConfigRepository,process=nodeagent]  
WebSphere:platform=common,cell=DmgrCell,version=6.1.0.0,name=repository  
,mbeanIdentifier=repository,type=ConfigRepository,node=AppSrvrNode,proc  
ess=nodeagent
```

```
wsadmin>$AdminControl invoke $config refreshRepositoryEpoch  
1098397549240
```

- ▶ From the deployment manager administrative console, select **System Administration** → **Nodes** to see a list of the nodes in the cell. Notice the Synchronize and Full Resynchronize buttons on the page. The Synchronize button causes a normal synchronize operation with no re-reading of the files. The Full Resynchronize button is the reset and recalculate function. Select the node or nodes to be updated with manual changes, then click the **Full Resynchronize** button.
- ▶ Use the **syncNode** command. This command is a stand-alone program that runs separately from the node agent. It has no cache of epoch values that could be used for an optimized synchronization, therefore performing a complete synchronization. For this same reason, if you restart a node agent, the very first synchronization it performs will always be a complete synchronization. Note that this requires the node agent to be stopped.

The **syncNode** command resides in the bin directory of the base install. To use the **syncNode** command, type the following from the command line:

```
cd <profile_home>\bin  
syncNode <cell_host>
```

Example 2-3 shows the use of the **snycNode** command.

Example 2-3 Using the syncNode command

```
<install_root>\profiles\<server_name>\bin>stopnode
ADMU0116I: Tool information is being logged in file

<install_root>\profiles\<server_name>\logs\nodeagent\stopServer.log
ADMU0128I: Starting tool with the AppSrv01 profile
ADMU3100I: Reading configuration for server: nodeagent
ADMU3201I: Server stop request issued. Waiting for stop status.
ADMU4000I: Server nodeagent stop completed.

<install_root>\profiles\<server_name>\bin>syncnode carlavm2
ADMU0116I: Tool information is being logged in file
<install_root>\profiles\<server_name>\logs\syncNode.log
ADMU0128I: Starting tool with the AppSrv01 profile
ADMU0401I: Begin syncNode operation for node AppSrvrNode with Deployment
Manager carlavm2: 8879
ADMU0016I: Synchronizing configuration between node and cell.
ADMU0402I: The configuration for node AppSrvrNode has been synchronized with
Deployment Manager carlavm2: 8879
```

Tip: The repository is flexible in that there is no predefined list of document types that it permits. You can add any file you want. Perhaps you have some unique configuration data that needs to be used on all nodes. You could put it in the config/cells/<cell name> folder and it would be synchronized to all nodes. If it applies to just one node, you could put it in the folder corresponding to that node and it would be synchronized only to that node. The same applies for any additional documents in a server level folder.

As a way to use this tip, under normal circumstances, all application files are packaged in the EAR file for the application. However, consider a configuration file specific to an application. Any changes to that file would require that you update the EAR file and synchronize the entire application.

One possibility is to put a properties file in the application deployment directory in the master configuration repository, so that it is replicated to all nodes where the application is installed automatically but the entire EAR is not replicated. Then you could have an ExtensionMBean update the properties file in the master repository and normal synchronization would replicate just those changes out to the nodes without the need to synchronize the whole EAR and restart the application.

2.4 Configuration and application data repository

The configuration and application data repository is a collection of files containing all the information necessary to configure and execute servers and their applications. Configuration files are stored in XML format, while application data is stored as EAR files and deployment descriptors.

2.4.1 Repository directory structure

With V6.x, the directory structure of a WebSphere Application Server installation is slightly different than in previous releases. We will discuss this in detail in Chapter 3, “Getting started with profiles” on page 47, but for now, it is important to know configuration files defining a run time environment are stored in profile directories. Each node, deployment manager, and stand-alone application server has its own profile directory under the `<was_home>/profiles` directory.

Note: In the rest of this book, when we talk about a specific profile directory, located at, `<was_home>/profiles/<profile_name>`, we will refer to it as the `<profile_home>` directory.

When we are speaking specifically of the profile directory for the deployment manager, we will refer to it as `<dmgr_profile_home>`.

The repository files are arranged in a set of cascading directories under each profile directory structure, with each directory containing a number of files relating to different components of the cell. You can see this in Figure 2-10. The repository structure follows the same format, regardless of whether you have a stand-alone server environment or distributed server environment.

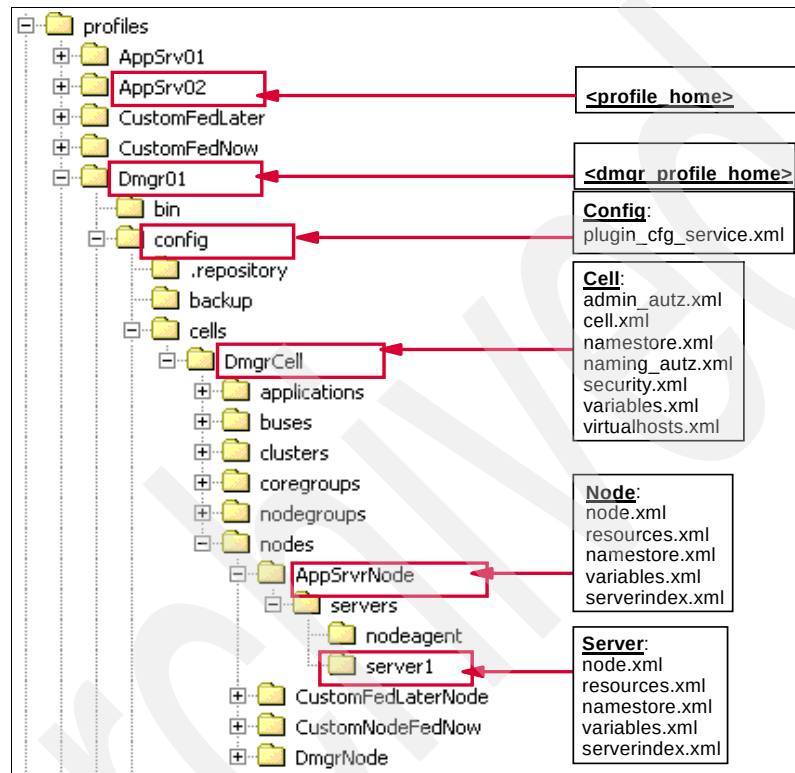


Figure 2-10 Repository directory structure

The `<profile_home>/config` directory is the root of the repository for each profile. It contains the following directory structure:

- `cells/<cell>/`

This is the root level of the configuration for the cell. The directory contains a number of cell-level configuration files. Depending on the types of resources that have been configured, you might see the following subdirectories:

- `cells/<cell>/applications/` contains one subdirectory for every application that has been deployed within the cell.

- cells/<cell>/buses/ contains one directory for each service integration bus (bus) defined.
- cells/<cell>/coregroups/ contains one directory for each core group defined.
- cells/<cell>/nodegroups/ contains one directory for each node group defined.
- cells/<cell>/nodes/ contains the configuration settings for all nodes and servers managed as part of this cell. The directory contains one directory per node. Each cells/<cell>/nodes/<node> directory will contain node-specific configuration files and a server directory which in turn will contain one directory per server and node agent on that node.
- cells/<cell>/clusters/ contains one directory for each of the clusters managed as part of this cell. Each cluster directory contains a single file, cluster.xml, which defines the application servers of one or more nodes that are members of the cluster.

The overall structure of the master repository is the same for both a stand-alone server environment and a distributed server environment. The differences are summarized in the following sections.

In a stand-alone server environment, the structure has the following:

- ▶ The master repository is held on a single machine. There are no copies of this specific repository on any other node.
- ▶ The repository contains a single cell and node.
- ▶ There is no node agent because each application server is stand-alone, so there is no directory for the node agent (nodeagent).
- ▶ Clusters are not supported, and therefore will not contain the clusters directory or subdirectories.

In a distributed server environment, the structure has the following characteristics:

- ▶ The master repository is held on the node containing the deployment manager. It contains the master copies of the configuration and application data files for all nodes and servers in the cell.
- ▶ Each node also has a local copy of the configuration and application data files from the master repository that are relevant to the node.
- ▶ Changes can be made to the configuration files on a node, but the changes will be temporary. Such changes will be overwritten by the next file synchronization from the deployment manager. Permanent changes to the configuration require changes to the file or files in the master repository.

Configuration changes made to node repositories are not propagated up to the cell.

- The applications directory of the master repository contains the application data (binaries and deployment descriptors) for all applications deployed in the cell. The local copy of the applications directory on each node will only contain the directories and files for the applications deployed on application servers within that node.

Information about the individual files found in each of these directories can be found in the *Configuration Document Descriptions* topic in the Information Center.

2.4.2 Variable scoped files

Identically named files that exist at differing levels of the configuration hierarchy are termed *variable scoped* files. There are two uses for variable scoped files:

- Configuration data contained in a document at one level is logically combined with data from documents at other levels of the configuration hierarchy. In the case of conflicting definitions, the “most specific” value takes precedence. For example, if an identical entry exists in the files at the cell and node level (as with a variable defined in both the cell and node’s `variables.xml`), the entry at the node level takes precedence.
- Documents representing data that is not merged but is rather scoped to a specific level of the topology. For example, the `namestore.xml` document at the cell level contains the cell persistent portion of the name space, while the `namestore.xml` at the node level contains the node persistent root of the name space.

2.4.3 Application data files

The master repository is also used to store the application binaries (EAR files) and deployment descriptors. This allows modified deployment descriptors to be kept in the repository, and allows system administrators to make application updates more automatic.

The `<profile_home>/config` directory of the master repository contains the following directory structure used to hold application binaries and deployment settings:

- `cells/<cell>/applications/`

This directory contains a subdirectory for each application deployed in the cell. The names of the directories match the names of the deployed applications.

Note: The name of the deployed application does not have to match the name of the original EAR file used to install it. Any name can be chosen when deploying a new application, as long as the name is unique across all applications in the cell.

- ▶ cells/<cell>/applications/<appname>.ear

Each application's directory in the master repository contains the following:

- A copy of the original EAR, called <appname>.ear, which does not contain any of the bindings specified during the installation of the application
- A deployments directory, which contains a single <appname> directory used to contain the deployed application configuration

- ▶ cells/<cell>/applications/<appname>.ear/deployments/<appname>

The deployment directory of each application contains the following:

- deployment.xml

This file contains configuration data for the application deployment, including the allocation of application modules to application servers, and the module startup order.

- META-INF/

This directory contains the following:

- application.xml
J2EE standard application deployment descriptor
- ibm-application-bnd.xmi
IBM WebSphere-specific application bindings
- ibm-application-ext.xmi
IBM WebSphere-specific application extensions
- was.policy
Application-specific Java 2 security configuration

This file is optional. If not present, then the policy files defined at the node level will apply for the application.

Note: The deployment descriptors stored in the repository contain the bindings chosen during application installation.

The subdirectories for all application modules (WARs and EJB JARs) are contained in the was.policy along with each module's deployment descriptors.

Note: The subdirectories for each module do not contain application binaries (JARs, classes, and JSPs), only deployment descriptors and other configuration files.

Repository files used for application execution

The installation of an application onto a WebSphere Application Server application server results in:

- ▶ The storage of the application binaries (EAR) and deployment descriptors within the master repository.
- ▶ The publishing of the application binaries and deployment descriptors to each node that will be hosting the application. These files are stored in the local copy of the repository on each node.

Each node then installs applications ready for execution by exploding the EARs under the `<profile_home>/installedApps/<cell>/` as follows:

- ▶ `<profile_home>/installedApps/<cell>/`

This directory contains a subdirectory for each application deployed to the local node.

Note: The name of each application's directory reflects the name under which the application is installed, not the name of the original EAR. For example, if an application is called myapp, then the `installedApps/<cell>` directory will contain a `myapp.ear` subdirectory.

- ▶ `<profile_home>/installedApps/<cell>/<appname>.ear/`

Each application-specific directory contains the contents of the original EAR used to install the application.

- The deployment descriptors from the original EAR. These descriptors do not contain any of the bindings specified during application deployment.
- All application binaries (JARs, classes, and JSPs)

Figure 2-11 summarizes how the node's local copy of the repository contains the application's installed deployment descriptors, while the directory under `installedApps` contains the application binaries.

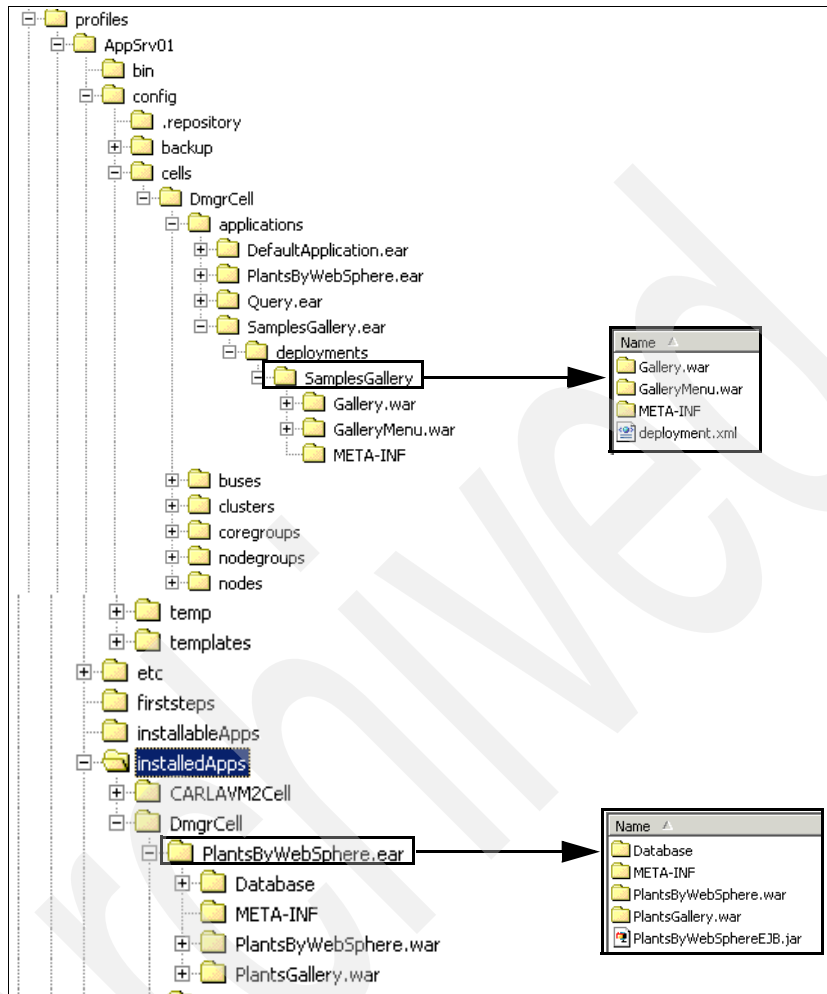


Figure 2-11 Location of application data files

By default, a WebSphere Application Server application server executes an application by performing the following tasks:

1. Loading the application binaries stored under:

`<profile_home>/installedApps/<cell>/<appname>.ear/`

You can change this location by altering the Application binaries setting for the enterprise application or by altering the `$(APP_INSTALL_ROOT)` variable setting.

2. Configuring the application using the deployment descriptors stored under:
`<profile_home>/config/cells/<cell>/applications/<appname>.ear/deployments/<appname>`

You can change this for applications deployed to V6.x application servers by modifying the Use metadata from binaries setting for the enterprise application. This is the Use Binary Configuration field on the application installation and update wizards.

By default, the setting is not enabled. Enabling it specifies that you want the application server to use the binding, extensions, and deployment descriptors located in the application EAR file rather than those stored in the deployments directory.

Getting started with profiles

Installing a WebSphere Application Server environment requires careful planning. A major decision point is the topology for the system. These decisions include, for example, whether you will have a stand-alone server, a distributed managed server environment, clustering, and so forth.

These topics are covered in detail in *Planning and Designing for WebSphere Application Server V6.1*, SG24-7305. That IBM Redbook is designed to help you select a topology and develop a clear idea of what steps are needed to set up your chosen environment. Your options will depend on your chosen WebSphere Application Server package. The installation process is well-documented in the installation guide packaged with the product.

The purpose of this chapter is to help you build your initial WebSphere Application Server environment after you have installed the product. It includes the following topics:

- ▶ Understanding profiles
- ▶ Building a system using profiles
- ▶ Creating profiles on distributed systems (non z/OS)
- ▶ Managing profiles
- ▶ Managing the processes

Important: This chapter assumes you are performing a new installation. For migration issues, see *WebSphere Application Server V6.1 Migration Guide*, SG24-6369.

3.1 Understanding profiles

The WebSphere Application Server installation process simply lays down a set of core product files required for the run time processes. After installation, you need to create one or more *profiles* that define the run time to have a functional system. The core product files are shared among the run time components defined by these profiles.

With Base and Express, you can only have stand-alone application servers, as shown in Figure 3-1. Each application server is defined within a single cell and node. The administration console is hosted within the application server and can only connect to that application server. No central management of multiple application servers is possible. An application server profile defines this environment. You can also create stand-alone application servers with the Network Deployment package, though you would most likely do so with the intent of federating that server into a cell for central management.

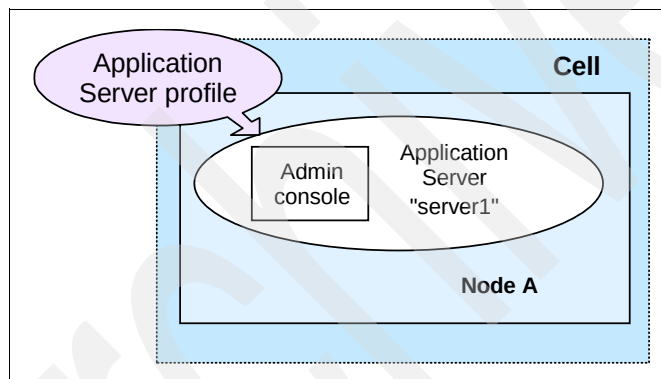


Figure 3-1 System management topology: Stand-alone server (Base and Express)

With the Network Deployment package, you have the option of defining multiple application servers with central management capabilities, as summarized in Figure 3-2 on page 49. The administration domain is the cell, consisting of one or more nodes. Each node contains one or more application servers and a node agent that provides an administration point management by the deployment manager.

The deployment manager can be located on the same machine as one or more of the application servers. This would be a common topology for single machine development and testing environments. In most production topologies, we recommend that the deployment manager be placed on a separate dedicated machine.

The basis for this run time environment starts with the deployment manager that provides the administration interface for the cell. As you would expect, the deployment manager is defined by a deployment manager profile.

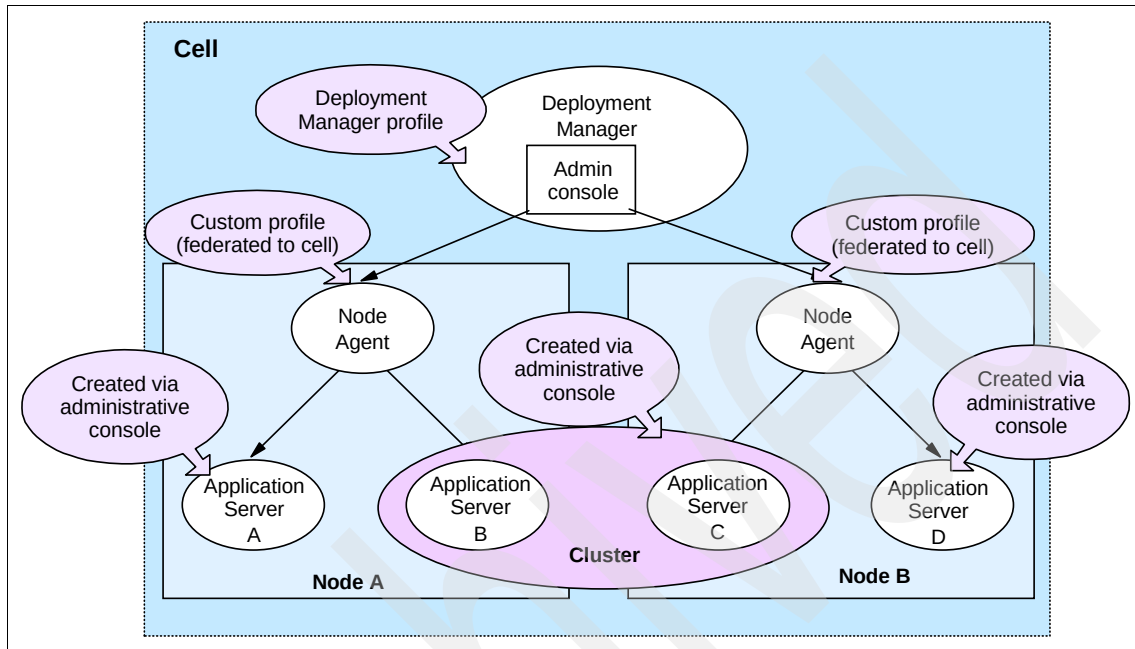


Figure 3-2 System management topology: Network Deployment

Nodes can be added to the cell in one of two ways:

- ▶ You can create an application server profile, then federate it to the cell. When a node is added to a cell, a node agent is created on the node and configuration files for the node are added to the master configuration repository for the cell. The deployment manager then assumes responsibility for the configuration of all servers on the node.

Note that the server name for a federated application server is always going to be “server1”.

- ▶ You can define a custom profile to create an empty node for federation to the cell. After federation, you can further configure the node by creating application servers and clusters from the deployment manager administrative console. If you are using a naming convention for servers, this is the best option.

3.1.1 Types of profiles

We mentioned the types of profiles available for defining the run time. In the following sections, we take a closer look at these profiles.

Application server profile

The application server profile defines a single stand-alone application server. Using this profile gives you an application server that can run stand-alone, or unmanaged. The environment will have the following characteristics:

- ▶ The profile consists of one cell, one node, and one server. The cell and node are not relevant in terms of administration, but you see them when you administer the server through the administrative console scopes.
- ▶ The name of the application server is “server1”.
- ▶ The application samples are installed on the server (optional).
- ▶ The server has a dedicated administrative console.

The primary use for this type of profile is:

- ▶ To build a stand-alone server in a Base or Express installation.
- ▶ To build a stand-alone server in a Network Deployment installation that is not managed by the deployment manager (a test machine, for example).
- ▶ To build a server in a distributed server environment to be federated and managed by the deployment manager. If you are new to WebSphere Application Server and want a quick way of getting an application server complete with samples, this is a good option. When you federate this node, the default cell becomes obsolete and the node is added to the deployment manager cell. The server name remains “server1” and the administrative console is removed from the application server.

Deployment manager profile

The deployment manager profile defines a deployment manager in a distributed server environment. Although you could conceivably have the Network Deployment package and run only stand-alone servers, this would bypass the primary advantages of Network Deployment, which is workload management, failover, and central administration.

In a Network Deployment environment, you should create one deployment manager profile. This gives you:

- ▶ A cell for the administrative domain
- ▶ A node for the deployment manager
- ▶ A deployment manager with an administrative console
- ▶ No application servers

Once you have the deployment manager, you can:

- ▶ Federate nodes built either from existing application server profiles or custom profiles.
- ▶ Create new application servers and clusters on the nodes from the administrative console.

Custom profile

A custom profile is an empty node, intended for federation to a deployment manager. This type of profile is used when you are building a distributed server environment. Use a custom profile in the following way:

1. Create a deployment manager profile.
2. Create one custom profile on each node on which you will run application servers.
3. Federate each custom profile to the deployment manager, either during the custom profile creation process or later by using the **addNode** command.
4. Create new application servers and clusters on the nodes from the administrative console.

Cell profile

Cell profile (new): This new option allows you to quickly set up a distributed server environment on a single system.

A cell profile is actually a combination of two profiles: a deployment manager profile and an application server profile. The application server profile is federated to the cell. The deployment manager and application server reside on the same system. This type of profile lets you get a quick start with a distributed server environment and is especially useful for test environments that typically have all nodes on one test system.

3.1.2 Directory structure and default profiles

If you have worked with previous versions of WebSphere Application Server, you will notice a difference in the directory structure. First, all packages (Base, Express, and Network Deployment) specify the same default root directory during installation. For example, in Windows installations, this is commonly c:\Program Files\IBM\WebSphere\AppServer. In this IBM Redbook, we refer to this root directory as the *<was_home>* directory.

In addition to the traditional directories under the *<was_home>* directory (bin, config, installedapps, and so on), you now have a profiles directory containing a

subdirectory for each profile you create and allow to use the default home location. The directory structure for each profile resembles the primary structure. In other words, there is a bin, config, installedApps, and other directories required for a unique run time under each profile.

For example, if you installed on a Windows system, and created a profile named AppSrvr01, you would normally see a directory structure like that shown in Figure 3-3 on page 52.

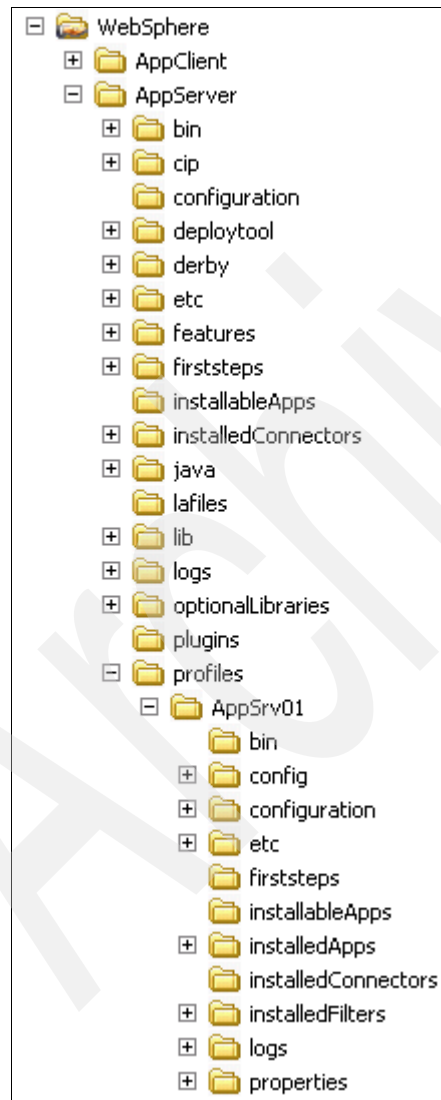


Figure 3-3 Directory structure

However, profiles can be stored in any folder, so we suggest storing them in a more friendly structure (by default, there are at least six levels). We refer to the root of each profile directory (by default `<was_home>/profiles/profile_name`) as `<profile_home>`.

Why do we emphasize this point? If you enter commands while in the `<was_home>/bin` directory, they are executed against the run time defined by the default profile. The default profile is determined by the following:

- ▶ The profile was defined as the default profile when you created it. The last profile specified as the default takes precedence. You can also use the **manageprofiles** command to specify the default profile.
- ▶ If you have not specified the default profile, it will be the first profile you created.

To make sure command line operations are executed for the correct run time, you need to do one of two things:

- ▶ Specify the `-profileName` option when using a command and execute the command from the `<was_home>/bin` directory.
- ▶ Execute the command from its `<profile_home>/bin` directory.

z/OS considerations

The configuration information for a profile is kept in the HFS or zFS depending on how your system is set up. A unique directory serves as the mount point for each file system that will hold a profile. The name of the mount point can be anything you want and can be set during the profile customization process that is covered in the following sections.

Under the mount point, you will find two directories, one for the daemon server and the other for the profile. Figure 3-4 on page 54 shows an application server profile. You can see the structure of the files under the mount point, including the application server directory and the daemon directory.

The daemon directory structure is similar to the profile directory structure and the name “Daemon” is fixed, although the “AppServer” name can be changed during the profile customization process that is covered in the following sections.

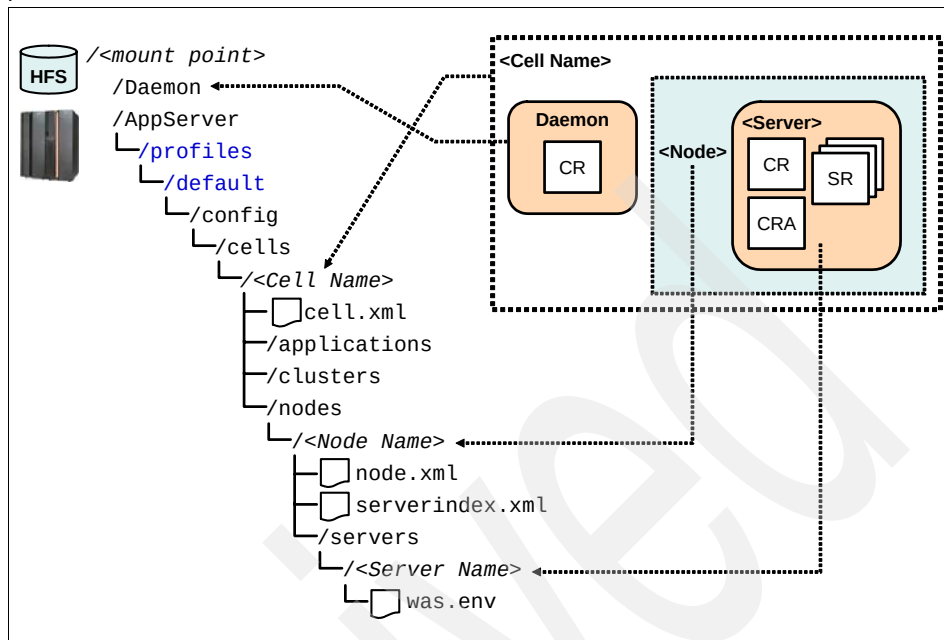


Figure 3-4 z/OS directory structure

If you check under the `AppServer/profiles/default/config/cells` directory, you will see a directory that will have the same name as you provided for the cell long name during the customization for this profile. Further down in the structure, under the `/nodes` directory, there will be a directory with the same name you provided for the node long name

Note: Under profile you will see default. The profile name you created from your workstation is not reflected here. The name “default” is always used in WebSphere Application Server V6.1 for z/OS. Profile names cannot be created or modified directly.

3.2 Building a system using profiles

During the planning cycle, a topology was selected for the WebSphere Application Server environment. There are many topologies to choose from, each with its own unique features.

However, when we discuss using profiles to build a WebSphere Application Server environment, we are focusing on the WebSphere Application Server processes or daemons for z/OS. Regardless of the topology you select, there are

really only two primary situations to consider when deciding which profiles you need to create:

- ▶ You plan to create one or more stand-alone application servers. We will refer to this as a *stand-alone server environment*.
- ▶ You plan to create a deployment manager and one or more nodes with application servers. We refer to the application servers in this environment as managed servers. These nodes can coexist or reside on different machines. We refer to this as a *distributed server environment*.

The following topics will give the basic steps for each. You can extend this to suit your own environment.

3.2.1 Stand-alone server environment

If you are creating a stand-alone application server, do the following:

1. Install your choice of Base, Express, or Network Deployment on the system.
An application server profile is created during the installation of Express and Base. With Network Deployment, you have the option of creating a profile of any type, including an application server profile.
2. Create an application server profile on that system. Since you have an application server automatically after Base and Express installation, you only need to do this if you want an additional stand-alone server environment.

3.2.2 Distributed server environment

There are two options for building this environment. The option you select depends on your circumstance. If you are building a new production environment from scratch, we would recommend method 1. Either method is fine for a development or test environment.

Note: When defining multiple deployment managers or application servers on a single machine or LPAR, you need to ensure that the ports and names you select for each are unique. For more information about ports, see *Planning and Designing for WebSphere Application Server V6.1*, SG24-7305.

Method 1

This method assumes that you do not have a stand-alone application server to federate, but instead will create application servers from the deployment manager. This gives you a little more control over the characteristics of the application servers during creation, including the server name (all application servers created with the application server profile are named server1). You can

also create an application server, customize it, and then use it as a template for future application servers you create. If you are using clustering, you can create the cluster and its application servers as one administrative process.

When you create an application server this way, you do not automatically get the sample applications, but can install them later if you want.

The process to follow for this method is:

1. Install Network Deployment on a server. If this is a multiple-machine install with the deployment manager on one machine and application servers on one or more separate machines, install the product on each machine.
2. Create a deployment manager profile on the deployment manager machine and start the deployment manager.
3. Create and federate a custom profile on the application server machine and start the node. You can federate the node to the cell as part of the profile creation process, or you can elect to do it manually as a second step.
4. Verify that the node agent is started. It should be started automatically as part of the federation process.
5. Open the deployment manager's administrative console, then create application servers or clusters on the custom profile node from the administrative console.

Method 2

This method assumes you will federate an application server profile to the cell. With the application server profile, you have an existing application server (server1) and might have applications installed, including the sample applications and any user applications you have installed.

1. Install Network Deployment on the server. If this is a multiple machine install (deployment manager on one and application servers on one or more separate machines), install the product on each machine.
2. Create a deployment manager profile on the deployment manager machine and start the deployment manager.
3. Create an application server profile on the application server machine and start the application server.
4. Open the deployment manager's administrative console and add the node defined by the application server profile to the cell.
5. This deletes the application server cell, and federates the node to the deployment manager cell. If you want to keep applications that have been installed on the server, be sure to specify this when you federate the node.

6. The new node agent is started automatically by the federation process, but you need to start the application server manually.

3.3 Creating profiles on distributed systems (non z/OS)

This section shows how to create profiles using the Profile Management Tool. Note that the Profile Management Tool is not available on 64-bit or Linux on System z platforms.

Silent install: You can also create profiles in silent mode using the `manageprofiles` command (see “Creating a profile in silent mode” on page 125).

The first steps are common, regardless of the type of profile you will create. You can start the Profile Management Tool in one of the following ways:

1. From the Start menu in Windows only, select **Start → Programs → IBM WebSphere → Application Server Network Deployment v6.1 → Profile Management Tool**.
2. Use the platform-specific command in the `<was_home>/bin/ProfileManagement` directory:
 - Windows: `pmt.bat`
 - Linux/HP-UX/Solaris/AIX: `pmt.sh`
3. Check the box directly after installation from the install wizard to launch the Profile Management Tool.

When you start the wizard, the first window you see is the Welcome window. Click **Next** to select the type of profile you will create, as in Figure 3-5 on page 58.



Figure 3-5 Creating a profile: Profile type selection

The rest of the wizard varies, depending on the type of profile you are creating. The steps to create each type of profile are discussed more in the following sections.

Default profiles: As you create profiles, you will have the option of specifying a default profile. This is the profile that commands are executed against if you execute them from the `<was_home>/bin` directory and you do not specify the `-profileName` argument. The default profile is the first profile that you create, unless you subsequently specify another profile as the default. To see this option, you must take the Advanced path through the Profile Management Tool.

First Steps: At the end of the Profile Management Tool, you have the opportunity to start the First Steps interface. This interface helps you start the deployment manager or application server and has other useful links, such as opening the administrative console, migration help, starting the Profile Management Tool, and installation verification.

Each profile you create has its own First Steps program located here:

```
<profile_home>/firststeps/firststeps.bat (.sh)
```

If you choose not to start the First Steps program at the completion of the wizard, you can start it later from this location.

You will always have two options when using the Profile Management Tool to create a profile. The “Typical” path will determine a set of default values to use for most settings without giving you the option to modify them. The “Advanced” path lets you specify values for each option.

3.3.1 Creating a deployment manager profile

Table 3-1 shows a summary of the options you have for creating a deployment manager. The table shows the options and results you will see depending on which path (typical or advanced) you take.

Table 3-1 Deployment manager profile options

Typical settings	Advanced options
The administrative console is deployed by default.	You can choose whether to deploy the administrative console. We recommend that you do so.
The profile name is Dmgrxx by default, where xx is 01 for the first deployment manager profile and increments for each one created. The profile is stored in <was_home>/profiles/Dmgrxx.	You can specify the profile name and its location.
The cell name is <hostname>Cellxx. The node name is <hostname>CellManagerxx. The host name is prefilled in with your system's host name.	You can specify the node, host, and cell names.
You can enable administrative security (yes or no). If you select yes, you will be asked to specify a user name and password that will be given administrative authority.	

Typical settings	Advanced options
TCP/IP ports will default to a set of ports not used by any profiles in this WebSphere installation instance.	You can use the recommended ports (unique to the installation), use the basic defaults, or select port numbers manually.
(Windows) The deployment manager will be run as service.	(Windows) You can choose whether to the deployment manager will run as a service.

The following steps outline the process of creating a deployment manager.

1. Start the Profile Management Tool and click **Next** on the Welcome page.
2. Select the **deployment manager profile** option. Click **Next**.
3. Select whether to take the typical settings or to go through the advanced windows. The options you see next depend on the path you take.
 If **Typical** is selected, then you will only see one more option (to enable security).
 If **Advanced** is selected, you will continue with the following steps.
4. Select whether to deploy the administrative console application. This is recommended, but if you choose not to, you can install it after profile creation.

5. Enter a unique name for the profile or accept the default. The profile name will become the directory name for the profile files (see Figure 3-6). Click the box if you want this to be the default profile for receiving commands. Select the location for the profile and click **Next**.

Specify a profile name and directory path to contain the files for the run-time environment, such as commands, configuration files, and log files. Click **Browse** to select a different directory.

Profile name:
DmgrProfile1

Profile directory:
c:\myWAS61Profiles\dmgrProfiles\DmgrProfile1

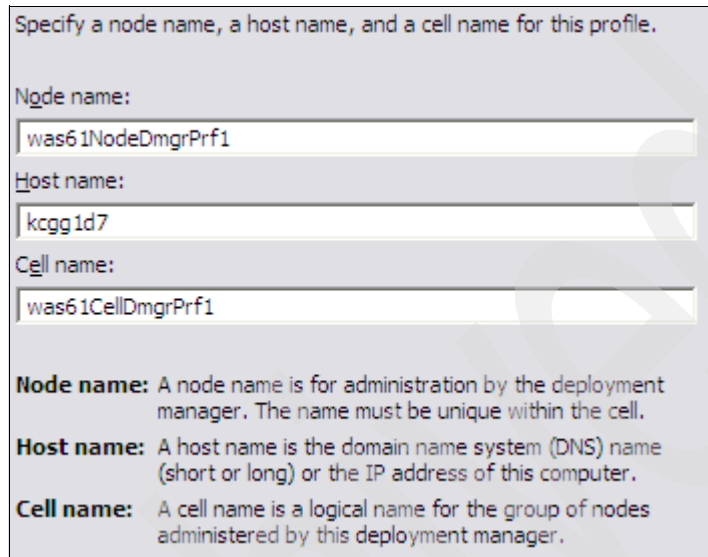
☒ Make this profile the default.

Each installation of WebSphere Application Server always has one default profile. Commands that run without referring to a specific profile use the default profile. Select this option to make this profile the new default

Important: Deleting the directory a profile is in does not completely delete the profile. Use the **manageprofiles** command to completely delete a profile.

Figure 3-6 Creating a deployment manager profile: Enter name and location

6. Enter the node, host, and cell names. These default based on the host name of your system. The wizard recognizes if there are existing cells and nodes in the installation and takes this into account when creating the default names. See Figure 3-7 on page 62.



Specify a node name, a host name, and a cell name for this profile.

Node name:
was61NodeDmgrPrf1

Host name:
kcg1d7

Cell name:
was61CellDmgrPrf1

Node name: A node name is for administration by the deployment manager. The name must be unique within the cell.

Host name: A host name is the domain name system (DNS) name (short or long) or the IP address of this computer.

Cell name: A cell name is a logical name for the group of nodes administered by this deployment manager.

Figure 3-7 Creating a deployment manager profile: Enter cell, host, and node names

Click **Next**.

7. Choose whether to enable administrative security. If you enable security here, you will be asked for a user ID and password that will be added to a file-based user registry with the Administrative role. Click **Next**.
8. The wizard presents a list of TCP/IP ports for use by the deployment manager. If you already have existing profiles on the system, this is taken into account when the wizard selects the port assignments. However, you should verify that these ports will be unique on the system. See Figure 3-8 on page 63.

The values in the following fields define the ports for the deployment manager and do not conflict with other profiles in this installation. Another installation of WebSphere Application Server or other programs might use the same ports. To avoid run-time port conflicts, verify that each port value is unique.

Default Port Values	Recommended Port Values
Administrative console port (Default 9060):	9060
Administrative console secure port (Default 9043):	9043
Bootstrap port (Default 9809)(W):	9809
SOAP connector port (Default 8879)(X):	8879
SAS SSL ServerAuth port (Default 9401)(Z):	9401
CSTV2 ServerAuth listener port (Default 9403)(1):	9403

Figure 3-8 Creating a deployment manager profile: Select ports

Note two ports: You might want to note the following ports for later use:

- ▶ *SOAP connector port:* If you use the `addNode` command to federate a node to this deployment manager, you need to know this port number. This is also the port you connect to when using the `wsadmin` administration scripting interface.
- ▶ *Administrative console port:* You need to know this port in order to access the administrative console. When you turn on security, you need to know the *Administrative console secure port*.

9. On Windows systems, you have the option of running the deployment manager as a service. This provides you a simple way of automatically starting the deployment manager when the system starts. If you would like to run the process as a Windows service, check the box and enter the values for the logon and startup type. See Figure 3-9 on page 64.

Choose whether to use a Windows service to run WebSphere Application Server. Windows services can start and stop WebSphere Application Server, and configure startup and recovery actions.

☒ Run the deployment manager process as a Windows service.

☒ Log on as a local system account.

☐ Log on as a specified user account.

User name:
fagalde

Password:

Startup type:
Automatic

The user account that runs the Windows service must have the following user rights:
- Log on as a service

Figure 3-9 Creating a deployment manager profile: Run as a Windows service

Note that the window lists the user rights the user ID you select needs to have. If the user ID does not have these rights, the wizard will automatically add them.

Click **Next**.

10. Review the options you have chosen and click **Next** to create the profile. After the wizard has finished, you will be presented with the window in Figure 3-10 on page 65.

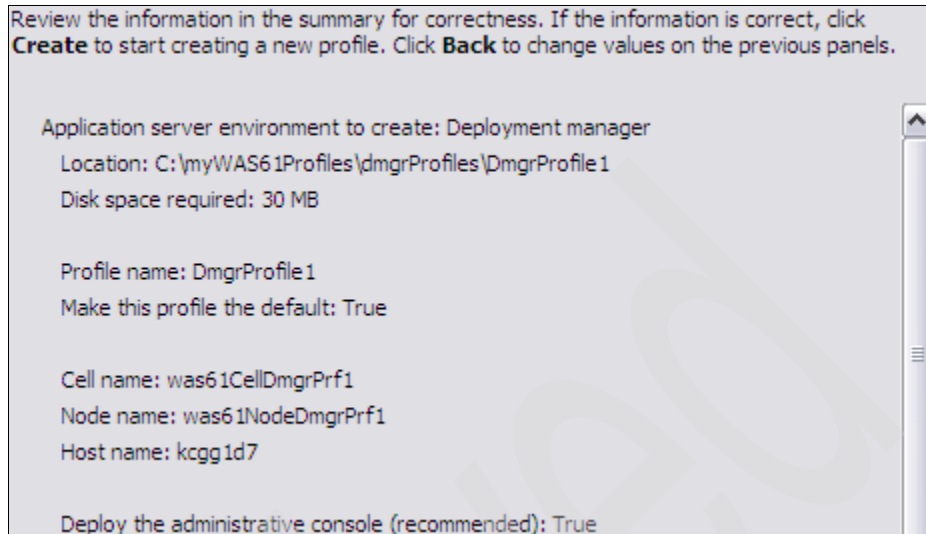


Figure 3-10 Creating a deployment manager profile: Finish

This final window indicates the success or failure of the profile creation. If you have errors, check the log at:

`<was_home>/logs/manageprofiles/<profile_home>_create.log`

You will also find logs for individual actions stored in:

`<profile_home>/logs`

11. Click **Finish** to close the wizard and start the First Steps application, as shown in Figure 3-11.

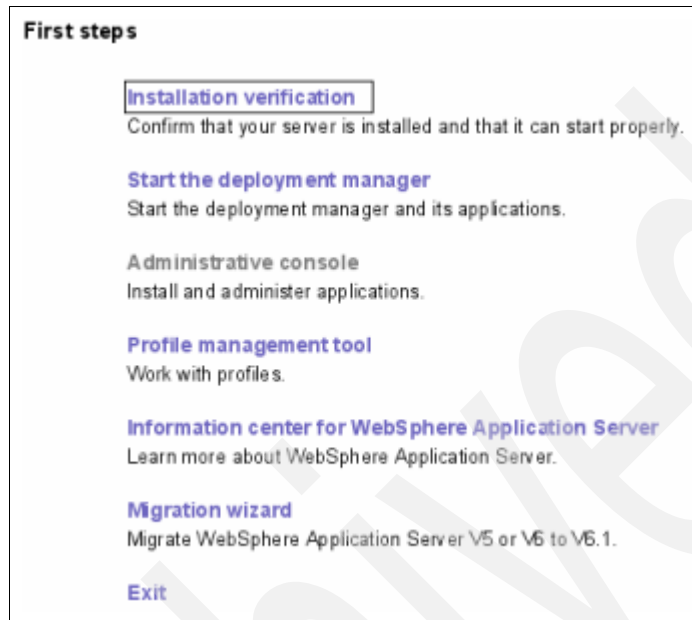


Figure 3-11 Deployment manager First Steps menu

Check your results

If the creation was successful, do the following to familiarize yourself with the profile and how to use it:

1. View the directory structure and find the new profile. In this IBM Redbook, we refer to the location as `<profile_home>`. This is where you find, among other things, the config directory containing the deployment manager configuration files, the bin directory for entering commands, and the logs directory where information is recorded.
2. Verify the installation. You can do this directly from the First Steps menu. This process starts the deployment manager and checks the log file for warnings or errors on start. Messages are displayed on the First Steps window and logged in the following places:
 - `<profile_home>/logs/dmgr/startServer.log`
 - `<profile_home>/logs/dmgr/SystemOut.log`
3. Open the administrative console, either by selecting the option in the First Steps window, or by accessing its URL from a Web browser:
`http://<dmgr_host>:<admin_console_port>/ibm/console`

Here is a sample URL in the address bar:

```
http://localhost:9060/ibm/console/
```

The administrative console port of 9060 was selected during the Profile Management Tool. See Figure 3-8 on page 63.

Click the **Log in** button. If you did not enable security, you do not have to enter a user name. If you choose to enter a name, it can be any name. It is used to track changes you make from the console. If you enabled security, enter the user ID and password you specified.

4. Display the configuration from the console. You should be able to see the following items from the administrative console:
 - a. Cell information: Select **System administration** → **Cell**.
 - b. Deployment manager: Select **System administration** → **Deployment manager**.
 - c. Deployment manager node: Select **System administration** → **Nodes**.
 - d. The default node group: Select **System administration** → **Node groups**.

Note that at the completion of this process you will not have:

- a. A node agent

Node agents reside on nodes with managed application servers. You will not see node agents appear until you federate a node to the cell.
 - b. Application servers
5. Stop the deployment manager. You can do this from the First Steps menu, or better yet, use the **stopManager** command:

```
cd <profile_home>\bin
stopManager
```

On a UNIX system, use the following command:

```
cd <profile_home>/bin
stopManager.sh
```

Tip: In the same manner, you can use the **startManager** command to start the deployment manager.

3.3.2 Creating an application server profile

An application server profile defines a new stand-alone application server. This server can be run stand-alone or can be later federated to a deployment manager cell for central management.

Table 3-2 shows a summary of all steps involved in process of creating a application server profile.

Table 3-2 Application server profile options - V6.1

Typical	Advanced
The administrative console and default application are deployed by default. The sample applications are not deployed.	You have the option to deploy the administrative console (recommended), the default application, and the sample applications (if installed).
The profile name is AppSrvxx by default, where xx is 01 for the first application server profile and increments for each one created. The profile is stored in <was_home>/profiles/AppSrvxx.	You can specify the profile name and its location.
The profile is not the default profile.	You can choose whether to make this the default profile. (Commands run without specifying a profile will be run against the default profile.)
The application server is built using the default application server template.	You can choose the default template, or a development template that is optimized for development purposes.
The node name is <host>Nodexx. The host name is prefilled in with your system's DNS host name.	You can specify the node name and host name.
You can enable administrative security (yes or no). If you select yes, you will be asked to specify a user name and password that will be given administrative authority.	
TCP/IP ports will default to a set of ports not used by any profiles in this WebSphere installation instance.	You can use the recommended ports (unique to the installation), use the basic defaults, or select port numbers manually.
(Windows) The deployment manager will be run as a service.	(Windows) You can choose whether the deployment manager will run as a service.
Does not create a Web server definition.	Allows you to define an external Web server to the configuration.

This section takes you through the steps of creating the application server profile:

1. Start the Profile Management Tool. Click **Next** on the Welcome page.
2. Select the **Application server profile** option. Click **Next**.
3. Select the kind of creation process you want to run: **Typical** or **Advanced**.

If Typical is selected, then you will only see one more option (to enable security).

If Advanced is selected, you will continue with the next step.

4. Select whether you want to deploy the administrative console and the default application. If you have installed the sample applications (optional during WebSphere Application Server installation), then you can opt to deploy these as well.
5. Enter a unique name for the profile or accept the default. The profile name will become the directory name for the profile files. See Figure 3-12.

Click the box if you want this directory to be the default profile for receiving commands.

If the application server will be used primarily for development purposes, check the option to create it from the development template.

Click **Next**.

Specify a profile name and directory path to contain the files for the run-time environment, such as commands, configuration files, and log files. Click **Browse** to select a different directory.

Profile name:
AppSrvProfile1

Profile directory:
C:\myWAS61Profiles\appSrvProfiles\AppSrvProfile1

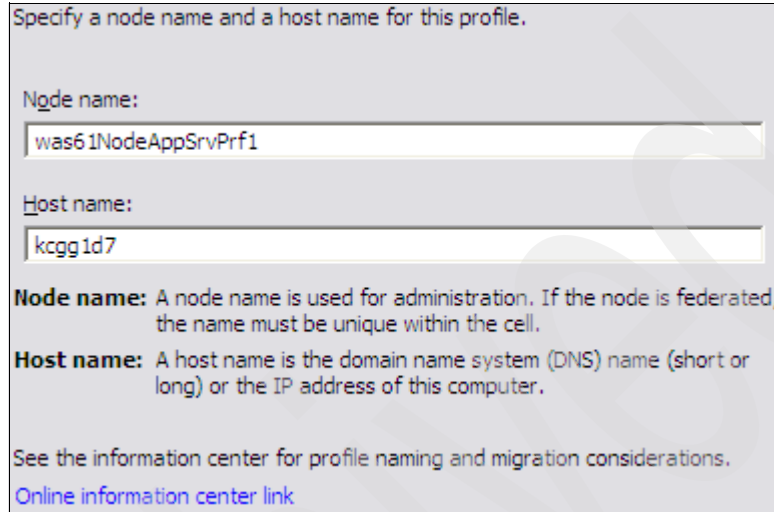
Browse...

☐ Create the server using the development template.
Select this option to create a server using configuration settings optimized for development. The development template reduces startup time and allows the server to run on less powerful hardware. Do not use this option for production servers.

☐ Make this profile the default.
Each installation of WebSphere Application Server always has one default profile. Commands that run without referring to a specific profile use the default profile. Select this option to make this profile the new default

Figure 3-12 Creating an application server profile: Enter name and location

6. Enter the new node name and the system host name. See Figure 3-13. The node name will default based on the host name of your system. The wizard recognizes if there are existing nodes in the installation and takes this into account when creating the default node name. Click **Next**.



Specify a node name and a host name for this profile.

Node name:
was61NodeAppSrvPrf1

Host name:
kcg1d7

Node name: A node name is used for administration. If the node is federated, the name must be unique within the cell.

Host name: A host name is the domain name system (DNS) name (short or long) or the IP address of this computer.

See the information center for profile naming and migration considerations.
[Online information center link](#)

Figure 3-13 Creating an application server profile: Enter host and node names

Note: If you are planning to create multiple stand-alone application servers for federation later to the same cell, make sure you select a unique node name for each application server.

7. Choose whether to enable administrative security. If you enable security here, you will be asked for a user ID and password that will be added to a file-based user registry with the Administrative role. Click **Next**.
8. The wizard will present a list of TCP/IP ports for use by the application server, as in Figure 3-14. If you already have existing profiles on the system (within this installation), this will be taken into account when the wizard selects the port assignments, but you should verify that these ports will be unique on the system.

The values in the following fields define the ports for the application server and do not conflict with other profiles in this installation. Another installation of WebSphere Application Server or other programs might use the same ports. To avoid run-time port conflicts, verify that each port value is unique.

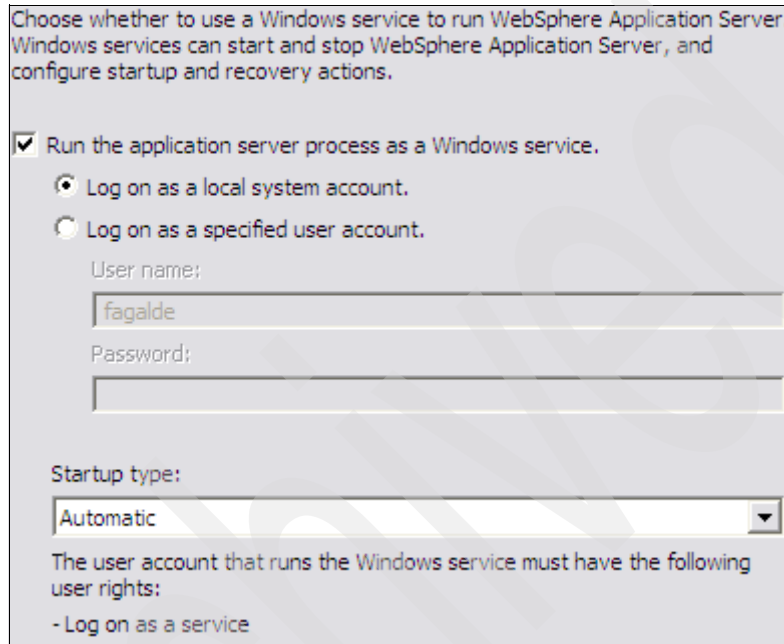
Default Port Values	Recommended Port Values
Administrative console port (Default 9060):	9061
Administrative console secure port (Default 9043):	9044
HTTP transport port (Default 9080):	9080
HTTPS transport port (Default 9443):	9443
Bootstrap port (Default 2809):	2809
SIP port (Default 5060):	5060
SIP secure port (Default 5061):	5061
SOAP connector port (Default 8880):	8880

Figure 3-14 Creating an application server profile: Select ports

Note two ports: You might want to note the following ports for later use.

- ▶ *SOAP connector port:* If you plan to federate this node to a deployment manager later using the deployment manager administrator console, you will need to know this port number. This is also the port you will connect to when using the **wsadmin** administration scripting interface.
- ▶ *Administrative console port:* You will need to know this port in order to access the administrative console. When you turn on security, you will need to know the *Administrative console secure port*.

9. On Windows systems, you have the option of running the application server as a service. This provides you a simple way of automatically starting the application server when the system starts. If you would like to run the process as a Windows service, check the box and enter the values for the logon and startup type, as shown in Figure 3-15.



Choose whether to use a Windows service to run WebSphere Application Server. Windows services can start and stop WebSphere Application Server, and configure startup and recovery actions.

☒ Run the application server process as a Windows service.

☐ Log on as a local system account.

☐ Log on as a specified user account.

User name:

Password:

Startup type:

The user account that runs the Windows service must have the following user rights:

- Log on as a service

Figure 3-15 Creating an application server profile: Run as a service

Note that the window lists the user rights the user ID you select needs to have. If the user ID does not have these rights, the wizard will automatically add them.

Click **Next**.

10. The wizard will allow you to create an optional Web server definition, as in Figure 3-16. Web server definitions define an external Web server to WebSphere Application Server. This allows you to manage Web server plug-in configuration files for the Web server and in some cases to manage the Web server. If you have not installed a Web server or wish to do this later, you can easily do this from the administrative console.

Optionally create a Web server definition if you use a Web server to route requests for dynamic content to the application server. Alternatively, you can create a Web server definition from the administrative console or a script that is generated during Web server plug-ins installation.

☒ Create a Web server definition

Web server type:
IBM HTTP Server

Web server operating system:
Windows

Web server name:
webserver1

Web server host name or IP address:
kcgg1d7

Web server port (Default 80):
80

Figure 3-16 Creating an application server profile: Creating a Web server definition.

11. Review the options you have chosen and click **Next** to create the profile. See Figure 3-17.

Review the information in the summary for correctness. If the information is correct, click **Create** to start creating a new profile. Click **Back** to change values on the previous panels.

Application server environment to create: Application server
Location: C:\myWAS61Profiles\appSrvrProfiles\AppSrvProfile1
Disk space required: 200 MB

Profile name: AppSrvProfile1
Make this profile the default: False

Node name: was61NodeAppSrvPrf1
Host name: kcgg1d7

Figure 3-17 Creating an application server profile: Finish

This final window indicates the success or failure of the profile creation.

If you have errors, check the log at:

`<was_home>/logs/manageprofiles/<profile_name>_create.log`

Note that you will have to click **Finish** on the screen to unlock the log.

You will also find logs for individual actions stored in:

`<profile_home>/logs`

12. Click **Finish** to close the wizard and start the First Steps application. See Figure 3-18 on page 74.

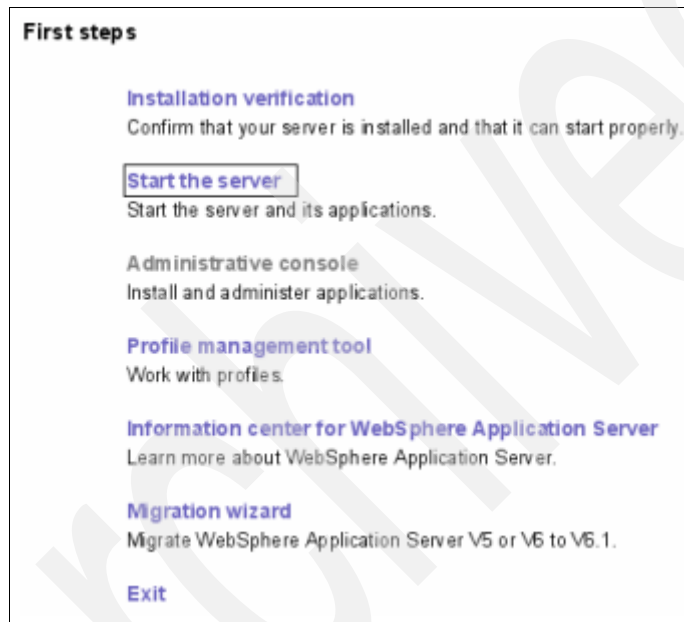


Figure 3-18 Application server First Steps menu

Check your results

If the creation was successful, do the following to familiarize yourself with the profile and how to use it:

1. View the directory structure and find the new profile. In this IBM Redbook, we refer to this directory as `<profile_home>`. This is where you will find, among other things, the config directory containing the application server configuration files, the bin directory for entering commands, and the logs directory where information is recorded.
2. Verify the installation. You can do this directly from the First Steps menu. This process will start the application server and verify the proper operation of the

Web and EJB containers. Messages are displayed on the First Steps window and logged in the following places:

- `<profile_home>/logs/server1/startServer.log`
- `<profile_home>/logs/server1/SystemOut.log`

3. Start the server. If you ran the installation verification, the server should already be started. You can check it using the following commands:

```
cd <profile_home>\bin
serverStatus -all
```

If the server status is not started, then start it from the First Steps menu or with the following commands:

```
cd <profile_home>\bin
startServer server1
```

4. Open the administrative console, either by selecting the option in the First Steps window, or by accessing its URL from a Web browser:

`http://<appserver_host>:<admin_console_port>/ibm/console`

Here is a sample URL:

`http://localhost:9061/ibm/console/`

The administrative console port of 9061 was selected during the Profile Management Tool (see Figure 3-14 on page 71).

Click the **Log in** button. If you did not enable security, you do not have to enter a user name. If you choose to enter a name, it can be any name. It is used to track changes you make from the console. If you enabled administrative security, enter the user ID and password you specified.

5. Display the configuration from the console. See Figure 3-19. You should be able to see the following items from the administrative console:
 - a. Application servers

Select **Servers** → **Application servers**. You should see server1. To see the configuration of this server, click the name in the list.

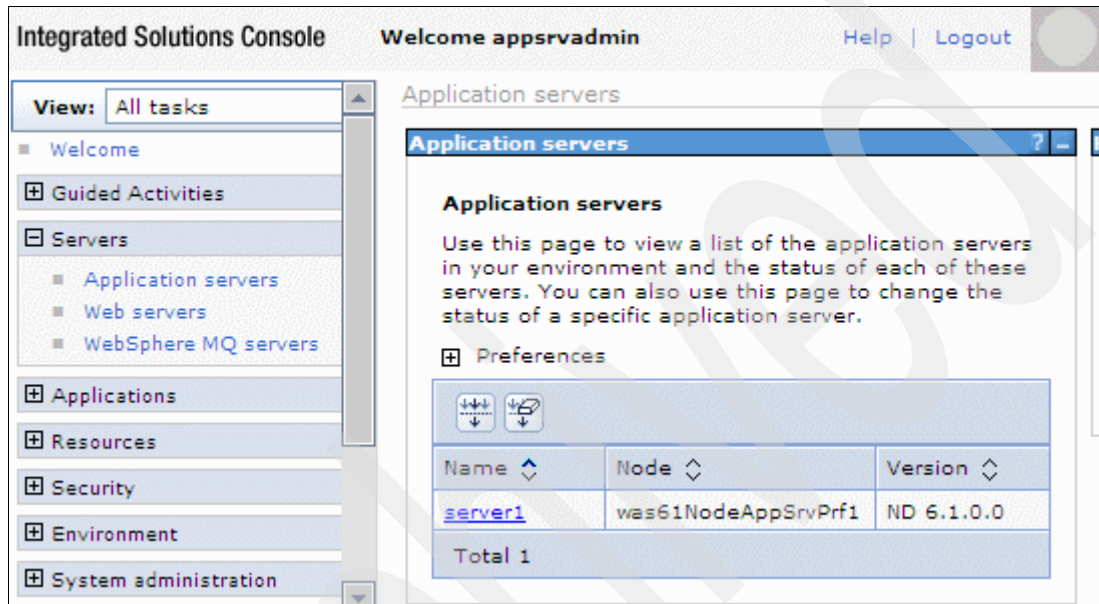


Figure 3-19 Application server defined by the application server profile

- b. Enterprise applications

Select **Applications** → **Enterprise Applications**. See Figure 3-20. You should see a list of applications. These are the WebSphere sample applications.

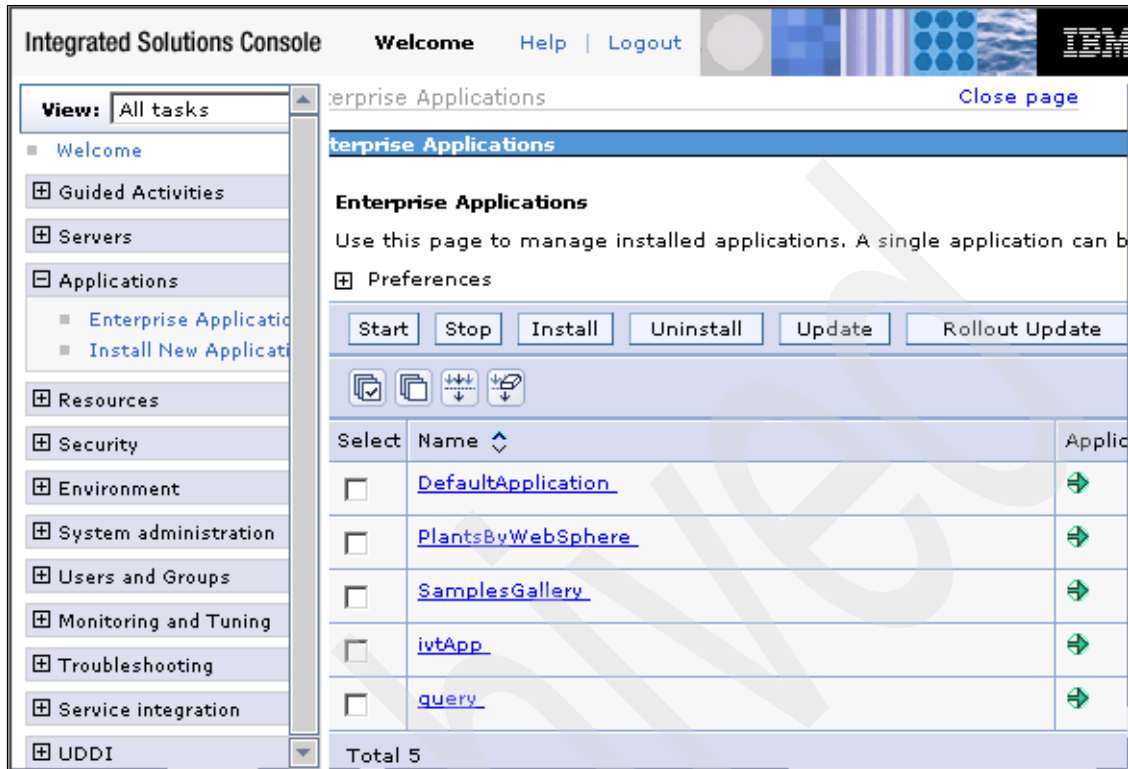


Figure 3-20 Applications installed on server1

Note: Although you cannot display the cell and node from the administrative console, they do exist. You will see this later as you begin to configure resources and choose a scope. You can also see them in the `<profile_home>/config` directory structure.

6. Stop the application server. You can do this from the First Steps menu, or better yet, use the **stopServer** command:

```
cd <profile_home>\bin
stopServer server1
```

On a UNIX system, use the following command:

```
cd <profile_home>/bin
stopServer.sh server1
```

3.3.3 Creating a cell profile

Table 3-3 shows a summary of the options you have during a cell profile creation. Using this option actually creates two distinct profiles, a deployment manager profile and an application server profile. The application server profile is federated to the cell. The options you see are a reflection of the options you would see if you were creating the individual profiles versus a cell. The Profile Management Tool windows give you basically the same options that you would see if you created a deployment manager, then an application server.

Table 3-3 Cell profile options

Typical	Advanced
The administrative console and default application are deployed by default. The sample applications are not deployed.	You have the option to deploy the administrative console (recommended), the default application, and the sample applications (if installed).
The profile name for the deployment manager is Dmgrxx by default, where xx is 01 for the first deployment manager profile and increments for each one created. The profile is stored in <was_home>/profiles/Dmgrxx.	You can specify the profile name and its location.
The profile name for the federated application server and node is AppSrvxx by default, where xx is 01 for the first application server profile and increments for each one created. The profile is stored in <was_home>/profiles/AppSrvxx.	You can specify the profile name and its location.
Neither profile is made the default profile.	You can choose to make the deployment manager profile the default profile.
The cell name is <host>Cellxx. The node name for the deployment manager is <host>CellManagerxx. The node name for the application server is <host>Nodexx. The host name is prefilled in with your system's DNS host name.	You can specify the cell name, the host name, and the profile names for both profiles.
You can enable administrative security (yes or no). If you select yes, you will be asked to specify a user name and password that will be given administrative authority.	
TCP/IP ports will default to a set of ports not used by any profiles in this WebSphere installation instance.	You can use the recommended ports for each profile (unique to the installation), use the basic defaults, or select port numbers manually.
(Windows) The deployment manager will be run as a service.	(Windows) You can choose whether the deployment manager will run as a service.

Typical	Advanced
Does not create a Web server definition.	Allows you to define an external Web server to the configuration.

3.3.4 Creating a custom profile

A custom profile defines an empty node on a system. The purpose of this profile is to define a node on a system to be federated to a cell for central management.

As you create the profile, you will have the option to federate the node to a cell during the wizard, or to simply create the profile for later federation. Before you can federate the custom profile to a cell, you will need to have a running deployment manager.

Note: With other profiles, you have the option of registering the processes as Windows services. This does not appear as an option when you create a custom profile. If you want to register the node agent as a Windows service later, see 3.6.3, “Enabling process restart on failure” on page 130.

Table 3-4 shows a summary of the options you have during profile creation for a custom node.

Table 3-4 Custom profile options

Typical	Advanced
The profile name is Customxx. The profile is stored in <was_home>/profiles/Customxx. By default, it is not considered the default profile.	You can specify profile name and location. You can also specify if you want this to be the default profile.
The node name is <host>Nodexx. The host name is prefilled in with your system's DNS host name.	You can specify node name and host name.
You can opt to federate the node later, or during the profile creation process. If you want to do it now, you have to specify the deployment manager host and SOAP port (by default, localhost:8879). If security is enabled on the deployment manager, you will need to specify a user ID and password.	
TCP/IP ports will default to a set of ports not used by any profiles in this WebSphere installation instance.	You can use the recommended ports for each profile (unique to the installation), use the basic defaults, or select port numbers manually.

This section takes you through the steps of creating a custom profile.

1. Start the Profile Management Tool. Click **Next** on the Welcome page.
2. Select the **Custom profile** option. Click **Next**.
3. Select the kind of creation process you want to run, typical or advanced. Click **Next**.

If Typical is selected, then you will be sent directly to the option to federate (Figure 3-23 on page 82).

If Advanced is selected, you will see the next step.

4. Enter a unique name for the profile or accept the default. The profile name will become the directory name for the profile files. See Figure 3-21.

Click the box if you want this directory to be the default profile for receiving commands. Click **Next**.

Specify a profile name and directory path to contain the files for the run-time environment, such as commands, configuration files, and log files. Click **Browse** to select a different directory.

Profile name:

Profile directory:

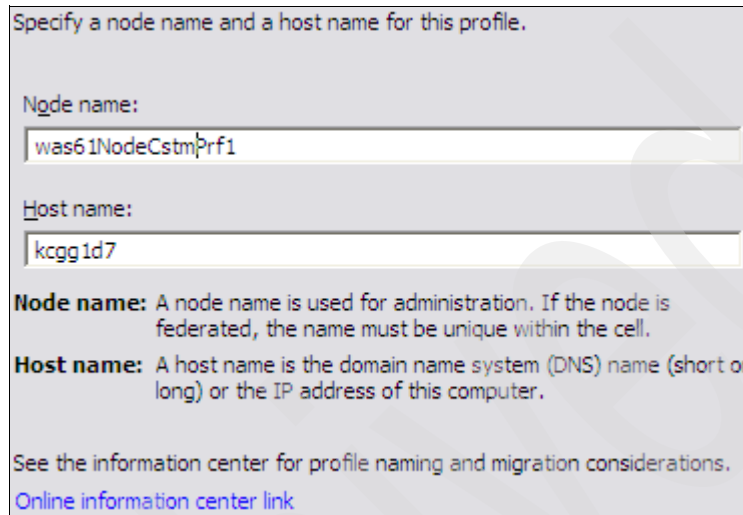
☐ Make this profile the default.

Each installation of WebSphere Application Server always has one default profile. Commands that run without referring to a specific profile use the default profile. Select this option to make this profile the new default

Important: Deleting the directory a profile is in does not completely delete the profile. Use the **manageprofiles** command to completely delete a profile.

Figure 3-21 Creating a Custom profile: Enter name and location

5. Enter the new node name and the system host name. See Figure 3-22. The node name defaults to the host name of your system. The wizard recognizes if there are existing nodes in the installation and takes this into account when creating the default node name. Click **Next**.



Specify a node name and a host name for this profile.

Node name:

Host name:

Node name: A node name is used for administration. If the node is federated, the name must be unique within the cell.

Host name: A host name is the domain name system (DNS) name (short or long) or the IP address of this computer.

See the information center for profile naming and migration considerations.
[Online information center link](#)

Figure 3-22 Creating a custom profile: Enter host, and node names

6. If you would like to federate, or add, the new node defined by the profile to a cell as part of the wizard process, leave the Federate this node later box unchecked and enter the host name and SOAP connector port (Figure 3-8 on page 63) for the deployment manager. See Figure 3-23.

Note: If you choose to federate now, make sure the deployment manager is started.

Specify the host name or IP address and the SOAP port number for an existing deployment manager. Federation can occur only if the deployment manager is running.

Deployment manager host name or IP address:
localhost

Deployment manager SOAP port number (Default 8879):
8879

Deployment manager authentication
Provide a user name and password that can be authenticated, if administrative security is enabled on the deployment manager.

User name:
Password:

☐ Federate this node later.
You must federate this node later using the **addNode** command if the deployment manager:
- is not running
- has the SOAP connector disabled

Figure 3-23 Creating a custom profile: Federate now or later

7. Review the options you have chosen. See Figure 3-24.

Review the information in the summary for correctness. If the information is correct, click **Create** to start creating a new profile. Click **Back** to change values on the previous panels.

Application server environment to create: Custom profile
Location: C:\myWAS61Profiles\cstmProfiles\CstmProfile1
Disk space required: 10 MB

Profile name: CstmProfile1
Make this profile the default: False

Node name: was61NodeCstmPrf1
Host name: kcg1d7

Federate to deployment manager: localhost:8879

Figure 3-24 Creating a custom profile: Summary

Click **Next** to create the profile.

This final window indicates the success or failure of the Custom profile creation.

If you have errors, check the log at:

`<was_home>/logs/manageprofiles/<profile_name>_create.log`

Note that you will have to click **Finish** on the window to unlock the log.

You will also find logs for individual actions stored in:

`<profile_home>/logs`

8. After the wizard has finished, you will be presented with a screen containing messages indicating the success or failure of the process. And you can launch First Steps if you want or even create another profile. If you have errors, check the log at:

`<was_home>/logs/manageprofiles/<profile_name>_create.log`

Note that you will have to click **Finish** on the window to unlock the log.

You will also find logs for individual actions stored in:

`<profile_home>/logs`

9. Click **Finish** to close the wizard and start the First Steps application if you want. See Figure 3-25 on page 84.

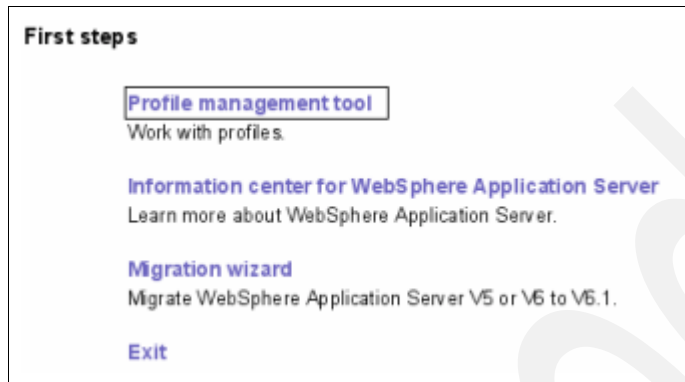


Figure 3-25 Custom profile First Steps window

Checking your results

If the creation was successful, do the following to familiarize yourself with the profile and how to use it:

1. View the `<profile_home>` directory structure and find the new profile. This is where you will find, among other things, the config directory containing the node configuration files.
2. If you federated the custom profile, open the deployment manager administrative console and view the node and node agent:
 - Select **System Administration** → **Nodes**. You should see the new node.
 - Select **System Administration** → **Node agents**. You should see the new node agent.
 - Select **System Administration** → **Cells**. Click the **Topology** tab and expand the view. From here, you can see a tree diagram of the cell, as in Figure 3-26 on page 85.



Figure 3-26 Topology view of a cell

3. The federation process creates a node agent for the new node, federates it to the cell, and starts the node agent.

You can stop the new node agent from the console or with the following commands on the node system:

```
cd <profile_home>\bin
stopNode
```

While you can restart a node agent from the administrative console, you cannot start a node that has been stopped. To start the new node agent, use the following commands on the node system.

```
cd <profile_home>\bin
startNode
```

If you have not federated the node, you will not be able to start it yet. Proceed to 3.3.5, “Federating a custom node to a cell” on page 86. Otherwise, you can continue by defining an application server on the new node. To do this, see 3.3.6, “Creating a new application server on an existing node” on page 88.

3.3.5 Federating a custom node to a cell

Note: You only have to do this if you created a custom profile and chose *not* to federate it at the time. This requires that you have a deployment manager profile and that the deployment manager is up and running.

An custom profile is used to define a node that can be added to a cell. To federate the node to the cell, do the following:

1. Start the deployment manager.
2. Open a command window on the system where you created the custom profile for the new node. Switch to the `<profile_home>/bin` directory (for example, `cd C:\myWAS61Profiles\cstmProfiles\CstmProfile1`).
3. Run the **addNode** command. Here you need the host name of the deployment manager and the SOAP connector address (see Figure 3-7 on page 62 and Figure 3-8 on page 63):

```
addNode <dmgrhost> <dmgr_soap_port>
```

Example 3-1 shows an example of using the **addNode** command on a Windows system to add Node06 to the deployment manager using 8879 as the SOAP connector address.

Example 3-1 addNode command

```
C:\WebSphere\ND\profiles\Node06\bin>addnode localhost 8879
ADMU0116I: Tool information is being logged in file
          c:\WebSphere\ND\profiles\Node06\logs\addNode.log
ADMU0128I: Starting tool with the Node06 profile
CWPKI0308I: Adding signer alias "dummyclientsigner" to local keystore
            "ClientDefaultTrustStore" with the following SHA digest:
            0B:3F:C9:E0:70:54:58:F7:FD:81:80:70:83:A6:D0:92:38:7A:54:CD
CWPKI0308I: Adding signer alias "dummyserversigner" to local keystore
            "ClientDefaultTrustStore" with the following SHA digest:
            FB:38:FE:E6:CF:89:BA:01:67:8F:C2:30:74:84:E2:40:2C:B4:B5:65
CWPKI0308I: Adding signer alias "default_2" to local keystore
            "ClientDefaultTrustStore" with the following SHA digest:
            CC:60:A6:33:99:B0:D9:34:B2:6A:89:5F:A7:5F:C8:C1:9E:CC:8C:2A
CWPKI0308I: Adding signer alias "default_1" to local keystore
            "ClientDefaultTrustStore" with the following SHA digest:
            20:83:69:46:D9:B9:95:51:00:99:3C:D9:3B:EF:E4:1B:C1:9A:C1:84
CWPKI0308I: Adding signer alias "default" to local keystore
            "ClientDefaultTrustStore" with the following SHA digest:
            DA:29:33:E3:61:67:91:79:B4:54:EA:95:04:D0:47:8A:14:70:DF:90
ADMU0001I: Begin federation of node Node06 with Deployment Manager at
```

```

localhost:8879.
ADMU0001I: Begin federation of node Node06 with Deployment Manager at
localhost:8879.
ADMU0009I: Successfully connected to Deployment Manager Server: localhost:8879
ADMU0507I: No servers found in configuration under:
c:\WebSphere\ND\profiles\Node06\config/cells/kadw028Node04Cell/nodes
Node06/servers
ADMU2010I: Stopping all server processes for node Node06
ADMU0024I: Deleting the old backup directory.
ADMU0015I: Backing up the original cell repository.
ADMU0012I: Creating Node Agent configuration for node: Node06
ADMU0014I: Adding node Node06 configuration to cell: kadw028Cell01
ADMU0016I: Synchronizing configuration between node and cell.
ADMU0018I: Launching Node Agent process for node: Node06
ADMU0020I: Reading configuration for Node Agent process: nodeagent
ADMU0022I: Node Agent launched. Waiting for initialization status.
ADMU0030I: Node Agent initialization completed successfully. Process id is:
2120
ADMU0505I: Servers found in configuration:
ADMU0506I: Server name: nodeagent
ADMU9990I:
ADMU0300I: The node Node06 was successfully added to the kadw028Cell01 cell.
ADMU9990I:
ADMU0306I: Note:
ADMU0302I: Any cell-level documents from the standalone kadw028Cell01
configuration have not been migrated to the new cell.
ADMU0307I: You might want to:
ADMU0303I: Update the configuration on the kadw028Cell01 Deployment Manager
with values from the old cell-level documents.
ADMU9990I:
ADMU0306I: Note:
ADMU0304I: Because -includeapps was not specified, applications installed on
the standalone node were not installed on the new cell.
ADMU0307I: You might want to:
ADMU0305I: Install applications onto the kadw028Cell01 cell using wsadmin
$AdminApp or the Administrative Console.
ADMU9990I:
ADMU0003I: Node Node06 has been successfully federated.

C:\WebSphere\ND\profiles\Node06\bin>

```

-
4. Open the deployment manager administrative console and view the node and node agent:
 - Select **System Administration** → **Nodes**. You should see the new node.

- Select **System Administration** → **Node agents**. You should see the new node agent and its status. It should be started. If not, check the status from a command window on the custom node system:

```
cd <profile_home>\bin  
serverStatus -all
```

If you find that it is not started, start it with this command:

```
cd <profile_home>\bin  
startNode
```

3.3.6 Creating a new application server on an existing node

The custom profile does not automatically give you an application server. You can follow these steps to create a new server once the custom profile has been federated to a cell.

Note: This topic outlines the procedure to create and start an application server. For detailed information about creating and customizing application servers, see 4.4, “Working with application servers” on page 170.

If you plan to use clustering, you can create application servers when you create the cluster. For information about working with clusters, see 4.6, “Working with clusters” on page 222.

1. Ensure the custom profile node agent is started.
2. Open the deployment manager administrative console.
3. Select **Servers** → **Application Servers**
4. Click **New**.
5. Select the custom profile node and enter a new name for the server (Figure 3-27). Click **Next**.

Create New Server

→ **Step 1: Select a node**

Step 2: Select a server template

Step 3: Specify server specific properties

Step 4: Confirm new server

Select a node

Select the node that corresponds to the server you wish to create.

Select node
CustomFedLaterNode

* Server name
ServerX

Next Cancel

Figure 3-27 Creating a new server: Enter a node and name

6. Select a template to use as a basis for the new application server configuration. See Figure 3-28.

Use this page to create a new application server.

Step 1: Select a node

→ **Step 2: Select a server template**

Step 3: Specify server specific properties

Step 4: Confirm new server

Select a server template

Select	Name	Type	Specifies a description of an applic
☐	DeveloperServer	System	This template is optimized to perform development uses.
☐	Server2Template	User Defined	Development-Restrict-Single
☒	default	System	The WebSphere Default Server Tem

Previous Next Cancel

Figure 3-28 Creating a new server: Select a template

The DeveloperServer and default templates have been created for you. The default template is used to create a typical server for production.

New in V6.1: The DeveloperServer template is used to create a server optimized for development. It turns off PMI and sets the JVM into a mode that disables class verification and allows it to startup faster via the **-Xquickstart** command. Note that it does not enable the "developmentMode" configuration property (Run in development mode setting on the application server window). If you would like to set this to speed up the application server startup, you will need to configure it after server creation using the administrative console.

You can also create templates based on existing servers.

If you have not previously set up a template based on an existing application server, select the default template. Click **Next**.

7. Each application server on a node must have unique ports assigned. The next window gives you the option of having unique ports generated for this application server, as opposed to the default set. Click **Next**. See Figure 3-29 on page 90.



Figure 3-29 Creating a new server: Generate unique ports

8. The last window summarizes your choices. See Figure 3-30. Click **Finish** to create the profile.

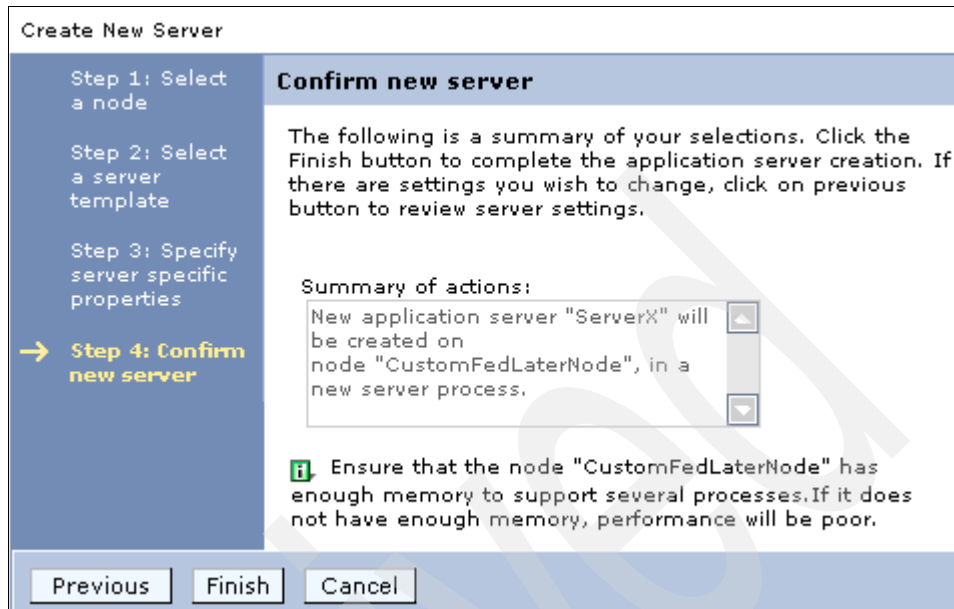


Figure 3-30 Creating a new server: Summary and finish

9. In the messages box, click **Save** to save the changes to the master configuration.
10. Start the application server from the administrative console.
 - Select **Servers** → **Application Servers**.
 - Check the box to the left of the server and click **Start**.

Note: WebSphere Application Server provides sample applications that you can use to familiarize yourself with WebSphere applications. These samples can be installed (optional) when you create an application server profile. If you create an application server from the administrative tools, you will not get the samples installed automatically. For information about the samples available and how to install them, see the Accessing the Samples topic under *Learn about WebSphere Applications* in the Information Center.

3.3.7 Federating an application server profile to a cell

If you created an application server profile and now want to add the node and server to the cell, do the following:

1. Start the application server.

2. Start the deployment manager.
3. Open the deployment manager administrative console.
4. Select **System Administration** → **Nodes**
5. Click **Add Node**.
6. Select **Managed node** and click **Next**.
7. Enter the host name and SOAP connector port specified when you created the application server profile. See Figure 3-13 on page 70 and Figure 3-14 on page 71.

If you want to keep the sample applications and any other applications you have installed, check the **Include applications** box. If this is a newly created application server profile, it will contain the sample applications, so be sure to check this box if you want to keep the samples.

If you have created a service integration bus on the server, you can opt to have it included in the managed server as well. By default, you do not have a service integration bus in a newly created application profile. If you have created a bus, and choose to include it, the name must be unique in the cell.

See Figure 3-32 on page 94.

Add Managed Node

Specify a remote WebSphere Application Server instance to add into the cell. The remote server must be running.

Node connection
Host
carlavm2
JMX connector type
SOAP
JMX connector port
8882

Options
☒ Include applications
☐ Include buses

Starting port
☒ Use default
☐ Specify
Port number

Figure 3-31 Adding a standalone application profile to a cell

Click **OK**.

8. If the node is a Windows node, you have the opportunity to register the new node agent as a Windows service, as shown in Figure 3-32. Make your selection and click **OK**.

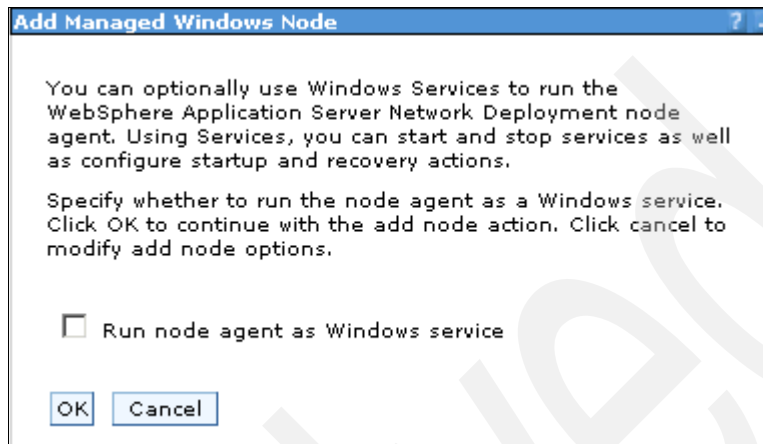


Figure 3-32 Run a node agent as a Windows service

The federation process stops the application server. It creates a new node agent for the node, and adds the node to the cell. The application server becomes a managed server in the cell. It then starts the node agent, but not the server.

9. You can now display the new node, node agent, and application server from the console. You can also start the server from the console.

At the completion of the process:

- ▶ The profile directory for the application server still exists and is used for the application server.
- ▶ The old cell name for the application server has been replaced with a profile directory with the cell name of the deployment manager.

```
<profile_home>/config/cells/<dmgr_cellname>/
```

- ▶ A new entry in the deployment manager profile directory has been added for the new node.

```
<dmgr_profile_home>/config/cells/<dmgr_cellname>/nodes/<federated node>
```

- ▶ An entry for each node in the cell is added to the application server profile configuration. Each node entry contains the serverindex.xml file for the node.

```
<profile_home>/config/cells/<dmgr_cellname>/nodes/<federated node>
```

In turn, an entry for the new node is added to the nodes directory for each node in the cell with a serverindex.xml entry for the new node.

3.4 Creating profiles on z/OS systems

Configuring a WebSphere Application Server for z/OS consists of setting up the configuration directory for the environment and making any required changes to the z/OS target system that pertain to the particular application serving environment. Configuring these application serving environments after product installation requires a fair amount of planning and coordination. For example, when defining multiple deployment managers or application servers on a single machine or LPAR, you need to ensure that the ports and names you select for each are unique and the z/OS environment variables, generated jobs, and so on, were all set up properly. We strongly recommend you spend time planning the installation and if possible, practice by configuring a stand-alone application server using the default options.

For more information about planning for installation, *WebSphere Application Server V6.1: Planning and Design*, SG24-7305

There are three main ways to create profiles on z/OS:

- Using the WebSphere Application Server for z/OS Profile Management Tool (zPMT) available with the Application Server Toolkit (new with V6.1)

zPMT is a dialog tool that runs in the Application Server Toolkit. It is an Eclipse plug-in that allows you to do the initial setup of WebSphere Application Server for z/OS cells and nodes. It provides the same functionality as the ISPF dialogs plus additional features to help you.

The zPMT itself does not create the cells and nodes; however, it creates batch jobs, scripts, and data files that you can use to perform WebSphere Application Server for z/OS customization tasks. These jobs, scripts, and data files form a customization definition on your workstation, which is then uploaded to z/OS where you submit the jobs. The zPMT is used to create profiles and cannot be used to perform functions like delete or list profiles. Only the manageprofiles command-line script interface can perform these functions.

Note: In this IBM Redbook, we will only talk about creating profiles using zPMT and the manageprofiles command-line interface since the ISPF dialogs are deprecated and will be removed in a future release.

- Using the ISPF dialogs

As mentioned above, the ISPF customization dialogs are deprecated and will be removed in a future release, although you can still set up your profile through ISPF dialogs in V6.1. On V6.1, the windows have the same interface as on V6.0 with only few changes to accommodate the new V6.1 functions.

- Using the manageprofiles script interface

Review the documentation:

The WebSphere Application Server information center contains planning topics for each WebSphere Application Server package that is tailored to each platform. This section will give you a high level look at the planning tasks you will need to perform.

If you are planning a WebSphere Application Server for z/OS environment, we strongly suggest that you review the following:

For more information and examples of defining a naming convention for WebSphere for z/OS, see the following:

- *WebSphere z/OS V6 -- WSC Sample ND Configuration*, found at:

<http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100653>

To go with this document, a spreadsheet has been developed that will help you create and document your names. This spreadsheet can be downloaded from this same URL. Many of the values you will be asked to use in the zPMT tool can be planned using this spreadsheet.

For information about differences you will find in V6.1:

- *WebSphere for z/OS V6.1 - New Things Encountered During Configuration*, found at:

<http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100781>

Creating a cell profile

In this section, we are going to explain how to use the zPMT to generate the jobs, scripts, and data files you need to generate a cell consisting of a deployment manager and a federated application server.

Using zPMT requires that you have the Application Server Toolkit for V6.1 installed. For more information about how to download and install the Application Server Toolkit, see the following URL:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/tins_astinstall.html

Through the zPMT, you have the following options:

- ▶ Create an application server profile
Generates the customization jobs to create a z/OS stand alone application server.
- ▶ Create a cell profile (deployment manager profile and federated application server)
Generates the customization jobs to create a z/OS deployment manager and a federated node that contains an application server. This option only exists in the zPMT.
- ▶ Create a deployment manager profile
Generates the customization jobs to create a z/OS deployment manager cell without a federated application server.
- ▶ Federate an application server to a cell
Generates the customization jobs to federate an existing stand alone z/OS application server into an existing network deployment cell.
- ▶ Create a managed (custom) node
Generates the customization jobs to create a z/OS managed node and federate it into an existing network deployment cell.

All these options are also available through the ISPF Dialogs except the *z/OS cell (deploy manager and application server)* that is unique to zPMT.

In this section, we are going to show how to use zPMT to create a z/OS cell (deploy manager and application server). We chose this option because it will basically go through all the steps you will find on the other options. Figure 3-33 on page 98 illustrates the flow you will go through in order to generate the jobs for this option.

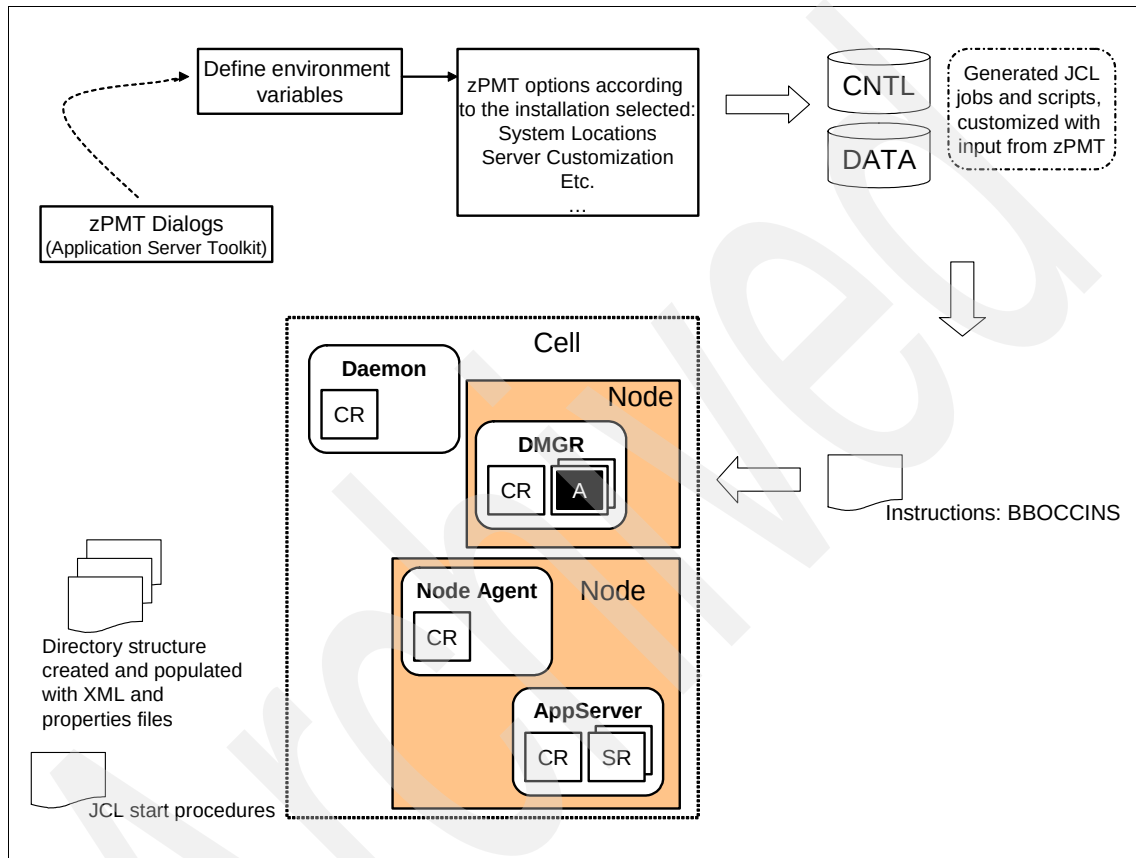


Figure 3-33 zPMT basic flow

To create the profile, do the following:

1. Open the Application Server Toolkit and select **Window** → **Preferences**.
2. From the Preferences window, expand **Server** category and select **WebSphere for z/OS**.
3. Click on the **Create** button (on the top right corner).

The Welcome window for the Profile Management Tool will open (Figure 3-34 on page 99). Note that this window contains a link to the WebSphere Application Server V6.1 Information Center.

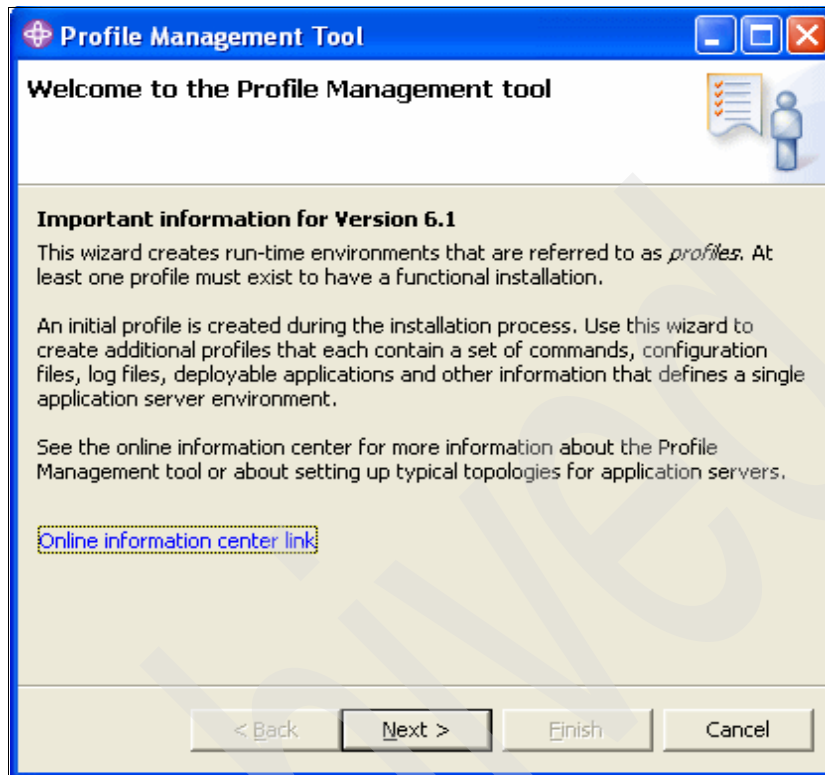


Figure 3-34 Creating a profile: zPMT welcome window

Click **Next**.

4. Select the appropriate configuration in the next pane (Figure 3-35 on page 100). In this example, we will be creating a cell to illustrate the zPMT utilization.

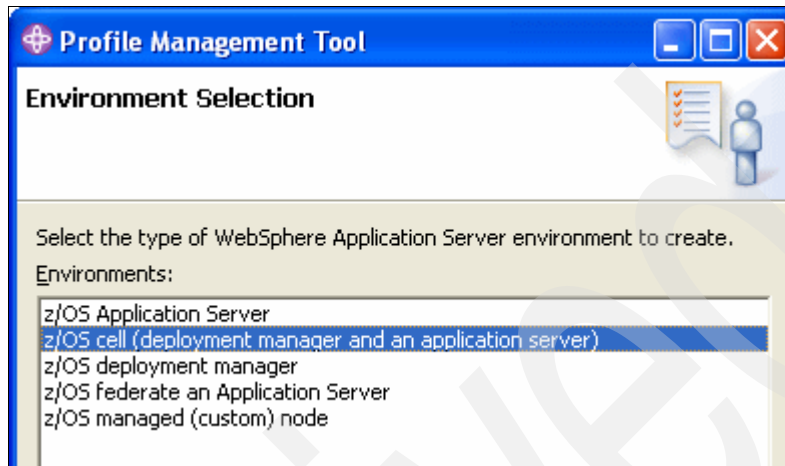


Figure 3-35 Creating a profile: Environment selection window

Select **z/OS cell (deployment manager and application server)** and click **Next**.

5. The next window (Figure 3-36 on page 101) contains the following fields:
 - Customization definition name: Used to specify the customization profile you are about to create. This name is not transported to your host system.
 - Customization definition directory: The location on your workstation where the CNTL and DATA files will be stored, and from which they will be uploaded to your host system.
 - Response file path name: Allows you to specify a saved file with values from a previously created configuration. Doing this populates the fields throughout the windows with the values that are contained in the response file. This field is optional. Since it is the first time you will be creating a profile, you probably do not have this file. A response file is written each time a z/OS customization definition is created and its name is the customization definition name itself + *.responseFile* created under the root directory for the customization definition. Normally, you should specify a response file from a customization definition of the same type as you are about to define. However, a response file from a similar customization type can be used to pre-load most of the default values.

Profile Management Tool

Customization name and location
z/OS cell (deployment manager and an application server)

Specify a customization definition name and directory path to contain the batch jobs and instructions that will be used to define the selected z/OS run-time environment.

Customization definition name:
WSC61DMGRFAS

Customization definition directory:
C:\Documents and Settings\kleinubing\AST\workspace\metadata\plugins\com.ibm
Browse...

Specify the full pathname of the response file that contains the default values. When this value is specified, the input fields in the wizard will be pre-loaded with the values in the response file.

Response file pathname (optional):
C:\Documents and Settings\kleinubing\AST\workspace\metadata\plugins\com.ibm
Browse...

Note: WebSphere for z/OS uses the Profile Management Tool to create customization jobs and instructions that are used to configure a z/OS runtime environment. However, a z/OS runtime profile is not created until these customization jobs are successfully run on a z/OS system.

< Back Next > Finish Cancel

Figure 3-36 Creating a profile: customization name and location window

After completing the required fields, click **Next**.

6. The next window (Figure 3-37 on page 102) asks you to specify a high level qualifier for the target z/OS data sets that will contain the generated jobs and instructions.

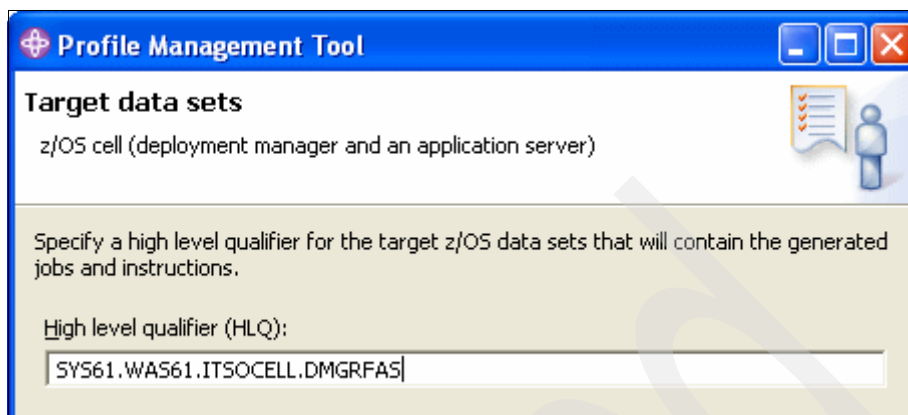


Figure 3-37 Creating a profile: target data sets window

The high level qualifier can be composed of multiple qualifiers up to 39 characters. When a customization profile is uploaded on the target z/OS system, the generated jobs and files are written on a pair of data sets. The same data sets can be reused for a future installation; however, we strongly recommend you create a new pair of data sets for every new profile installation. A good planning and naming convention is crucial when defining this type of information. As a best practice, try to set the high level qualifier according to the version and release of WebSphere Application Server for z/OS, the task you are performing, and the cell (and, in some cases, the node name) you are configuring.

For example, on a stand-alone installation for a cell named ITSOCCELL, you could use the following qualifier:

```
SYSPLEX1.WAS61.ITSOCCELL.APPSERV
```

In this case, the following data sets will be created when the customization profile is uploaded to the target z/OS system:

```
SYSPLEX1.WAS61.ITSOCCELL.APPSERV.CNTL
```

```
SYSPLEX1.WAS61.ITSOCCELL.APPSERV.DATA
```

The CNTL data set is a partitioned data set with a fixed block 80-byte records that keeps the customization jobs. The DATA data set is a partitioned data set as well, but with variable length data to contain the other customization data.

Note: Once the customization profile is created, the data set names cannot be changed, since all jobs are based on these data set names.

After completing the HLQ field, click **Next**.

7. The next window (Figure 3-38 on page 103) contains the fields to configure common groups and users.

The screenshot shows a window titled "Profile Management Tool" with a subtitle "Configure common groups and users" and a description "z/OS cell (deployment manager and an application server)". The window contains several sections for configuring WebSphere Application Server information:

- WebSphere Application Server configuration group information:**
 - Group: WSCFG1
 - GID: 2500
- WebSphere Application Server file system owner information:**
 - User ID: WSOWNER
 - UID: 2405
- WebSphere Application Server servant group information:**
 - Group: WSSR1
 - GID: 2501
- WebSphere Application Server local user group information:**
 - Group: WSCLGP
 - GID: 2502
- WebSphere Application Server user ID home directory:**
 - /var/WebSphere/home

At the bottom, there are four buttons: "< Back", "Next >", "Finish", and "Cancel".

Figure 3-38 Creating a profile: Configure common groups and users window

You will find five main sections that you need to fill out on this window:

- WebSphere Application Server Configuration Group Information: Used to specify the group name for the WebSphere Application Server administrator user ID and all server user IDs.
- WebSphere Application Server file system owner Information: Specify the user ID that owns the file system.
- WebSphere Application Server Servant Group Information: Used to connect all servant user IDs to this group. You can use it to assign subsystem permissions, such as DB2 authorizations, to all servants in the security domain.
- WebSphere Application Server Local User Group Information: Specify the local client group and unauthorized user IDs. This group provides minimal access to the cell.
- WebSphere Application Server user ID home directory: Specify a new or existing z/OS file system directory in which home directories for WebSphere Application Server for z/OS user IDs will be created by the customization process. Note that this directory does not need to be shared among z/OS systems in a WebSphere Application Server cell.

After completing the required fields, click **Next**.

8. The next window asks for information about the z/OS system.

- System name: The system name of the target z/OS system.
- Sysplex name: The sysplex name of the target z/OS system.

Note: If you are not sure of the System and Sysplex names for your target z/OS system, you can use the console command D SYMBOLS on the target z/OS system to display them.

- PROCLIB data set name: The PROCLIB data set where the WebSphere Application Server for z/OS cataloged procedures are to be added.
- WebSphere product data set high level qualifier: This name prepends your system libraries, for example, WAS61.SBBOLOAD. A multi-level high level qualifier can be specified as the WebSphere product data set high level qualifier.

Note: You are no longer required to enter a PARMLIB name. It was a required field on ISPF windows for Version 6.0.

Profile Management Tool

System Locations: Names and data set qualifier
z/OS cell (deployment manager and an application server)

System name:
SC61

Sysplex name:
WTSCPLX1

PROCLIB data set name:
SYS1.PROCLIB

WebSphere product data set high level qualifier:
WAS61

Note: A multi-level high level qualifier can be specified as the WebSphere product data set high level qualifier.

< Back Next > Finish Cancel

Figure 3-39 Creating a profile: System Locations: Names and data set qualifier window

After completing the required fields, click **Next**.

9. The next window asks for the product data set location information (see Figure 3-40). In this window, you specify the product file system directory and the data sets.

In Version 6.1, most LPA-resident modules are merged into a single member in SBBLOAD, which is loaded into common storage for each node if SBBLOAD is not in LPA. The remaining module in SBBOLPA is BBORTS61, the CTRACE support module for WebSphere Application Server Version 6.1, which should always be loaded into LPA.

Profile Management Tool

System Locations: Product data sets

z/OS cell (deployment manager and an application server)

WebSphere Application Server product file system directory:

SBBOLPA data set name:

SBBOEXEC data set name:

SBBOMSG data set name:

SBBLOAD data set name:

SBBGLOAD data set name:

SBBOLD2 data set name:

☒ Run WebSphere Application Server from STEPLIB

Figure 3-40 zPMT - System locations

Note: LPA is a major element of MVS/ESA™ virtual storage below the 16 MB line. The storage areas that make up the LPA contain all the common reentrant modules shared by the system. The LPA provides economy of real storage by sharing one copy of the modules, protection because LPA code cannot be overwritten even by key 0 programs, and reduced path length, because the modules can be branched to. The LPA is duplicated above the 16 MB line as the extended link pack area (ELPA).

Important considerations:

- The SBBOLD2 data set is the WebSphere Application Server for z/OS load module library that you installed through SMP/E. It has members that should go into the link list, or into STEPLIB. *Do not* place them in LPA.
- The SBBOEXEC data set contains the WebSphere Application Server for z/OS CLIST library.
- The SBBOMSG data set contains the WebSphere Application Server for z/OS message skeletons for language translation.
- The WebSphere Application Server product file system directory is the name of the directory where WebSphere Application Server for z/OS product files reside after installation.
- The “run WebSphere Application Server from STEPLIB” check box specifies whether to load WebSphere Application Server for z/OS load modules from STEPLIB or from the link pack area and link list. The load modules must be loaded from STEPLIB if you have another instance of WebSphere Application Server for z/OS (Version 4 or later) in the system link pack area or link list.

After completing the required fields, click **Next**.

10. The next window allows you to specify the long and short names to use for the components of the profile.

- Cell short name: Identifies the cell to z/OS facilities, such as SAF.
- Cell long name: It is the primary external identification of this WebSphere Application Server for this z/OS cell. This name identifies the cell as displayed through the administrative console.
- Deployment manager short name: It is a name that identifies the node to z/OS facilities, such as SAF.
- Deployment manager long name: It is the primary external identification of this WebSphere Application Server for the z/OS node. This name identifies the node as displayed through the administrative console.

- Deployment manager server short name: Identifies the server to z/OS facilities, such as SAF. The server short name is also used as the server JOBNAME.
- Deployment manager server long name: It is the name of the application server and the primary external identification of this WebSphere Application Server for the z/OS server. This name identifies the server as displayed through the administrative console.
- Node agent and application server nodes short name: Identifies the node to z/OS facilities, such as SAF.
- Node agent and application server nodes long name: It is the primary external identification of this WebSphere Application Server for the z/OS node. This name identifies the node as displayed through the administrative console.
- Node agent server short name: Identifies the node agent to z/OS facilities, such as SAF. The server short name is also used as the node agent JOBNAME.
- Node agent server long name: It is the name of the node agent and the primary external identification of this WebSphere Application Server for the z/OS node agent.
- Application server short name: Identifies the server to z/OS facilities, such as SAF. The server short name is also used as the server JOBNAME.
- Application server long name: It is the name of the application server and the primary external identification of this WebSphere Application Server for the z/OS server. This name identifies the server as displayed through the administrative console.
- Deployment manager cluster transition name: The WLM application environment (WLM APPLENV) name for the deployment manager. If this is a server that is converted into a clustered server, this name becomes the cluster short name. The cluster short name is the WLM APPLENV name for all servers that are of the same cluster.
- Application server cluster transition name: The WLM application environment (WLM APPLENV) name for the application server. If this is a server that is converted into a clustered server, this name becomes the cluster short name. The cluster short name is the WLM APPLENV name for all servers that are of the same cluster.

After completing the required fields, click **Next**.

11. The next window (Figure 3-41 on page 109) contains the file system information for your z/OS system. The file system can be either HFS or zFS. It is used to hold WebSphere Application Server configuration information. You

will need to fill out this window twice, once for the deployment manager file system configuration and a second time for the application server.

Profile Management Tool

System Environment: Configuration file system
z/OS cell (deployment manager and an application server)

Deployment manager configuration file system

Mount point:
\\WebSphere/V6R1/WTSCPLX1Network/dmgr

Name:
OMVS.WAS61.WTSCPLX1.WTSCPLX1.HFS

Volume, or '*' for SMS:
*

Primary allocation in cylinders: 420

Secondary allocation in cylinders: 100

File system type:

☒ Hierarchical File System (HFS)

☐ zSeries File System (ZFS)

< Back Next > Finish Cancel

Figure 3-41 zPMT - Configuration file system

- Mount point: The read/write HFS directory where application data and environment files are written. The customization process creates this mount point if it does not already exist.
- Name: The file system data set you will create and mount at the specified mount point above.

- Volume, or '*' for SMS: Specify either the DASD volume serial number to contain the above data set or '*' to let SMS select a volume. Using '*' requires that SMS automatic class selection (ACS) routines be in place to select the volume. If you do not have SMS set up to handle data set allocation automatically, list the volume explicitly.
- Primary allocation in cylinders: The initial size allocation for the configuration file system data set. In the application server, the total space needed for this data set increases with the size and number of the installed applications. The minimum suggested size is 250 cylinders (3390).
- Secondary allocation in cylinders: The size of each secondary extent. The minimum suggested size is 100 cylinders.
- File System type: Select to allocate and mount your configuration file system data set using HFS or zFS.

After completing the required fields for the deploy manager configuration file system, click **Next**.

12. Complete the required fields for the application server configuration file system and click **Next**.

13. The next window contains the following fields. Complete the required information to set up the log stream.

- Error log stream name: It is the name of the WebSphere error log stream you will create.
- Trace Parmlib member suffix: It is the value that is appended to CTIBBO to form the member name for the CTRACE parmli member.

After completing the required fields, click **Next**.

14. The next window (Figure 3-42 on page 111) allows you to enter the required information for the application server and deployment manager home directory.

Profile Management Tool

Server: Defining App_Server_Root
z/OS cell (deployment manager and an application server)

WebSphere Application Server home directory pathname

Deployment manager configuration file system mount point:
/WebSphere/V6R1/WTSCPLX1Network/dmgr

Deployment manager directory pathname relative to mount point:
DeploymentManager

Application server configuration file system mount point:
/WebSphere/V6R1/WTSCPLX1Netw/SC61

Application Server directory pathname relative to mount point:
AppServer

Admin asynch operations procedure name:
BBOW65H

Asynchronous Administration Task

User ID: WSADMSH
UID: 2504

< Back Next > Finish Cancel

Figure 3-42 zPMT - Defining App_Server_Root

- Deploy manager configuration file system mount point: Specifies the Read/write file system directory where the application data and environment files are written. This field is not writable here, but was specified earlier on the “System Environment: Configuration file system information” window.

- Deployment Manager directory path name relative to mount point: It is the relative path name of the directory within the configuration file system in which the Deployment Manager configuration resides.
- Application server configuration file system mount point: Specifies the Read/write file system directory where the application data and environment files are written. This field is not writable here, but was specified earlier on the “System Environment: Configuration file system information” window.
- Application server directory path name relative to mount point: It is the relative path name of the directory within the configuration file system in which the application server configuration resides.
- Admin asynch operations procedure name: Specifies the JCL procedure name of a started task that is launched by way of the START command by application servers or node agents to perform certain asynchronous administrative operations.
- Asynchronous Administration Task User ID: This user ID is used to run the asynchronous administration operations procedure. It must be a member of the WebSphere Application Server configuration group.
- Asynchronous Administration Task UID: It is the UNIX System Services (UID) number for the Asynchronous Administration Task User ID.

After completing the required fields, click **Next**.

15. The next window allows you to select the applications to deploy on the environment that you are creating. We recommend that you select at least the administrative console application.

Your choices are:

- Administrative console
- Default application
- Sample applications

After selecting the applications, click **Next**.

16. The next window allows you to define the job names, procedure names, and user IDs to use for each process.
 - Deploy manager controller process: The job name is specified in the MVS™ START command JOBNAME parameter, associated with the control region. This is the same as the server short name and it cannot be changed during customization. The procedure name is the member name in your procedure library to start the control region. The User ID is the user ID associated with the control region.

- Deploy manager servant process: Specify the job name used by WLM to start the servant regions. This is set to the server short name, followed by the letter "S", and it cannot be changed during customization. The procedure name is the member name in your procedure library to start the servant regions. The User ID is the user ID associated with the servant regions.
- Application server controller process: Specify the name of the member in your procedure library to start the control region. The User ID is the user ID associated with the control region.
- Application server controller adjunct process: Specify the name of member in your procedure library that starts the control region adjunct. The User ID is the user ID associated with the control region adjunct.
- Application server servant process: Specify the name of member in your procedure library that starts the servant regions. The jobname used by WLM to start the servant regions. This is set to the server short name, followed by the letter "S", and it cannot be changed during customization. The User ID is the user ID associated with the servant regions

After completing the required fields, click **Next**.

17. The next window allows you to specify the ports to use for each process. Once again, good planning is very important to avoid port conflicts, so be sure you have all values you need in order to fill out this window. The required fields in this window are:

- Deployment manager
 - Node host name
 - SOAP JMX connector port
 - Cell Discovery Address port
 - ORB Listener host name
 - ORB port
 - ORB SSL port
 - HTTP transport host name
 - Administrative console port
 - Administrative secure console port
 - High availability manager communication port
- Node agent
 - JMX SOAP Connector port
 - ORB port

- ORB SSL port
- High Availability Manager Communication Port
- Node Discovery Port
- Node Multicast Discovery Port
- Node IPv6 Multicast Discovery Port
- Application server
 - JMX SOAP Connector port
 - ORB port
 - ORB SSL port
 - HTTP port
 - HTTP SSL port
 - High Availability Manager Communication Port
 - Service Integration port
 - Service Integration Secure port
 - Service Integration MQ Interoperability port
 - Service Integration MQ Interoperability Secure port
 - Session Initiation Protocol
 - Session initiation Secure Protocol

After completing the required ports, click **Next**.

18. The next window allows you to specify the location daemon settings. The location daemon service is the initial point of client contact in WebSphere Application Server for z/OS. The server contains the CORBA-based location service agent which places sessions in a cell. All RMI/IIOP IORs (for example, enterprise beans) establish connections to the location service daemon first, then forward them to the target application server.
- Daemon home directory: It is the directory in which the location service daemon resides. This is set to the configuration file system mount point/Daemon and cannot be changed.
 - Daemon job name: Specifies the jobname of the location service daemon, specified in the JOBNAM parameter of the MVS start command used to start the location service daemon. When configuring a new cell, be sure to choose a new daemon jobname value. A server automatically starts the location service daemon if it is not already running.
 - Procedure name: It is the member name in your procedure library to start the location service daemon.

- User ID: Specify the user ID associated with the location service daemon.
- UID: It is the user identifier associated with this user ID. UIDs must be unique numbers within the system.
- IP name: It is the fully qualified IP name, registered with the Domain Name Server (DNS), that the location service daemon uses. The default is your node host name. In a sysplex, you should consider using a virtual IP address (VIPA) for the location service daemon IP name. Select the IP name for the location service daemon carefully. You can choose any name you want, but, once chosen, it is difficult to change, even in the middle of customization.
- Listen IP: It is the address at which the daemon listens. Select either * or a dotted IP address for this value.
- Port number: Specify the port number on which the location service daemon listens.
- SSL port: The port number on which the location service daemon listens for SSL connections.

Note: Choose the IP name and port number carefully since it is difficult to change, even in the middle of customization.

- Register daemon with WLM DNS check box: If you use the WLM DNS (connection optimization), you must register your location service daemon. Otherwise, do not register your location service daemon. Only one location service daemon per LPAR can register its domain name with WLM DNS; if you have multiple cells in the same LPAR and register more than one location service, it will fail to start.

After completing the required fields, click **Next**.

19. The next window allows you to enter SSL configuration values.

- Certificate authority keylabel: It is the name that identifies the certificate authority (CA) to be used in generating server certificates.
- Generate certificate authority (CA) certificate check box: It is selected to generate a new CA certificate. Do not select this option to have an existing CA certificate generate server certificates.
- Expiration date for certificates: It is used for any X509 Certificate Authority certificates created during customization, as well as the expiration date for the personal certificates generated for WebSphere Application Server for z/OS servers. You must specify this even if you have not selected **Generate Certificate Authority (CA) certificate**.

- Default SAF keyring name: It is the default name given to the RACF® keyring used by WebSphere Application Server for z/OS. The keyring names created for repertoires are all the same within a cell.
- Enable SSL on location service daemon check box: Select if you want to support secure communications using Inter-ORB Request Protocol (IIOP) to the location service daemon using SSL. If selected, a RACF keyring will be generated for the location service daemon to use.

After completing the required SSL information, click **Next**.

20. The next window allows you to select the user registry to be used for administrative security. You can choose from the following options:

- z/OS security product option: This option uses the z/OS system's SAF compliant security product, such as IBM RACF or equivalent, to manage WebSphere Application Server identities and authorization according to the following:
 - The SAF security database will be used as the WebSphere user repository.
 - SAF EJBROLE profiles will be used to control role-based authorization, including administrative authority.
 - Digital certificates will be stored in the SAF security database.

Note: Select the z/OS security product option if you are planning to use the SAF security database as your WebSphere Application Server registry or if you plan to set up an LDAP or custom user registry whose identities will be mapped to SAF user IDs for authorization checking. For this security option, you must decide whether to set a security domain name, and choose an administrator user ID and an unauthenticated (guest) user ID.

- WebSphere Application Server security option: The WebSphere Application Server administrative security option is used to manage the Application Server identities and authorization according to the following:
 - A simple file-based user registry will be built as part of the customization process.
 - Application-specific role binds will be used to control role-based authorization.
 - The WebSphere Application Server console users and groups list will control administrative authority.
 - Digital certificates will be stored in the configuration file system as keystores.

Note: Choose this option if you plan to use an LDAP or custom user registry without mapping to SAF user IDs. (The file-based user registry is not recommend for production use.)

- No security

Although it is not recommended, you may disable administrative security. If you choose this security option, there are no other choices to make. Your WebSphere Application Server environment will not be secured until you configure and enable security manually. You can enable security manually later via the administrative console or using Jython scripts.

Select an option and click **Next**.

21. The next window you see will depend on the security option you choose.

- z/OS product option

Figure 3-43 shows the parameters to enter if you chose to use a z/OS product for security.

The screenshot shows the 'Profile Management Tool' window. The title bar is blue with a cross icon and the text 'Profile Management Tool'. Below the title bar, the main content area has a light beige background. At the top, it says 'Security managed by the z/OS product' in bold, followed by 'z/OS cell (deployment manager and an application server)' in a smaller font. To the right of this text is an icon of a person with a checkmark. Below this, there is a section for 'Security domain identifier (optional):' with a text input field. Then, there is a section for 'WebSphere Application Server administrator' with 'User ID:' set to 'WSADMIN' and 'UID:' set to '2403'. Finally, there is a section for 'WebSphere Application Server unauthenticated user' with 'User ID:' set to 'WSGUEST' and 'UID:' set to '2402'. Each UID field has a small dropdown arrow next to it.

Figure 3-43 Creating a profile: Using z/OS security

- **Security domain identifier:** (Optional) Used to distinguish between APPL or EJBROLE profiles based on security domain name; provides an alphanumeric security domain name of one to eight characters. Internally, this sets SecurityDomainType to the string cellQualified. All servers in the cell will prepend the security domain name you specify to the application-specific J2EE role name to create the SAF EJBROLE profile for checking. The security domain name is not used, however, if role checking is performed using WebSphere Application Server for z/OS bindings. The security domain name is also used as the APPL profile name and inserted into the profile name used for CBIND checks. The RACF jobs that the Customization Dialog generates create and authorize the appropriate RACF profiles for the created nodes and servers. If you do not want to use a security domain identifier, leave this field blank.
- **WebSphere Application Server administrator user ID:** The initial WebSphere Application Server administrator. It must have the WebSphere Application Server configuration group as its default UNIX System Services group. The UNIX System Services UID number for the administrator user ID is specified here, and must be a unique numeric value between 1 and 2,147,483,647.
- **WebSphere Application Server unauthenticated user ID:** Associated with unauthenticated client requests. It is sometimes referred to as the "guest" user ID. It should be given the RESTRICTED attribute in RACF, to prevent it from inheriting UACC-based access privileges. The UNIX System Services UID number for the user ID is specified here and is associated with unauthenticated client requests. The UID value must be unique numeric values between 1 and 2,147,483,647.

Click **Next**.

- WebSphere Application Server security option

On this window (Figure 3-44), specify a user name and password to log in to the administrative console and perform administrative tasks. The sample applications require a user and password as well.

Specify a user name and password to login to the administrative console and perform administrative tasks.

User name:

Password:

Confirm password:

Sample applications

The Sample applications require an account in the profile. Assign a password to the samples user account.

User name:

Password:

Confirm password:

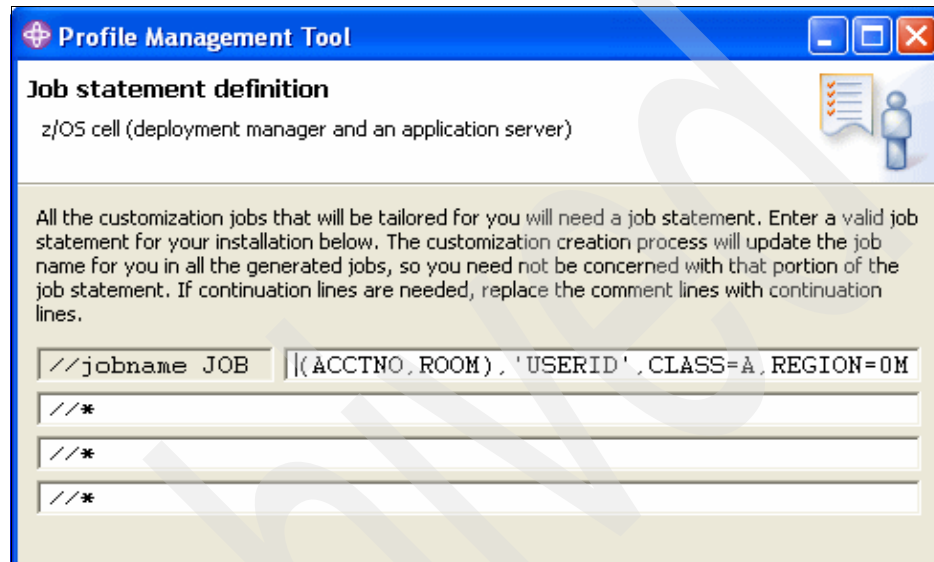
Figure 3-44 Creating a profile: Using WebSphere family security

Click **Next**.

22. The next window allows you to create a Web server definition for an existing Web server. This step is not required for your configuration and can be easily done after the run time environment is up and running by using the administrative console. You can only have one Web server defined on a stand-alone application server.

Click **Next**.

23. The next window (Figure 3-45) allows you to tailor the JCL for the customization jobs. Enter a valid job statement for your installation on this window. The profile creation process will update the job name for you in all the generated jobs, so you need not be concerned with that portion of the job statement. If continuation lines are needed, replace the comment lines with continuation lines.



The screenshot shows a window titled "Profile Management Tool" with a blue header bar. Below the header, the section "Job statement definition" is displayed. Under this section, the text "z/OS cell (deployment manager and an application server)" is shown. A large text area contains instructions: "All the customization jobs that will be tailored for you will need a job statement. Enter a valid job statement for your installation below. The customization creation process will update the job name for you in all the generated jobs, so you need not be concerned with that portion of the job statement. If continuation lines are needed, replace the comment lines with continuation lines." Below the text area, there are four input fields. The first field contains the text "//jobname JOB ||(ACCTNO,ROOM), 'USERID', CLASS=A, REGION=0M". The second, third, and fourth fields each contain the text "//*".

Figure 3-45 Creating a profile: job statement definition

After you are done with this window, click **Next**.

24. The last window shows a short summary of the customization, including profile type and where the generated jobs will be stored. To change the characteristics of this profile, click the **Back** button; otherwise, click **Create** to generate your z/OS Customization jobs and a status window is shown after clicking **Create**.

When zPMT is done, it will display a summary window (Figure 3-46) that indicates whether the jobs were created successfully or not. If the jobs were not created, a log file containing failure information will be identified. If successful, the next step in the z/OS customization process is to upload these jobs and the associated instructions to a pair of z/OS partitioned data sets. To do this, click **Finish** to return to the zWebSphere preference window and select **Upload**.

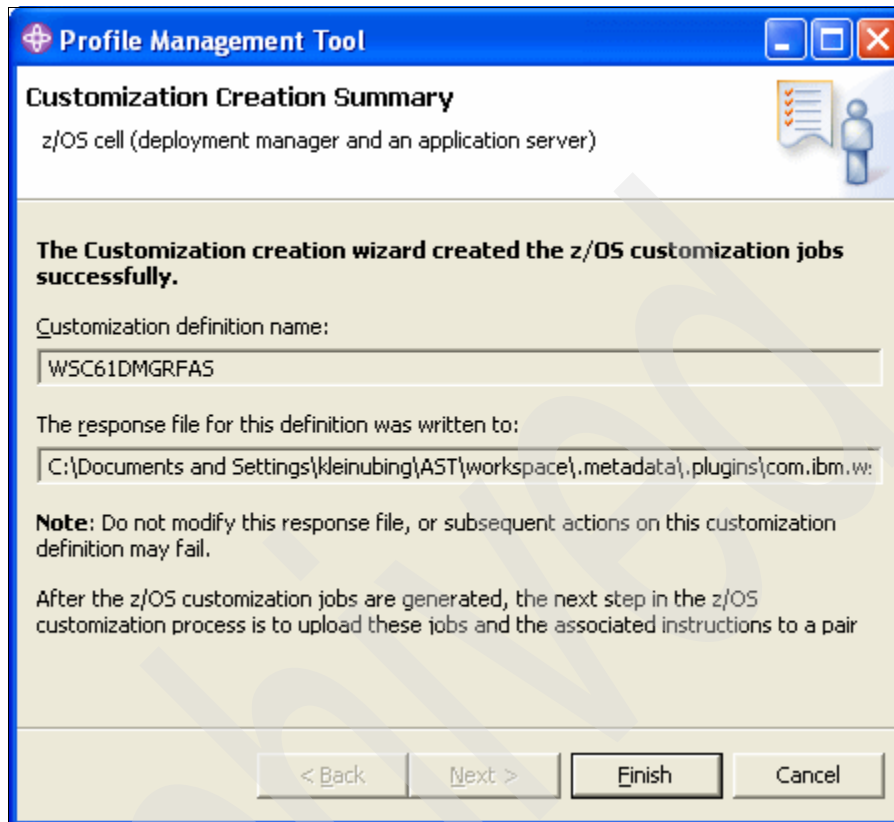


Figure 3-46 Creating a profile: Customization creation summary window

25. Now, on the main window, select the profile you have just create and then click the **Upload...** button.

In the upload customization definition window (Figure 3-47 on page 122), enter the target z/OS system. This must be fully qualified or the upload will fail.

Use the **Allocate target z/OS data sets** check box to specify whether to allocate the data sets if they do not exist (box check). If the data sets exist and are to be reused, clear the box.

You will see a progress information window while the upload is occurring.

IBM WebSphere for z/OS Customization

Upload customization definition

Specify the following ftp information and then click **Upload**.

Target z/OS system: MVS61.itso.raleigh.ibm.com

User ID: tkleinubing

Password: *****

The customization data will be uploaded to the following partitioned data sets on the target z/OS system:

SYS61.WAS61.ITSOCELL.DMGRFAS.CNTL

SYS61.WAS61.ITSOCELL.DMGRFAS.DATA

☒ Allocate target z/OS data sets

Optionally, specify both volume and unit for target data sets.

Volume:

Upload **Cancel**

Figure 3-47 Creating a profile: Upload customization definition window

26. Once the customization profile is uploaded, follow the instructions in the BBOSSINS member of the CNTL data set. You can view this data set on the host, or select the configuration and use the **View...** button.

These instructions will help you determine what jobs to run, the order to run them in, and the expected results. It will also tell you how to start the environment once you are done. From this point, the process is identical to that you would use with the ISPF Customization Dialog process.

3.5 Managing profiles

Each profile you create is registered in a profile registry:

```
<was_home>/properties/profileRegistry.xml
```

You have already seen how profiles are created with the Profile Management Tool. At the heart of this wizard is the **manageprofiles** command. This command provides you with the means to do normal maintenance activities for profiles. For example, you can call this command to create profiles natively or silently, list profiles, delete profiles, validate the profile registry, and other functions.

3.5.1 Using the manageprofiles command

The **manageprofiles** command can be found in the `<was_home>/bin` directory.

Syntax

Use the following syntax for the **manageprofiles** command:

- ▶ For Windows, use **manageprofiles.bat -mode -arguments**
- ▶ For UNIX, use **manageprofiles.sh -mode -arguments**

The following modes in Table 3-5 are available.

Table 3-5 *manageprofiles modes*

Mode	Use
-create:	Creates a new profile.
-augment	Augments the given profile using the given profile template.
-delete	Deletes a profile.
-unaugment:	Unaugments the profile.
-deleteAll	Deletes all registered profiles.
-listProfile	Lists the profiles in the profile registry.
-getName	Returns the name of the profile at the path specified.
-getPath	Returns the path of the profile name specified.
-validateRegistry	Validates the profile registry and returns a list of profiles that are not valid.
-validateAndUpdateRegistry	Validates the profile registry and lists the non-valid profiles that it purges.
-getDefaultName	Returns the name of the default profile.

Mode	Use
-setDefaultName	Sets the default profile.
-backupProfile	Back ups the given profile into a zip file.
-restoreProfile	Restores the given profile from a zip file.
-response	Manage profiles from a response file.
-help	Shows help.

The following two examples show the results of **manageprofiles -<mode> - help** and **manageprofiles -listProfiles** modes:

- Enter **manageprofiles -<mode> -help** for detailed help on each mode. See Example 3-2 for an example of the **manageprofiles -create -help** command.

Example 3-2 Getting help for the manageprofiles command

```
C:\Program Files\IBM\WebSphere\AppServer\bin>manageprofiles -create -help
```

The following command line arguments are required for this mode.
Command-line arguments are case sensitive.

-create: Creates a new profile. Specify -help -create -templatePath <path> to get template-specific help information.

-templatePath: The fully qualified path name of the profile template that is located on the file system. The following example selects a template:

-templatePath <app_server_home>/profileTemplates/<Template_name>

-profileName: The name of the profile.

-profilePath: The intended location of the profile in the file system.

The following command line arguments are optional, and have no default values.
Command-line arguments are case sensitive.

-isDefault: Make this profile the default target of commands that do not use their profile parameter.

- Enter **manageprofiles -listProfiles** to see a list of the profiles in the registry. The following is a sample output of -listProfiles:

```
C:\Program Files\IBM\WebSphere\AppServer\bin>manageprofiles -listProfiles
[Dmgr01, AppSrv01, Custom01, Custom02, Dmgr02]
```

3.5.2 Creating a profile

You can use the **manageprofiles** command to create profiles instead of using the Profile Management Tool.

Profile templates: The profiles are created based on templates supplied with the product. These templates are located in `<was_home>/profileTemplates`. Each template consists of a set of files that provide the initial settings for the profile and a list of actions to perform after the profile is created. Currently, there is no provision for modifying these templates for your use, or for creating new templates. When you create a profile using **manageprofiles**, you will need to specify one of the following templates:

- ▶ default (for application server profiles)
- ▶ dmgr (for deployment manager profiles)
- ▶ managed (for custom profiles)
- ▶ cell (for cell profiles)

For example, Example 3-3 shows the commands used to create an application server named `saserver1` on node `sanodel` in cell `sacell1` on host `kcgg1d7.itso.ibm.com` from the command line.

Example 3-3 Creating a profile with the `manageprofiles` command

```
cd $WAS_HOME\bin
{assuming WAS_HOME was set to some value, for example to "C:\Program
Files\IBM\WebSphere\AppServer"}
manageprofiles -create -profileName saserver1 -profilePath
C:\myWAS61Profiles\appSrvrProfiles\saserver1 -templatePath
$WAS_HOME\profileTemplates\default -nodeName sanodel -cellName sacell1
-hostName kcgg1d7.itso.ibm.com
```

Creating a profile in silent mode

Profiles can also be created in silent mode using a response file. The command to use is:

```
<profile_management_tool> -options <response_file> -silent
```

In this example, `<profile_management_tool>` is the command required to start the Profile Management Tool. The command to start the wizard is platform-specific and is located in `<was_home>/bin/ProfileManagement`. Choose your platform command from Table 3-6.

Table 3-6 Platform-specific creation wizard

Platform (32-bit)	Profile Management Tool command
Linux/HP-UX/Solaris/AIX	<code>pmt.sh</code>
Windows	<code>pmt.bat</code>

Sample response files are stored in the `<was_home>/bin/profileCreator` directory.

3.5.3 Deleting profiles

To delete a profile, you should do the following:

- ▶ If you are removing a custom profile or application server profile that has been federated to a cell:
 - Stop the application servers on the node.
 - Remove the node from the cell using the administrative console or the **removeNode** command. Removing a node does not delete it, but restores it to its pre-federated configuration that was saved as part of the federation process.
 - Delete the profile using **manageprofiles -delete**.
 - Use the **manageprofiles -validateAndUpdateRegistry** command to clean the profile registry.
 - Delete the `<profile_home>` directory.
- ▶ If you are removing an application server profile that has not been federated to a cell:
 - Stop the application server.
 - Delete the profile using **manageprofiles -delete**.
 - Use the **manageprofiles -validateAndUpdateRegistry** command to clean the profile registry.
 - Delete the `<profile_home>` directory.
- ▶ If you are removing a deployment manager profile:
 - Remove any nodes federated to the cell using the administrative console or the **removeNode** command. Removing a node does not delete it, but restores it to its pre-federated configuration that was saved as part of the federation process.
 - Stop the deployment manager.
 - Delete the profile using **manageprofiles -delete**.
 - Use the **manageprofiles -validateAndUpdateRegistry** command to clean the profile registry.
 - Delete the `<profile_home>` directory.

Deleting a profile with manageprofiles

To delete a profile, use the **manageprofiles -delete** command. The format is:

```
manageprofiles -delete -profileName <profile>
```

At the completion of the command, the profile will be removed from the profile registry, and the run time components will be removed from the *<profile_home>* directory with the exception of the log files.

If you have errors while deleting the profile, check the following log:

```
<was_home>/logs/manageprofile/<profile_name>_delete.log
```

For example, in Example 3-4, you can see the use of the **manageprofiles** command to delete the profile named Node06.

Example 3-4 Deleting a profile using manageprofiles

```
C:\WebSphere\ND\profiles\Dmgr01\bin>manageprofiles -delete -profileName Node06
INSTCONFSUCCESS: Success: The profile no longer exists.
```

As you can see in Example 3-4, all seems to have gone well. But, as an additional step to ensure the registry was properly updated, you can list the profiles to ensure the profile is gone from the registry and validate the registry. See Example 3-5.

Example 3-5 Verifying the delete profile results

```
C:\WebSphere\ND\profiles\Dmgr01\bin>manageprofiles -listProfiles
[Dmgr01, AppSrv01, AppSrv02, SamplesServer, WebServer2Node, DmgrSecure]

C:\WebSphere\ND\profiles\Dmgr01\bin> manageprofiles -validateAndUpdateRegistry
[]
```

Note: If there are problems during the delete, you can manually delete the profile. For information about this, see the *Deleting a profile* topic in the Information Center.

3.6 Managing the processes

In a stand-alone server environment, you only have one process, the application server, so it is clear how to stop and start the environment. But, when starting or stopping a distributed server environment, it helps to do this in an orderly manner. In some cases that we point out, the order is necessary. In others, it simply makes good administrative sense.

3.6.1 Starting a distributed server environment

An orderly procedure for starting a distributed server environment involves the following steps:

1. On the deployment manager machine:
 - a. Change the directory to the `<profile_home>/bin` directory of the Network Deployment installation.
 - b. Use the **startManager** command to start the deployment manager.
If you are successful, you will see the process ID for the deployment manager process displayed on the window. See Example 3-6.

Example 3-6 Starting the deployment manager from the command line

```
C:\myWAS61Profiles\dmgrProfiles\DmgrProfile1\bin>startmanager
ADMU0116I: Tool information is being logged in file
C:\myWAS61Profiles\dmgrProfiles\DmgrProfile1\logs\dmgr\startServer.log
ADMU0128I: Starting tool with the DmgrProfile1 profile
ADMU3100I: Reading configuration for server: dmgr
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server dmgr open for e-business; process id is 1120
```

If there are any errors, check the log file for the dmgr process:

`<profile_home>/logs/dmgr/SystemOut.log`

2. On each node, do the following:
 - a. Change directory to the `<profile_home>/bin` directory for the application server on that node.
 - b. Run the **startNode** command.

If successful, the node agent server process ID will be displayed on the window, as shown in this sample:

```
C:\myWAS61Profiles\appSrvrProfiles\AppSrvProfile1\bin>startnode
ADMU0116I: Tool information is being logged in file
C:\myWAS61Profiles\appSrvrProfiles\AppSrvProfile1\logs\nodeagent\startServer.log
ADMU0128I: Starting tool with the AppSrvProfile1 profile
ADMU3100I: Reading configuration for server: nodeagent
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server nodeagent open for e-business; process id is 3356
```

If there are any errors, check the log file for the node agent process by typing this command:

`<profile_home>/logs/nodeagent/SystemOut.log`

- c. Use the **startServer** command to start each of the application server processes on the node.
 - d. Check the node status by running the **serverStatus -all** command.
3. Repeat step 2 on page 128 for each and every node associated with this deployment manager.

3.6.2 Stopping the distributed server environment

The following is a logical sequence of steps to follow to stop a distributed server environment:

1. On each node agent machine:
 - a. Use the **stopServer** command to stop each of the application server processes on the node.
 - b. Use the **stopNode** command to stop the node agent process.
 - i. Change directory to the `<profile_home>/bin` directory for the application server on that node.
 - ii. Run the **stopNode** command.

If successful, the message `Server <node_agent> stop completed` is displayed on the console, as shown in this sample:

If there are any errors, check the log file for the node agent process:

`<profile_home>/logs/dmgr/SystemOut.log`
 - c. Check the node status by running the **serverStatus -all** command.
2. Repeat step 2 on page 128 for each and every node associated with this deployment manager.
3. On the deployment manager machine:
 - a. Change directory to the `<profile_home>/bin` directory of the deployment manager.
 - b. Use the **stopManager** command to stop the deployment manager (dmgr) process.

If the procedure is successful, you will see a `Server dmgr stop completed` message.

If there are any errors, check the log file for the dmgr process:

`<profile_home>/logs/dmgr/SystemOut.log`

Note: Stopping the deployment manager does not stop any node agents.

3.6.3 Enabling process restart on failure

WebSphere Application Server does not have either:

- ▶ A nanny process to monitor whether the AdminServer process is running, and restart it if the process has failed
- ▶ An AdminServer process to monitor whether each application server process is running, and restart it if the process has failed

Instead, WebSphere Application Server uses the native operating system functionality to restart a failed process. Refer to the sections below that discuss your operating system.

Windows

The administrator can choose to register one or more of the WebSphere Application Server processes on a machine as a Windows service during profile creation, or after by using the **WASService** command. With this command, Windows then automatically attempt to restart the service if it fails.

Syntax

Enter **WASService.exe** with no arguments to get a list the valid formats. See Example 3-7.

Example 3-7 WASService command format

Usage: WASService.exe (with no arguments starts the service)

```
|| -add <service name>
|| -serverName <Server>
|| -profilePath <Server's Profile Directory>
|| [-wasHome <Websphere Install Directory>]
|| [-configRoot <Config Repository Directory>]
|| [-startArgs <additional start arguments>]
|| [-stopArgs <additional stop arguments>]
|| [-userid <execution id> -password <password>]
|| [-logFile <service log file>]
|| [-logRoot <server's log directory>]
|| [-encodeParams]
|| [-restart <true | false>]
|| [-startType <automatic | manual | disabled>]
|| -remove <service name>
|| -start <service name> [optional startServer.bat parameters]
|| -stop <service name> [optional stopServer.bat parameters]
|| -status <service name>
|| -encodeParams <service name>
```

Be aware of the following when using the **WASService** command:

- ▶ When adding a new service, the **-serverName** argument is mandatory. The **serverName** is the process name. If in doubt, use the **serverstatus -all** command to display the processes. For a deployment manager, the **serverName** is **dmgr**, for a node agent it is **nodeagent**, and for a server, it is the server name.
- ▶ The **-profilePath** argument is mandatory. It specifies the home directory for the profile.
- ▶ Use unique service names. The services are listed in the Windows Services control window as:

IBM WebSphere Application Server V6.1 - <service name>

The convention used by the Profile Management Tool is to use the node name as the service name for a node agent. For a deployment manager, it uses the node name of the deployment manager node concatenated with **dmgr** as the service name.

Examples

Example 3-8 shows using the **WASService** command to add the deployment manager as a Windows service and sample successful output.

Example 3-8 Registering a deployment manager as a Windows service

```
C:\Program Files\IBM\WebSphere\AppServer\bin>WASService -add "Deployment Mgr"
-serverName dmgr -profilePath "C:\myWAS61Profiles\dmgrProfiles\DmgrProfile1
-restart true
```

Adding Service: Deployment Mgr

```
Config Root: C:\myWAS61Profiles\dmgrProfiles\DmgrProfile1 -restart
true\config
Server Name: dmgr
Profile Path: C:\myWAS61Profiles\dmgrProfiles\DmgrProfile1 -restart true
Was Home: C:\Program Files\IBM\WebSphere\AppServer\
Start Args:
Restart: 1
```

IBM WebSphere Application Server V6.1 - Deployment Mgr service successfully added.

Note that the service name added in Figure 3-48 on page 132 will be IBM WebSphere Application Server V6.1, concatenated with the name you specified for the service name.

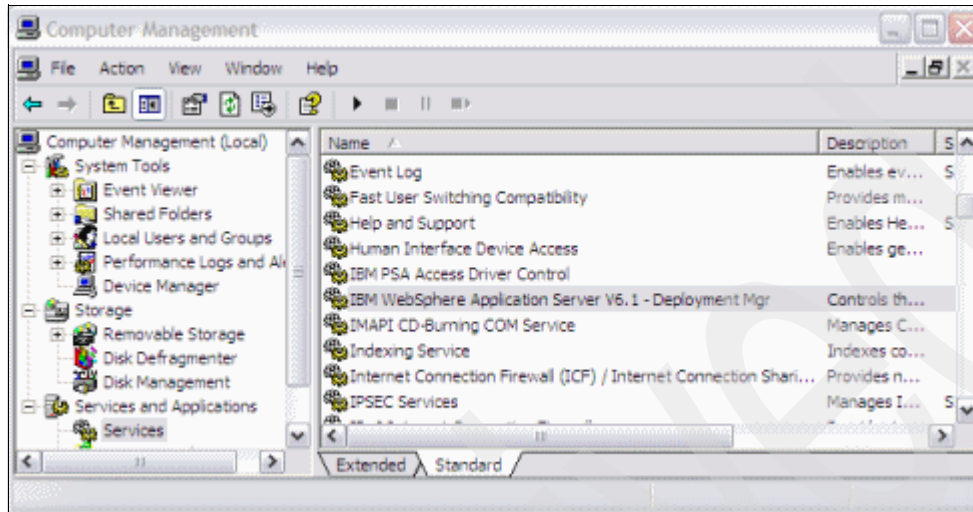


Figure 3-48 New service

If you remove the service using the **WASService -remove** command, specify only the latter portion of the name, as in Example 3-9.

Example 3-9 Removing a service

```
C:\Program Files\IBM\WebSphere\AppServer\bin>WASService -remove "Deployment Mgr"
```

```
Remove Service: Deployment Mgr
Successfully removed service
```

The commands shown in Example 3-10 are used on a WebSphere Application Server node to add the node agent and a server as Windows services.

Example 3-10 Registering WebSphere processes as Windows services

```
C:\Program Files\IBM\WebSphere\AppServer\bin>WASService -add CustomNode
-serverName nodeagent -profilePath
"C:\myWAS61Profiles\cstmProfiles\CstmProfile1 -restart true
```

```
Adding Service: CustomNode
Config Root: C:\myWAS61Profiles\cstmProfiles\CstmProfile1 -restart
true\config
Server Name: nodeagent
```



```

    Profile Path: C:\myWAS61Profiles\cstmProfiles\CstmProfile1 -restart true
    Was Home: C:\Program Files\IBM\WebSphere\AppServer\
    Start Args:
    Restart: 1
IBM WebSphere Application Server V6.1 - CustomNode service successfully added.

C:\Program Files\IBM\WebSphere\AppServer\bin>WASService -add "Cserver1"
-serverName Cserver1 -profilePath "C:\myWAS61Profiles\cstmProfiles\CstmProfile1
-restart true
dding Service: Cserver1
    Config Root: C:\myWAS61Profiles\cstmProfiles\CstmProfile1 -restart
true\config
    Server Name: Cserver1
    Profile Path: C:\myWAS61Profiles\cstmProfiles\CstmProfile1 -restart true
    Was Home: C:\Program Files\IBM\WebSphere\AppServer\
    Start Args:
    Restart: 1
BM WebSphere Application Server V6.1 - Cserver1 service successfully
added.

```

UNIX and Linux

The administrator can choose to include entries in `inittab` for one or more of the WebSphere Application Server V6.1 processes on a machine, as shown in Example 3-11. Each such process will then be automatically restarted if it has failed.

Example 3-11 Inittab contents for process restart

```

On deployment manager machine:
    ws1:23:respawn:/usr/WebSphere/DeploymentManager/bin/startManager.sh

On node machine:
    ws1:23:respawn:/usr/WebSphere/AppServer/bin/startNode.sh
    ws2:23:respawn:/usr/WebSphere/AppServer/bin/startServer.sh nodename_server1
    ws3:23:respawn:/usr/WebSphere/AppServer/bin/startServer.sh nodename_server2
    ws4:23:respawn:/usr/WebSphere/AppServer/bin/startServer.sh nodename_server2

```

Note: When setting the action for `startServer.sh` to `respawn` in `/etc/inittab`, be aware that `init` will always restart the process, even if you intended for it to remain stopped. As an alternative, you can use the `rc.was` script located in `${WAS_HOME}/bin`, which allows you to limit the number of retries.

The best solution is to use a monitoring product that implements notification of outages and logic for automatic restart.

z/OS

WebSphere for z/OS takes advantage of the z/OS Automatic Restart Management (ARM) to recover application servers. Each application server running on a z/OS system (including servers you create for your business applications) are automatically registered with an ARM group. Each registration uses a special element type called SYSCB, which ARM treats as restart level 3, ensuring that RRS (It is a z/OS facility that provides two-phase sync point support across participating resource managers) restarts before any application server.

Note: If you have an application that is critical for your business, you need facilities to manage failures. z/OS provides rich automation interfaces, such as automatic restart management, that you can use to detect and recover from failures. The automatic restart management handles the restarting of servers when failures occur.

Some important things to consider when using automatic restart management:

- ▶ If you have automatic restart management (ARM) enabled on your system, you might want to disable ARM for the WebSphere Application Server for z/OS address spaces before you install and customize WebSphere Application Server for z/OS. During customization, job errors might cause unnecessary restarts of the WebSphere Application Server for z/OS address spaces. After installation and customization, consider enabling ARM.
- ▶ If you are ARM-enabled and you cancel or stop a server, it will restart in place using the `armrestart` command.
- ▶ It is a good idea to set up an ARM policy for your deployment manager and node agents. For more information about how to change the ARM policies please refer to:
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/cins_changearm.html
- ▶ If you start the location service daemon on a system that already has one, it will terminate.
- ▶ Every other server will come up on a dynamic port unless the configuration has a fixed port. Therefore, the fixed ports must be unique in a sysplex.
- ▶ If you issue STOP, CANCEL, or MODIFY commands against server instances, be aware of how automatic restart management behaves regarding WebSphere Application Server for z/OS server instances; Table 3-7 on page 135 depicts the behavior of ARM regarding WebSphere Application Server for z/OS server instances.

Table 3-7 Behavior of ARM and WebSphere Application Server for z/OS server instances

When you issue	ARM behavior
STOP address_space	It will not restart the address space.
CANCEL address_space	It will not restart the address space.
CANCEL address_space, ARMRESTART	It will restart the address space.
MODIFY address_space, CANCEL	It will not restart the address space.
MODIFY address_space, CANCEL,ARMRESTART	It will restart the address space.

For more information about how to activate the ARM, please refer to:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/tins_activearm.html

Let us say you have activated ARM and want to check the status of address spaces registered for automatic restart management; in order to get this information, you need to:

1. Initialize all servers.
2. Issue one or both of the commands shown in Example 3-12.

Example 3-12 Displaying the status of address spaces registered for automatic restart management

To display all registered address spaces (including the address spaces of server instances), issue the command:

```
d xcf,armstatus,detail
```

To display the status of a particular server instance, use the display command and identify the job name. For example, to display the status of the Daemon server instance (job BBODMN), issue the following command:

```
d xcf,armstatus,jobname=bbodmn,detail
```

For more information about how to use the display command, please refer to:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/rxml_mvdisplay.html

Archived

Administration basics

In this chapter, we introduce the WebSphere administrative console, command line administration, and some basic administration tasks.

The topics we cover include:

- ▶ Introducing the WebSphere administrative console
- ▶ Securing the administrative console
- ▶ Working with the deployment manager
- ▶ Working with application servers
- ▶ Working with nodes
- ▶ Working with clusters
- ▶ Working with virtual hosts
- ▶ Managing applications
- ▶ Managing your configuration files

This IBM Redbook does not cover high availability, performance, scalability, or the settings related to these topics. This includes dynamic caching, performance monitoring, failover settings, and others. As you go through this chapter, keep in mind that more information about these topics and settings can be found in the following IBM Redbooks:

- ▶ *WebSphere Application Server Network Deployment V6: High Availability Solutions*, SG24-6688
- ▶ *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392

4.1 Introducing the WebSphere administrative console

The WebSphere administrative console is the graphical, Web-based tool that you use to configure and manage an entire WebSphere cell. It supports the full range of product administrative activities, such as creating and managing resources and applications, viewing product messages, and so on.

In a stand-alone server environment, the administrative console is located on the application server and can be used to configure and manage the resources of that server only.

In a distributed server environment, the administrative console is located in the deployment manager server, dmgr. In this case, the administrative console provides centralized administration of multiple nodes. Configuration changes are made to the master repository and pushed to the local repositories on the nodes by the deployment manager. In order for the administrative console to run, the dmgr server must be running. In order for the changes to the master repository to be pushed to the nodes, the node agents must also be running.

The administrative console groups administrative tasks into the following categories:

- ▶ Guided activities
- ▶ Servers
- ▶ Applications
- ▶ Resources
- ▶ Security
- ▶ Environment
- ▶ System administration
- ▶ Users and groups
- ▶ Monitoring and tuning
- ▶ Troubleshooting
- ▶ Service integration
- ▶ UDDI

4.1.1 Starting the administrative console

The way you access the administrative console is the same whether you have a stand-alone server environment or a distributed server environment. However, the location and how you start the necessary processes will vary.

Stand-alone server environment

In a single application server installation, the console is hosted on the application server, so you must start the server in order to reach the console.

To access the administrative console, do the following:

1. Make sure that application server, server1, is running by using this command:
 - Windows: `<profile_home>\bin\serverStatus -all`
 - UNIX and z/OS: `<profile_home>/bin/serverStatus.sh -all`
2. If the status of server1 is not STARTED, start it with the following command:
 - On Windows: `<profile_home>\bin\startServer server1`
 - On UNIX and z/OS: `<profile_home>/bin/startServer.sh server1`
3. Open a Web browser to the URL of the administrative console. The default port is 9060 for HTTP and 9043 for HTTPS. This port can vary, depending on the ports you specified when you created the application server profile.

`http://<hostname>:9060/ibm/console`
`https://<hostname>:9043/ibm/console`

<hostname> is your host name for the machine running the application server.
4. The administrative console loads and you are asked to log in.

Distributed server environment

If you are working with a deployment manager and its managed nodes, the console is hosted on the deployment manager. You must start it in order to use the console. To access the administrative console, do the following:

1. Make sure that deployment manager, dmgr, is running by using this command:
 - Windows: `<dmgr_profile_home>\bin\serverStatus -all`
 - UNIX and z/OS: `<dmgr_profile_home>/bin/serverStatus.sh -all`
2. If the dmgr status is not STARTED, start it with the following command:
 - On Windows: `<dmgr_profile_home>\bin\startManager`
 - On UNIX and z/OS: `<dmgr_profile_home>/bin/startManager.sh`
3. Open a Web browser to the URL of the administrative console. The default port is 9060 for HTTP and 9043 for HTTPS.

`http://<hostname>:9060/admin`
`https://<hostname>:9043/admin`

<hostname> is your host name for the machine running the deployment manager process, dmgr.
4. The administrative console loads and you are prompted for your user ID and password.

4.1.2 Logging in to the administrative console

The user ID specified during login is used to track configuration changes made by the user. This allows you to recover from unsaved session changes made under the same user ID, for example, when a session times out or the user closes the Web browser without saving. The user ID for login depends on whether WebSphere administrative security is enabled.

- ▶ WebSphere administrative security is not enabled.

You can enter any user ID, valid or not, to log in to the administrative console. The user ID is used to track changes to the configuration, but is not authenticated. You can also simply leave the User ID field blank and click the **Log In** button.

Note: Logging in without an ID is not a good idea if you have multiple administrators.

- ▶ WebSphere administrative security is enabled.

You must enter a valid user ID and password that has been assigned an administrative security role.

A user ID must be unique to the deployment manager. If you enter an ID that is already in use and in session, you will receive the message Another user is currently logged with the same User ID and you will be prompted to do one of the following:

- ▶ Force the existing user ID out of session. You will be allowed to recover changes that were made in the other user's session.
- ▶ Wait for the existing user ID to log out or time out of the session.
- ▶ Specify a different user ID.

Note: The message Another user is currently logged with the same User ID appears if a previous session ended without a logout. For example, if the user closed a Web browser during a session and did not log out first or if the session timed out.

Recovering from an interrupted session

Until you save the configuration changes you make during a session, the changes do not become effective. If a session is closed without saving the configuration changes made during the session, these changes are remembered and you are given the chance to pick up where you left off.

When unsaved changes for the user ID exist during login, you are prompted to do one of the following:

- ▶ Work with the master configuration

Selecting this option specifies that you want to use the last saved administrative configuration. Changes made to the user's session since the last saving of the administrative configuration will be lost.

- ▶ Recover changes made in a prior session

Selecting this option specifies that you want to use the same administrative configuration last used for the user's session. It recovers all changes made by the user since the last saving of the administrative configuration for the user's session.

As you work with the configuration, the original configuration file and the new configuration file are stored in a folder at:

`<profile_home>/wstemp`

Once you save the changes, these files are removed from the wstemp folder.

Each user who logs in has a folder created in wstemp. Even when there are no unsaved changes, the folder will contain a preferences.xml file with the user preference settings.

For information about how to change the default location, refer to the *Changing the location of temporary workspace files* topic in the Information Center.

4.1.3 Changing the administrative console session timeout

You might want to change the session timeout for the administrative console application. This is the time it takes for the console session to time out after a period of idleness. The default is 30 minutes. To change the session timeout value, do the following:

1. Copy the following script into a file. See Example 4-1.

Example 4-1 Jython script to change the console session expiration

```
dep = AdminConfig.getid("/Deployment:isclite/" )
appDep = AdminConfig.list("ApplicationDeployment", dep )
sesMgmt = AdminConfig.list("SessionManager", appDep )

# check if existing sesMgmt there or not, if not then create a new one, if
exist then modify it
if (sesMgmt == ""):
    # get applicationConfig to create new SessionManager
    appConfig = AdminConfig.list("ApplicationConfig", appDep )
    if (appConfig == ""):
        # create a new one
        appConfig = AdminConfig.create("ApplicationConfig", appDep, []
    )
    # then create a new SessionManager using new Application Config
    just created
    sesMgmt = AdminConfig.create("SessionManager", appConfig, [] )
    else:
        # create new SessionManager using the existing
        ApplicationConfig
        sesMgmt = AdminConfig.create("SessionManager", appConfig, [] )
    #endElse
#endIf

# get tuningParams config id
tuningParams = AdminConfig.showAttribute(sesMgmt, "tuningParams" )
if (tuningParams == ""):
    # create a new tuningParams
    AdminConfig.create("TuningParams", sesMgmt, [["invalidationTimeout",
<timeout value>]] )
else:
    #modify the existing one
    AdminConfig.modify(tuningParams, [["invalidationTimeout", <timeout
value>]] )
#endElse
# saving the configuration changes
AdminConfig.save()
```

2. Change the <timeout value> on the two lines of this sample to the new session expiration value. This number specifies the number of minutes the console preserves the session during inactivity.
3. Save the file to any directory using, for example, the filename timeout.py.
4. Start the **wsadmin** scripting client from the <was_home>/profiles/<profile_name>/bin directory.

Issue the following command.

```
wsadmin -f <path to jython file>/timeout.py
```

4.1.4 The graphical interface

The WebSphere administrative console has the following main areas:

- ▶ Taskbar
- ▶ Navigation tree
- ▶ Workspace, including the messages and help display areas.

Each area can be resized as desired. See Figure 4-1.

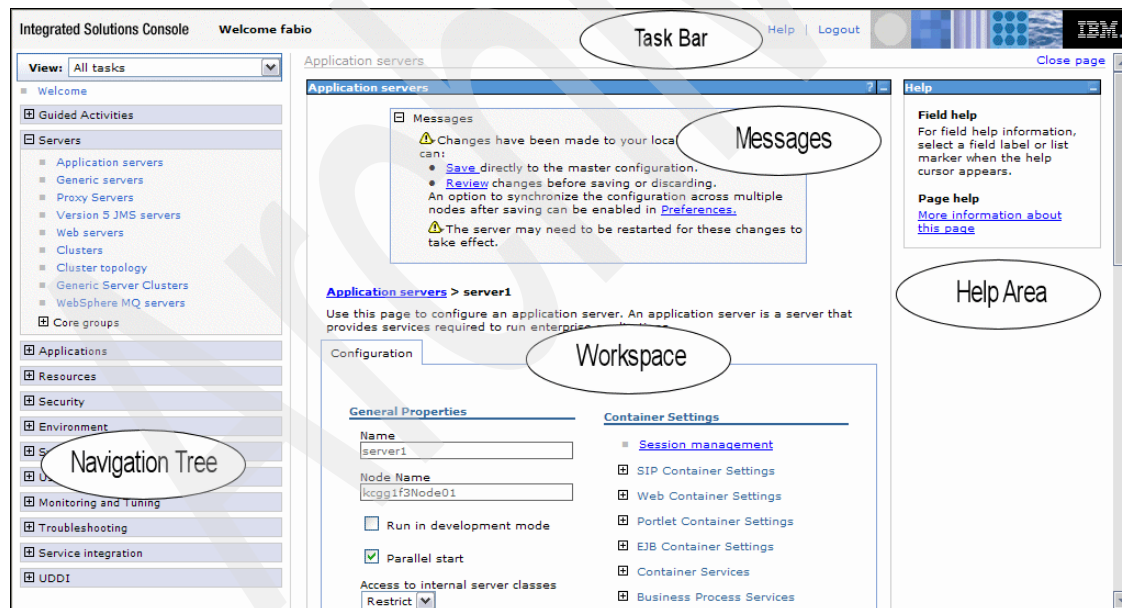


Figure 4-1 The administrative console graphical interface

Taskbar

The taskbar is the horizontal bar near the top of the console. The taskbar provides the following actions:

- ▶ Logout logs you out of the administrative console session and displays the Login page. If you have changed the administrative configuration since last saving the configuration to the master repository, the Save page displays before returning you to the Login page. Click **Save** to save the changes, **Discard** to return to the administrative console, or **Logout** to exit the session without saving changes.
- ▶ Help opens a new Web browser with detailed online help for the administrative console. This is not part of the Information Center.

The taskbar display is controlled with the Show banner setting in the console preferences. See “Setting console preferences” on page 146.

Navigation tree

The navigation tree on the left side of the console offers links for you to view, select, and manage components.

Clicking a + beside a tree folder or item expands the tree for the folder or item. Clicking a - collapses the tree for the folder or item. Double-clicking an item toggles its state between expanded and collapsed.

The content displayed on the right side of the console, the *workspace*, depends on the folder or item selected in the tree view.

The folders shown in Table 4-1 are provided for selection.

Table 4-1 Navigation tree options

Navigation tree option	Description	Standalone	Deployment Manager
Guided Activities	Guided activities lead you through common administrative tasks that require you to visit multiple administrative console pages.	Yes	Yes
Servers	Enables configuration of application servers, clusters, and external servers.	Limited	Yes
Applications	Enables installation and management of applications.	Yes	Yes
Resources	Enables configuration of resources, including JMS providers, asynchronous beans, caching, mail providers, URL providers, and others.	Yes	Yes

Navigation tree option	Description	Standalone	Deployment Manager
Security	Enables configuration and management of WebSphere security, SSL, and Web services security.	Limited	Yes
Environment	Enables configuration of hosts, replication domains, environment variables, naming, and others.	Yes	Yes
System Administration	Enables configuration and management of nodes, cells, console settings. This is also where you save configuration changes.	Limited	Yes
Users and Groups	Enables creation and update of user and groups and their administrative roles.	Yes	Yes
Monitoring and Tuning	Enables you to work with the Performance Monitor Infrastructure (PMI), request metrics, and the Tivoli Performance Viewer.	Yes	Yes
Troubleshooting	Enables you to check for and track configuration errors and problems. This section contains messages resulting from configuration changes and the run time messages.	Yes	Yes
Service Integration	Enables you to work with the service integration bus.	Yes	Yes
UDDI	Allows you to work with the private UDDI registry functions.	Yes	Yes

Workspace

The workspace, on the right side of the console in Figure 4-1 on page 143, allows you to work with your administrative configuration after selecting an item from the console navigation tree.

When you click a folder in the tree view, the workspace lists information about instances of that folder type, the collection page. For example, selecting **Servers** → **Application Servers** shows all the application servers configured in this cell. Selecting an item, an application server in this example, displays the detail page for that item. The detail page can contain multiple tabs. For example, you might have a Runtime tab for displaying the run time status of the item, and a Configuration tab for viewing and changing the configuration of the displayed item.

Messages are displayed at the top of the workspace, while help information is displayed to the right.

The display of help information can be controlled with the Show Descriptions console preference setting.

Setting console preferences

The look of the administrative console can be altered by setting console preferences. See Figure 4-2 on page 146.

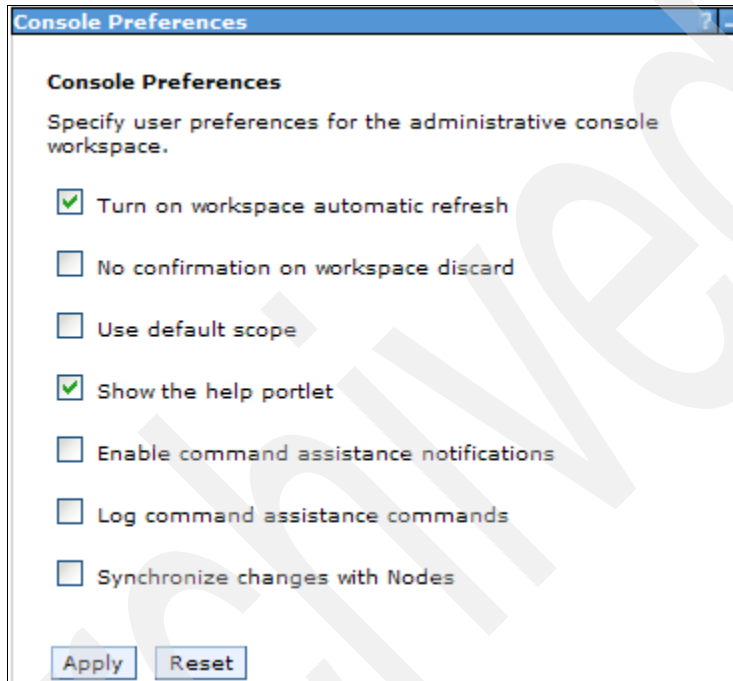


Figure 4-2 Administrative console preferences

To set console preferences, select **System Administration** → **Console settings** → **Preferences** in the navigation tree. You have the following options:

- ▶ Turn on WorkSpace Auto-Refresh specifies that the view automatically refreshes after a configuration change. If it is not selected, you must reaccess the page to see the changes.
- ▶ No Confirmation on Workspace Discard specifies that a confirmation window be displayed if you elect to discard the workspace. For example, if you have unsaved changes and log out of the console, you will be asked whether you want to save or discard the changes. If this option is not selected and you elect to discard your changes, you will be asked to confirm the discard action.
- ▶ Use default scope (administrative console node) sets the default scope to the node of the administration console.

- ▶ Show the help portlet displays the help portlet at right top.
- ▶ Enable command assistance notifications specifies whether you want to enable integration with the Application Server Toolkit scripting tool.
- ▶ Log command assistance commands specifies whether to log all the command assistance wsadmin data for the current user.
- ▶ Synchronize changes with Nodes synchronizes changes that are saved to the deployment manager profile with all the nodes that are running.

Click the boxes to select which preferences you want to enable and click **Apply**.

4.1.5 Finding an item in the console

To locate and display items within a cell, do the following:

1. Select the associated task from the navigation tree. For example, to locate an application server, select **Servers** → **Application Servers**.
2. Certain resources are defined at a scope level. If applicable, select the scope from the drop-down. With V6.1, you can now display resources at all scopes.
3. Set the preferences to specify how you would like information to be displayed on the page.

Select task

The navigation tree on the left side of the console contains links to console pages that you use to create and manage components in the cell. To create a JDBC provider, for example, expand **Resources** and then select the **JDBC Providers** action. See Figure 4-3.



Figure 4-3 Working with the administrative console

Select a scope

After selecting an action, use the scope settings to define what information is displayed. Not all actions will require a scope setting. See Figure 4-4.

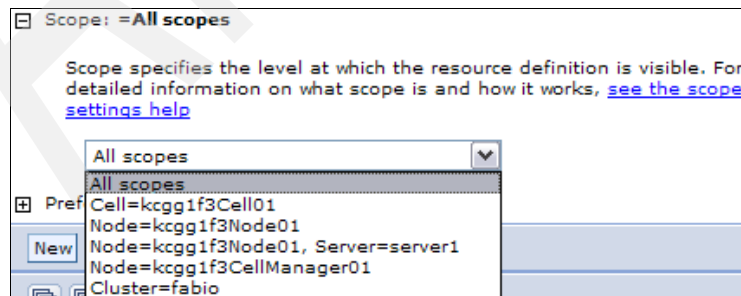


Figure 4-4 Setting scope

Configuration information is defined at the following levels: cell, cluster, node, server, and application. The scope determines which applications or application servers will see and use that configuration.

Configuration information is stored in the repository directory that corresponds to the scope. For example, if you scope a resource at the node level, the configuration information for that resource is in `<profile_home>/config/cells/<cell>/nodes/<node>/resources.xml`. If you scoped that same resource at the cell level, the configuration information for that resource is in `<profile_home>/config/cells/<cell>/resources.xml`.

The following lists the scopes in overriding sequence. Because you see application scope first, anything defined at this scope overrides any conflicting configuration you might find in the higher level scopes.

1. Resources and variables scoped at the application level apply only to that application. Resources and variables are scoped at the application level by defining them in an enhanced EAR. They cannot be created from the WebSphere administrative tools, but can be viewed and modified (in the administrative console, navigate to the details page for the enterprise application and select **Application scoped resources** in the References section).
2. Resources scoped at the server level apply only to that server. If a node and server combination is specified, the scope is set to that server. Shared libraries configured in an enhance EAR are automatically scoped at the server level.
3. Resources scoped at the node level apply to all servers on the node.
4. Resources scoped at the cluster level apply to all application servers in the cluster. New cluster members automatically have access to resources scoped at this level. If you do not have any clusters defined, you will not see this option.
5. Resources scoped at the cell level apply to all nodes and servers in the cell.

Select the scope from the drop-down.

The scope setting is available for all resource types, WebSphere variables, shared libraries, and name space bindings.

Stand-alone application servers: Although the concept of cells and nodes is more relevant in a managed server environment, scope is also set when working with stand-alone application servers. Because there is only one cell, node, and application server, and no clusters, simply let the scope default to the node level.

Set preferences for viewing the console page

After selecting a task and a scope, the administrative console page shows a collection table with all the objects created at that particular scope.

You can change the list of items you see in this table by using the filter and preference settings. The filter options can be displayed or set by clicking the Show Filter Function icon  at the top of the table. See Figure 4-5 on page 150.

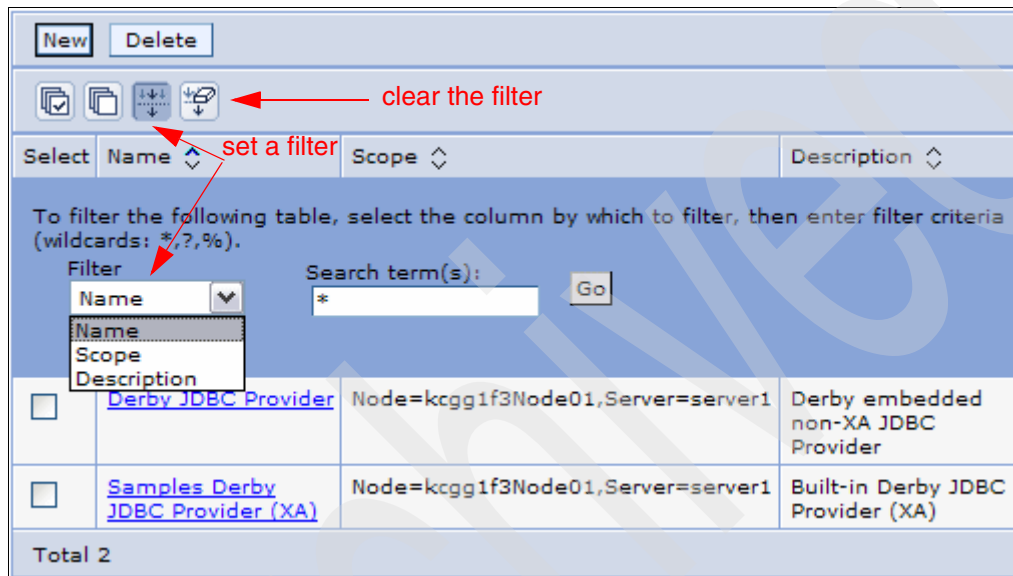


Figure 4-5 Setting filters and preferences

When you click the icon, a new area will appear at the top of the table allowing you to enter filter criteria. To filter entries, do the following:

1. Select the column to filter on. For example, in Figure 4-5, the display table has three columns to choose from. Your options vary depending on the type of item you are filtering.
2. Enter the filter criteria. The filter criteria is case sensitive and wild cards can be used. In our example, to see only providers with names starting with “S”, select the Name column to filter on and enter S* as the filter.
3. Click **Go**.
4. Once you have set the filter, click the **Show Filter** Icon again to remove the filter criteria from view. You still have a visual indication that the filter is set at the top of the table.

Setting the filter is temporary and only lasts for as long as you are in that collection. To keep the filter active for that collection, check the **Retain filter criteria** box in the Preferences section and click **Apply**. To clear the filter criteria, click the  icon.

The Preferences settings also allow you to specify the maximum number of rows to display per page.

Tip: For help on filtering, see:

- ▶ The *Administrative console buttons* topic in the Information Center.
- ▶ Click the Help item in the Task bar and select the **Administrative Console Buttons** topic under the Core Console heading.

4.1.6 Updating existing items

To edit the properties of an existing item, complete these tasks:

1. Select the category and type in the navigation tree. For example, select **Servers** → **Application Servers**.
2. A list of the items of that type in the scope specified will be listed in a collection table in the workspace area. Click an item in the table. This opens a detail page for the item.
3. In some cases, you see a Configuration tab and a Runtime tab on this page. In others, you only see a Configuration tab. Updates are done under the Configuration tab. Specify new properties or edit the properties already configured for that item. The configurable properties will depend on the type of item selected.

For example, if we click an application server, this opens a page resembling Figure 4-6 on page 152.

The screenshot shows the 'Configuration' tab of the application server configuration page. It includes sections for General Properties, Container Settings, Applications, and Server messaging. The General Properties section contains fields for Name and Node Name, and checkboxes for Run in development mode and Parallel start. The Container Settings section lists various container settings like Session management, SIP, Web, Portlet, EJB, Container Services, and Business Process Services. The Applications section has a link for Installed applications. The Server messaging section has links for Messaging engines, Messaging engine inbound transports, and WebSphere MQ link inbound transports. At the bottom, there are buttons for Apply, OK, Reset, and Cancel.

Figure 4-6 Editing application server properties

The detail page provides fields for configuring or viewing the more common settings and links to configuration pages for additional settings.

4. Click **OK** to save your changes to the workspace and exit the page. Click **Apply** to save the changes without exiting. The changes are still temporary. They are only saved to the workspace, not to the master configuration. This still needs to be done.
5. As soon as you save changes to your workspace, you will see a message in the Messages area reminding you that you have unsaved changes. See Figure 4-7.

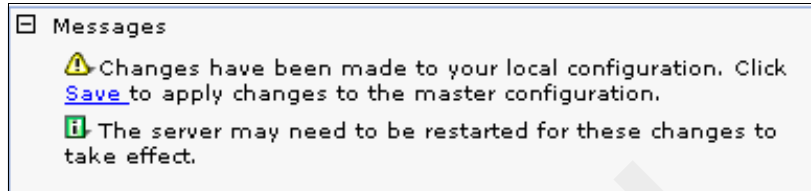


Figure 4-7 Save changes to the master repository

At intervals during your configuration work and at the end, you should save the changes to the master configuration. You can do this by clicking **Save** in the message, or by selecting **System administration** → **Save Changes to Master Repository** in the navigation tree.

To discard changes, use the same options. These options simply display the changes you have made and give you the opportunity to save or discard.

4.1.7 Adding new items

To create new instances of most item types, complete these tasks:

1. Select the category and type in the navigation tree. See Figure 4-8.
2. Select **Scope**. (To create a new item, you cannot select the **All** option for scope.)
3. Click the **New** button above the collection table in the workspace.

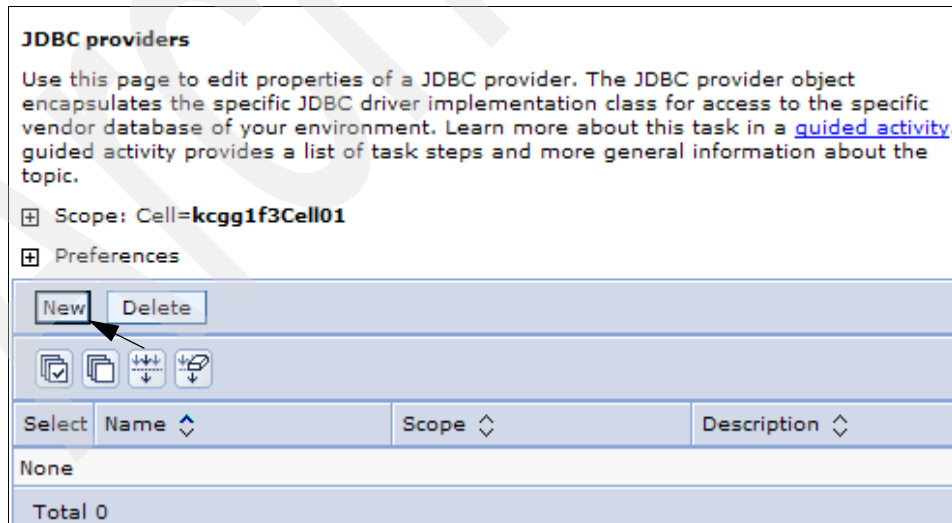


Figure 4-8 Create a new item

You might be presented with a wizard prompting you to enter information and taking you through the process. Proceed until all the required properties are entered.

Note: In the configuration pages, you can click **Apply** or **OK** to store your changes in the workspace. If you click **OK**, you will exit the configuration page. If you click **Apply**, you will remain in the configuration page. As you are becoming familiar with the configuration pages, we suggest that you always click **Apply** first. If there are additional properties to configure, you will not see them if you click **OK** and leave the page.

4. Click **Save** in the task bar or in the Messages area when you are finished.

4.1.8 Removing items

To remove an item, complete these tasks:

1. Find the item.
2. Select the item in the collection table by checking the box next to it.
3. Click **Delete**.
4. If asked whether you want to delete it, click **OK**.
5. Click **Save** in the taskbar or in the Messages area when you are finished.

For example, to delete an existing JDBC provider, select **Resources** → **JDBC Providers**. Check the provider you want to remove and click **Delete**.

4.1.9 Starting and stopping items

To start or stop an item using the console:

1. Select the item type in the navigation tree.
2. Select the item in the collection table by checking the box next to it.
3. Click **Start** or **Stop**. The collection table shows the status of the item. See Figure 4-9 on page 155.

For example, to start an application server in a distributed server environment, select **Servers** → **Application Servers**. Place a check mark in the check box beside the application server you want and click **Start**.

Application servers

Use this page to view a list of the application servers in your environment and the status of each of these servers. You can also use this page to change the status of a specific application server.

⊞ Preferences

☐ ☐ ☐ ☐

Select	Name ↕	Node ↕	Version ↕	Cluster Name ↕	Status ↕
☐	server1	kadw028Node03	ND 6.1.0.0		
☒	server2	kadw028Node03	ND 6.1.0.0		

Total 2

Figure 4-9 Starting and stopping items

You can start and stop the following items from the administrative console:

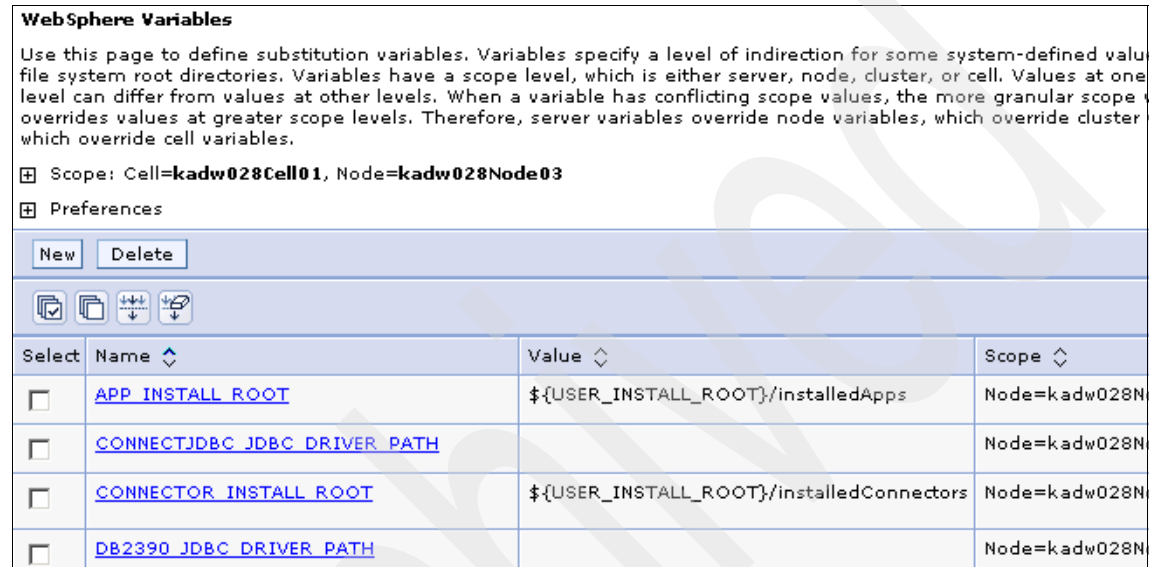
- ▶ Applications
- ▶ Application, Web, and generic servers
- ▶ Clusters: Starting or stopping a cluster will start or stop all the servers in the cluster.
- ▶ Nodes: Stop only.
- ▶ Node agents: Stop or recycled only.
- ▶ Deployment manager: Stop only. This kills the console. It does not stop any of the node agents or the application servers running under those node agents.

4.1.10 Using variables

WebSphere variables are name and value pairs used to represent variables in the configuration files. This makes it easier to manage a large configuration.

To set a WebSphere variable:

1. Click **Environment** → **WebSphere Variables**. See Figure 4-10.



The screenshot shows the 'WebSphere Variables' configuration page. At the top, there is a title 'WebSphere Variables' and a descriptive paragraph: 'Use this page to define substitution variables. Variables specify a level of indirection for some system-defined values, such as file system root directories. Variables have a scope level, which is either server, node, cluster, or cell. Values at one level can differ from values at other levels. When a variable has conflicting scope values, the more granular scope value overrides values at greater scope levels. Therefore, server variables override node variables, which override cluster variables, which override cell variables.' Below this, there are expandable sections for 'Scope: Cell=kadw028Cell01, Node=kadw028Node03' and 'Preferences'. A toolbar contains 'New' and 'Delete' buttons, along with icons for copy, paste, and undo. The main area is a table with four columns: 'Select', 'Name', 'Value', and 'Scope'. It lists four variables: 'APP_INSTALL_ROOT' with value '\${USER_INSTALL_ROOT}/installedApps', 'CONNECTJDBC_JDBC_DRIVER_PATH', 'CONNECTOR_INSTALL_ROOT' with value '\${USER_INSTALL_ROOT}/installedConnectors', and 'DB2390_JDBC_DRIVER_PATH'. All variables are currently set to 'Node=kadw028Node03' scope.

Select	Name	Value	Scope
☐	APP_INSTALL_ROOT	<code>\${USER_INSTALL_ROOT}/installedApps</code>	Node=kadw028Node03
☐	CONNECTJDBC_JDBC_DRIVER_PATH		Node=kadw028Node03
☐	CONNECTOR_INSTALL_ROOT	<code>\${USER_INSTALL_ROOT}/installedConnectors</code>	Node=kadw028Node03
☐	DB2390_JDBC_DRIVER_PATH		Node=kadw028Node03

Figure 4-10 WebSphere variables

2. To add a new variable, click **New**, or click a variable name to update its properties.
3. Enter a name and value and click **Apply**. See Figure 4-11.

The screenshot shows a 'Configuration' window with a 'General Properties' tab. It contains three input fields: 'Name' with the value 'DB2_JDBC_DRIVER_PATH', 'Value' with 'c:\sqlib\java', and 'Description' with 'The directory that contains the DB2 JDBC Driver.' At the bottom are buttons for 'Apply', 'OK', 'Reset', and 'Cancel'.

Figure 4-11 New WebSphere variable

4.1.11 Saving work

As you work with the configuration, your changes are saved to temporary workspace storage. For the configuration changes to take effect, they must be saved to the master configuration. If you have a distributed server environment, a second step is required to *synchronize*, or send, the configuration to the nodes. Consider the following:

1. If you work on a page, and click **Apply** or **OK**, the changes are saved in the workspace under your user ID. This allows you to recover changes under the same user ID if you exit the session without saving.
2. You need to save changes to the master repository to make them permanent. This can be done from the Navigation tree by selecting **System administration** → **Save Changes to Master Repository** from the Messages area, or when you log in if you logged out without saving the changes.
3. The Save window presents you with the following options:
 - Save
 - Discard

Discard reverses any changes made during the working session and reverts to the master configuration.

- Cancel

Cancel does not reverse changes made during the working session. It just cancels the action of saving to the master repository for now.

- Synchronize changes with nodes

This distributes the new configuration to the nodes in a distributed server environment.

Before deciding whether you want to save or discard changes, you can see the changed items by expanding **Total changed documents** in the Save window.

Important: All the changes made during a session are cumulative. Therefore, when you decide to save changes to the master repository, either at logon or after clicking **Save** on the taskbar, all changes are committed. There is no way to be selective about what changes are saved to the master repository.

4. When you are finished, log out of the console using the **Logout** option on the taskbar.

4.1.12 Getting help

To access help, do the following:

1. Use the **Help** menu in the taskbar. This opens a new Web browser with online help for the administrative console. It is structured by administrative tasks. See Figure 4-12 on page 159.

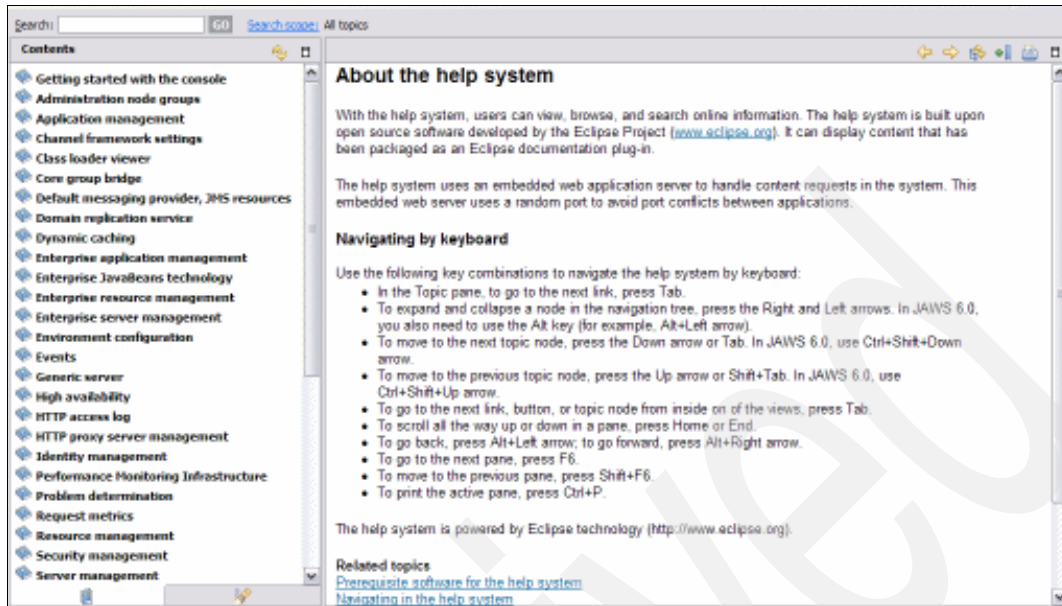


Figure 4-12 Online help

2. With the option **Show the help portlet** enabled, you can see the Help window in the workspace.
3. The Information Center can be viewed online or downloaded from:
<http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp>

4.2 Securing the administrative console

WebSphere Application Server provides the ability to secure the administrative console so only authenticated users can use it. Note that enabling administrative security does not enable application security.

With V6.1, you can enable administrative security during installation and profile creation. If you do this on distributed systems, you will automatically get a file-based user registry populated with an administrative user ID of your choosing. This registry can later be federated with the registry you choose for application security. On z/OS platforms, you will have the option of using the file-based registry or the z/OS system's SAF-compliant security database.

You can enable administrative security after profile creation through the administrative console by navigating to **Security** → **Secure administration, applications, and infrastructure**. Doing this allows you more flexibility in specifying security options.

Before enabling any type of security, you will need to complete the configuration items for authentication, authorization, and realm (user registry). You will also need to populate the chosen user registry with at least one user ID to be used as an administrator ID.

The screenshot shows the 'Security Configuration Wizard' interface. It has two tabs: 'Security Configuration Wizard' (active) and 'Security Configuration Report'. The main content area is divided into several sections:

- Administrative security:** Contains a checked checkbox 'Enable administrative security' and two links: 'Administrative User Roles' and 'Administrative Group Roles'.
- Application security:** Contains an unchecked checkbox 'Enable application security'.
- Java 2 security:** Contains three checkboxes: 'Use Java 2 security to restrict application access to local resources' (unchecked), 'Warn if applications are granted custom permissions' (checked), and 'Restrict access to resource authentication data' (unchecked).
- User account repository:** Contains a text field for 'Current realm definition' with the value 'Local operating system'. Below it is a dropdown for 'Available realm definitions' also showing 'Local operating system'. At the bottom of this section are 'Configure' and 'Set as current' buttons.
- Authentication:** Contains an unchecked checkbox 'Use domain-qualified' and several expandable sections with plus icons: 'Web security', 'RMI/IIOP security', and 'Java Authentication and Authorization'. Below these are links for 'Authentication mechanism', 'External authorization provider', and 'Custom properties'.

Figure 4-13 Enabling administrative security

Attention: Beware that when you check the box to enable administrative security, the application security and Java 2 security check boxes are enabled automatically. If you are not prepared to use Java 2 or application security at this time, be sure to uncheck the boxes.

Administrative security is based on identifying users or groups that are defined in the active user registry and assigning roles to each of those users. When you log in to the administrative console, you must use a valid administrator user ID and password. The role of the user ID determines the administrative actions the user can perform.

Fine-grained administrative security (new):

In releases prior to WebSphere Application Server Version 6.1, users granted administrative roles could administer all of the resource instances under the cell. With V6.1, administrative roles are now per resource instance rather than to the entire cell. Resources that require the same privileges are placed in a group called the authorization group. Users can be granted access to the authorization group by assigning to them the required administrative role within the group.

A cell-wide authorization group for backward compatibility: Users assigned to administrative roles in the cell-wide authorization group can still access all of the resources within the cell.

- ▶ **Administrator**
The administrator role has operator permissions, configurator permissions, and the permission required to access sensitive data, including server password, Lightweight Third Party Authentication (LTPA) password and keys, and so on.
- ▶ **Configurator**
The configurator role has monitor permissions and can change the WebSphere Application Server configuration.
- ▶ **Operator**
The operator role has monitor permissions and can change the run time state. For example, the operator can start or stop services.
- ▶ **Monitor**
The monitor role has the least permissions. This role primarily confines the user to viewing the WebSphere Application Server configuration and current state.
- ▶ **Deployer**
The deployer role is only available for **wsadmin** users, not for administrative console users. Users granted this role can perform both configuration actions and run time operations on applications.
- ▶ **AdminSecurityManager**
The AdminSecurityManager role is only available for **wsadmin** users, not for administrative console users. When using **wsadmin**, users granted this role can map users to administrative roles. When fine grained administrative security is used, users granted this role can manage authorization groups.

► Iscadmins

The iscadmins role has administrator privileges for managing users and groups from within the administrative console only.

Role assignments are managed through the administrative console. Navigate to **Users and groups** → **Administrative User Roles** or **Users and groups** → **Administrative Group Roles**.

If you are using a file-based repository, you can add users and groups through the console by navigating to **Users and groups** → **Manage Users** or **Users and groups** → **Manage Groups**.

After saving the configuration, you must restart the application server in a stand-alone server environment or the deployment manager in a distributed server environment.

The next time you log in to the administrative console, you must authenticate with one of the users that were identified as having an administrative role. Entering commands from a command window will also prompt you for a user ID and password.

4.3 Working with the deployment manager

This section will provide information about how to manage the deployment manager and will introduce you to the configuration settings associated with it.

4.3.1 Deployment manager configuration settings

A deployment manager is created by creating a deployment manager profile. Once created, there usually is not much that you need to do. However, it is good to note that there are settings that you can modify from the administration tools. This section gives you a brief look at these settings.

To view the deployment manager from the administrative console, select **System Administration** → **Deployment manager**. You have two pages available, the Runtime page and the Configuration page. Figure 4-14 on page 163 shows the Configuration page.

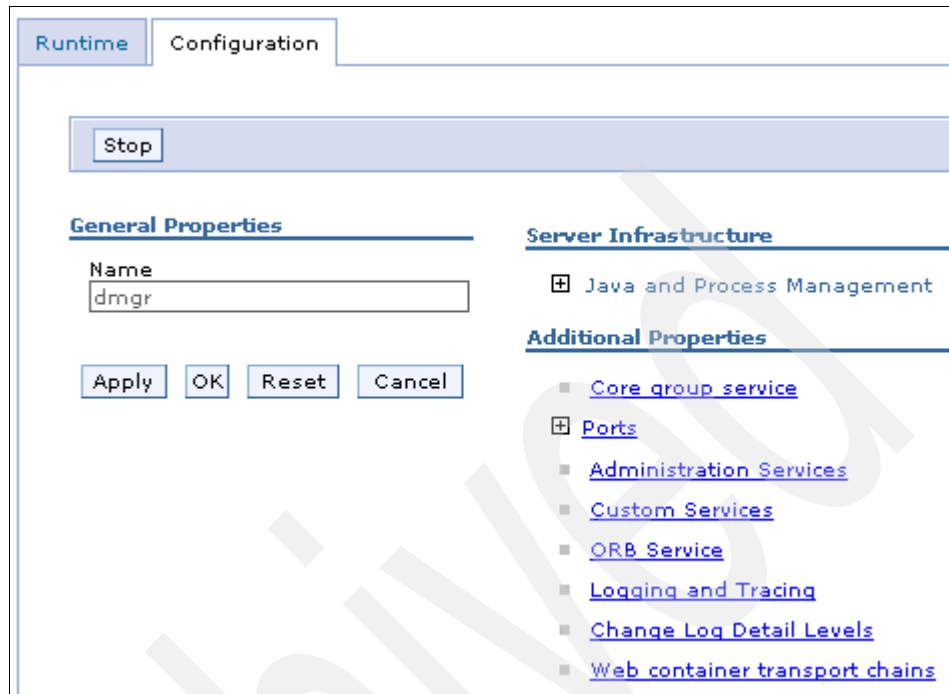


Figure 4-14 Deployment manager configuration

Configuration tab

Because it is unlikely that you will need to change most of these settings, we only give you a brief description here of the settings you can configure.

Java and process management

The Java and process management settings allow you to define how the deployment manager process is initialized. The only category of settings under this group is the process definition settings. These include:

- ▶ JVM settings, including heap size, class path and boot class path, and verbose settings for garbage collection, class loading, and JNI
- ▶ Environment entries consisting of name/value pairs that define custom properties
- ▶ Process execution properties (not used on Windows) that allow you to define process priority, run as settings, file permission mode mask, and process group assignment (for processor partitioning)
- ▶ Process log settings for stdout and stderr logs

Core group service

A *core group* is a set of processes that participate in providing high availability function to each other. In a distributed server environment, there is one default core group automatically defined called DefaultCoreGroup. The deployment manager is automatically added to this core group. New core groups can be defined and the servers can be moved from one core group to another.

The core group settings allow you to modify core group settings related to the deployment manager.

For more information about high availability and using core groups, see *WebSphere Application Server Network Deployment V6: High Availability Solutions*, SG24-6688.

Ports

The port settings allow you to modify the TCP/IP port settings used for the deployment manager process. These settings were first defined when the deployment manager profile was created.

Administration services

These settings allow you to configure properties related to the administrative services. These include:

- ▶ Repository service settings used to enable auditing.
- ▶ Existing JMX connectors (RMI and SOAP). This allows you to update, add HTTP and JMS connectors, or remove connectors.
- ▶ Mbean extensions. This allows you to add new extensions in order to manage new types of resources.
- ▶ Custom properties consisting of name/value pairs.
- ▶ Web server plug-in automation.

Custom services

Custom services settings provide an extension point for configuration data for plug-in services. This allows you to add in custom code that will be executed during process initialization.

ORB service

These settings allow you to specify settings for the Object Request Broker service.

Change log level details

These settings allow you to control which events are processed by Java logging.

Logging and tracing

Log settings are available for the following logs:

- ▶ Diagnostic trace
- ▶ JVM logs
- ▶ Process logs
- ▶ IBM Service logs
- ▶ Change log detail levels

Web container transport chains

Transport chains represent network protocol stacks operating within a client or server. These settings give you access to transport chain definitions. For information about transport chains, see “Working with nodes” on page 201.

Deployment manager Runtime tab

In addition to the Configuration page, the administrative console contains a Runtime page for the deployment manager. To view the Runtime page, select **System Administration** → **Deployment manager** and click the **Runtime** tab at the top of the page. Figure 4-15 on page 165 shows the Runtime tab.

Deployment manager

Use this page to stop the deployment manager from running, and to link to other pages which you can use to define additional properties for the deployment manager. The deployment manager provides a single, central point of administrative control for all elements of the WebSphere(R) Application Server distributed cell.

Runtime Configuration

General Properties

Process Id
3696

Cell Name
kadw028Cell01

Node Name
kadw028CellManager01

State
Started

Back

Troubleshooting

☐ Diagnostic Provider service

- [Tests](#)
- [State Data](#)
- [Configuration Data](#)

Figure 4-15 Deployment manager run time page

The fact that the state is Started does not mean much, because you would not be able to access the administrative console otherwise.

The Diagnostic Provider framework is a new feature in V6.1. It allows you to query components for current configuration data, state data, and to run a self-diagnostic test routine. Not all components have a diagnostic provider in this release.

4.3.2 Starting and stopping the deployment manager

The deployment manager must be started and stopped with commands. The administrative console is not available unless it is running.

On a Windows system, you have the option of registering the deployment manager as a Windows service. In order to have it registered, you must select this option when you create the deployment manager profile or register it later using the **WASService** command (see 3.6.3, “Enabling process restart on failure” on page 130).

On Windows you also have the option of starting and stopping the deployment manager using the Start menu. Select the following:

- ▶ **Start → Programs → IBM WebSphere → WebSphere Application Server Network Deployment V6.1 → Profiles → <profile_name> → Start the deployment manager**
- ▶ **Start → Programs → IBM WebSphere → WebSphere Application Server Network Deployment V6.1 → Profiles → <profile_name> → Stop the deployment manager**

Starting the deployment manager with startManager

Using the **startManager** command is the most common way to start the deployment manager, as shown in Example 4-2.

Example 4-2 startManager command

```
c:\>cd <was_home>\profiles\<dmgr_profile>\bin
C:\<was_home>\profiles\<dmgr_profile>\bin>startManager

ADMU7701I: Because dmgr is registered to run as a Windows Service, the request
           to start this server will be completed by starting the associated
           Windows Service.
ADMU0116I: Tool information is being logged in file
           C:\WebSphere\AppServer\profiles\Dmgr01\logs\dmgr\startServer.log
ADMU0128I: Starting tool with the Dmgr01 profile
ADMU3100I: Reading configuration for server: dmgr
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server dmgr open for e-business; process id is 1536
```

Run this command from the deployment manager *<profile_home>/bin* directory. If you run it from the *<was_home>/bin* directory, use the `-profileName` parameter to ensure the command is run against the deployment manager profile.

Syntax of startManager

The syntax of the `startManager` command is:

```
startManager.bat(sh) [options]
```

All arguments are optional. See Table 4-2.

Table 4-2 Options for startManager

Option	Description
-nowait	Do not wait for successful initialization of the deployment manager process.
-quiet	Suppress the printing of progress information.
-logfile <fileName>	Specify a log file location to which information gets written. The default is <i><profile_home>/logs/dmgr/startServer.log</i> .
-profileName <profile>	Specify a profile to run the command against. If the command is run from <i><was_home>/bin</i> and <code>-profileName</code> is not specified, the default profile is used. If it is run from <i><profile_home>/bin</i> , that profile is used.
-trace	Generates trace information into a file for debugging purposes. The output goes to <i>startServer.log</i> .
-script [<script filename>] -background	Generate a launch script instead of starting the server. The script file name is optional. If the file name is not provided, the default script file name is <i>start_dmgr.bat(sh)</i> . The script is saved to the <i><dmgr_profile_home>/bin</i> directory. The <code>-background</code> parameter specifies that the generated script runs in the background.
-timeout <seconds>	Specifies the waiting time before server initialization times out and returns an error.
-statusport <portnumber>	Set the port number for server status callback.
-replacelog	Replace the log file instead of appending to the current log.
-J-<java option>	Options are to be passed through to the Java interpreter. Options are specified in the form: <code>-D<name>=<value></code> .

Option	Description
-help or -?	Prints the command syntax to the console.

Starting the deployment manager on z/OS (START command)

On z/OS, the deployment manager can be started using a JCL start procedure. The exact command can be found in the BBOCCINS instruction member of the JCL generated to create the profile.

For example:

```
START CHMGCR,JOBNAME=CHDMGR,ENV=CHCELL.CHDMNODE.CHDMGR
```

Where:

- ▶ CHMGCR is the JCL start procedure.
- ▶ CHDMGR is the Job name.
- ▶ ENV is the concatenation of the cell short name, node short name, and server short name.

Starting the deployment manager will start the following:

- ▶ A daemon. In our example, named CHDEMNI. There will be one daemon per cell per MVS image. One of the functions of the daemon server is to provide the "location name service" for the cell. All daemons in the cell are fully aware of all the objects in the cell and use the same port values.
- ▶ A controller region. In our example, named CHDMGR. The controller region serves many functions, including acting as the endpoint for communications.
- ▶ A servant region. In our example, named CHDMGRS. The servant region contains the JVM where the applications are run.
- ▶ If you are using messaging, you will also see a control region adjunct server start.

Stopping the deployment manager

The deployment manager is stopped with the **stopManager** command, as shown in Example 4-3.

Example 4-3 stopManager command

```
c:\>cd <was_home>\profiles\<dmgr_profile>\bin
C:\<was_home>\profiles\<dmgr_profile>\bin>stopmanager
```

```
ADMU7702I: Because dmgr is registered to run as a Windows Service, the request
           to stop this server will be completed by stopping the associated
           Windows Service.
```

```
ADMU0116I: Tool information is being logged in file
```

```

C:\WebSphere\AppServer\profiles\Dmgr01\logs\dmgr\stopServer.log
ADMU0128I: Starting tool with the Dmgr01 profile
ADMU3100I: Reading configuration for server: dmgr
ADMU3201I: Server stop request issued. Waiting for stop status.
ADMU4000I: Server dmgr stop completed.

```

Syntax of stopManager

The syntax of the **stopManager** command is:

```
stopManager.bat(sh) [options]
```

All arguments are optional. See Table 4-3.

Table 4-3 Options for stopManager

Option	Description
-nowait	Do not wait for successful shutdown of the deployment manager process.
-quiet	Suppress the printing of progress information.
-logfile <fileName>	Specify the location of the log file to which information is written. The default is <profile_home>/logs/dmgr/startServer.log.
-profileName <profile>	Specify the profile against which to run the command. If the command is run from <was_home>/bin and -profileName is not specified, the default profile is used. If run from <profile_home>/bin, that profile is used.
-trace	Generate trace information into a file for debugging purposes. The output goes to stopServer.log.
-timeout <seconds>	Specify the waiting time before server shutdown times out and returns an error.
-statusport <portnumber>	Set the port number for server status callback.
-replacelog	Replace the log file instead of appending to the current log.
-username <name>	Specify the user name for authentication if security is enabled in the server.
-password <password>	Specify the password for authentication if security is enabled.
-conntype <type>	Specify the JMX connector type to use for connecting to the deployment manager. Valid types are SOAP or RMI.
-port <portNumber>	Specify the deployment manager JMX port to use explicitly, so that you can avoid reading the configuration files to obtain information.
-help or -?	Print the command syntax to the console.

Stopping the deployment manager on z/OS (STOP command)

To stop the deployment manager with a STOP command, use the following format:

```
STOP dmgr_JOBNAME
```

For example:

```
STOP CHDMGR
```

Stopping the daemon server will stop all servers for that cell, and all the servers on that daemon instance's MVS image will be stopped in an order fashion. For example:

```
STOP CHDEMNI
```

4.4 Working with application servers

This section discusses the following topics:

- ▶ Creating an application server
- ▶ Viewing the status of an application server
- ▶ Starting an application server
- ▶ Stopping an application server
- ▶ Viewing run time attributes of an application server
- ▶ Customizing application servers

Server types: This section uses the following terms.

- ▶ A *stand-alone application server* is an application server created through the use of an application server profile and is not federated to a cell. This is the only option in the Base and Express environments. You can also create a stand-alone application server in the Network Deployment package. However, the expectation is that you will federate the application server to a cell for centralized management in the future.
- ▶ A *managed application server* is one that is managed from a deployment manager. This is only possible with the Network Deployment package. A managed server can either be an application server that was created using an application server profile and subsequently federated to the cell, or it can be created directly from the deployment manager's administrative console.

4.4.1 Creating an application server

The process to create an application server depends on your WebSphere Application Server package.

Stand-alone application servers

Stand-alone application servers are created by creating an application server profile. This results in a profile that defines one stand-alone application server called server1. This application server hosts the sample applications and the administrative console application. During the Profile creation wizard, you have the option of registering the new application server as a Windows service.

For information about creating an application server profile, see 3.3.2, “Creating an application server profile” on page 67.

Managed application servers

In a Network Deployment distributed server environment, you can create an application server from the deployment manager administrative console. The following directions assume that you have created a deployment manager profile and have started the deployment manager.

Note: If you are creating an application server with the intention of adding it to a cluster, using the **Servers** → **Cluster** menu options is more efficient. See 4.6, “Working with clusters” on page 222.

To create an application server from the administrative console:

1. Open the deployment manager administrative console.
2. Select **Servers** → **Application Servers**.

3. Click **New**. See Figure 4-16.
4. Select the node for the new server and enter a name for the new server.

The screenshot shows a window titled "Create New Application Server". Inside, it says "Use this page to create a new application server." On the left, a sidebar lists four steps: "Step 1: Select a node" (highlighted with a yellow arrow), "Step 2: Select a server template", "Step 3: Specify server specific properties", and "Step 4: Confirm new server". The main area is titled "Select a node" and contains the instruction "Select the node that corresponds to the server you wish to create." Below this, there is a "Select node" dropdown menu with "kadm028Node03" selected. Underneath, there is a field for "Server name" with "Server3" entered. At the bottom, there are "Next" and "Cancel" buttons.

Figure 4-16 Create an application server: Step 1

Click **Next**.

5. Select a template to use by clicking the appropriate radio button. See Figure 4-17. You have the following options:
 - Default: Standard production server.
 - DeveloperServer: Optimized to developer uses.
 - defaultZOS: This is only available on z/OS platforms and is the only option until you create new templates.

Later, you can also create templates based on existing application servers. (see “Creating a template” on page 174).

The screenshot shows a window titled "Select a server template". It contains a table with two columns: "Select" and "Name". The table has two rows: one for "DeveloperServer" and one for "default". The "default" row is selected with a radio button. To the right of the table, there is a description for each template.

Select	Name	Type	Specifies a description of an application server template.
☐	DeveloperServer	System	This template is optimized to perform well for development uses.
☒	default	System	The WebSphere Default Server Template

Figure 4-17 Create an application server: Step 2

Click **Next**.

6. The options you see on the next window vary depending on the platform. For distributed platforms, you see Figure 4-18. Select the core group from the list. You will only have this option if you have more than one core group defined.
- Check the **Generate Unique Http Ports** box to have unique ports generated for this server.

The screenshot shows a window titled "Create New Application Server". On the left, a vertical sidebar lists four steps: "Step 1: Select a node", "Step 2: Select a server template", "Step 3: Specify server specific properties" (highlighted with a yellow arrow), and "Step 4: Confirm new server". The main area is titled "Specify server specific properties" and contains a checked checkbox for "Generate Unique Http Ports" and a dropdown menu for "Core Group" with "DefaultCoreGroup" selected.

Figure 4-18 Create an application server: Step 3 for distributed systems

For z/OS systems, you will see Figure 4-19.

The screenshot shows a window titled "Create New Application Server". On the left, a vertical sidebar lists four steps: "Step 1: Select a node", "Step 2: Select a server template", "Step 3: Specify server specific properties" (highlighted with a yellow arrow), and "Step 4: Confirm new server". The main area is titled "Specify server specific properties" and contains a checked checkbox for "Generate Unique Ports", a text input field for "Server Specific Short Name", and another text input field for "Server Generic Short Name". At the bottom, there are three buttons: "Previous", "Next", and "Cancel".

Figure 4-19 Create an application server: Step 3 for z/OS

The server specific short name specifies the short name for the server. This is also used as the job name (for example, BBOS002). The generic short name is the short name that is converted to a cluster short name if the server is later used in a cluster.

Click **Next**.

7. A summary window is presented with the options you chose. See Figure 4-20.

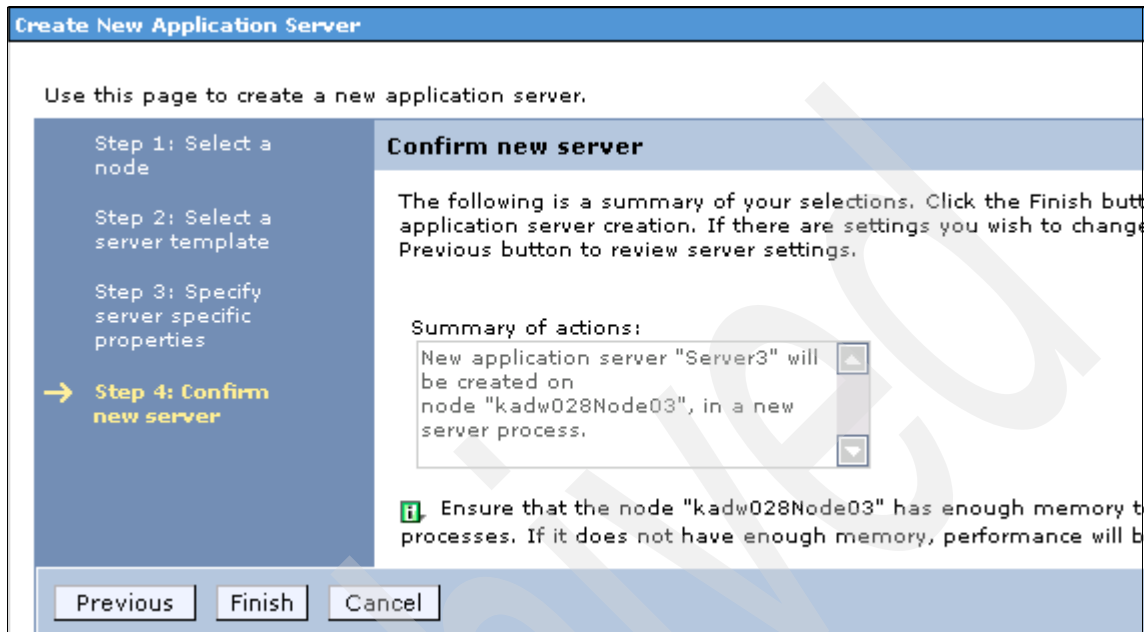


Figure 4-20 Create an application server: Step 4

Click **Finish** to create the server.

8. In the messages box, click **Save** to save the changes to the master repository.

Note: If you are creating an application server on a Windows operating system, this process does not give you the option of registering the new server as a Windows service. You can do this later with the **WASService** command (see 3.6.3, "Enabling process restart on failure" on page 130).

Creating a template

To create an application server template based on an existing server:

1. Select **Servers** → **Application Servers**.
2. Click **Templates...** at the top of the server list.
3. Click **New**.
4. Select a server from the list to build the template from and click **OK**.
5. Enter a name and description for the template and click **OK**.
6. Save your configuration.

The new template will be in the list of templates and available to select the next time you create an application server.

4.4.2 Viewing the status of an application server

Table 4-4 shows a summary of ways to view the status of an application server.

Table 4-4 Methods to view the status of an application server

Method	Server types	Summary
Windows service	Managed and stand-alone	If an application server is registered as a Windows service, then check the Windows services window for its status.
Command line	Managed and stand-alone	To view the status of a stand-alone application server, type: <pre>cd <profile_home>/bin serverStatus(.sh) server1</pre> To view the status of a managed application server, type: <pre>cd <profile_home>/bin serverStatus(.sh) <server_name></pre> To check the status of all servers on the node, type: <pre>cd <profile_home>/bin serverStatus(.sh) -all</pre>
Administrative console	Managed	Select Servers → Application Servers .

Using the administrative console

To check the status of a managed server using the deployment manager's administrative console, the node agent must be started. To use the administrative console, do the following:

1. Select **Servers** → **Application Servers**.

- The servers are listed. The last column to the right contains an icon to indicate the status of each server. Figure 4-21 shows the icons and the corresponding status.

Status
 Started
 Stopped
 Unavailable
 Unknown
 Partial Start
 Partial Stop
 Synchronized
 Not synchronized

Figure 4-21 Status icons

Note: If the server status is Unavailable, the node agent on the node in which the application server is installed is not active. The server cannot be managed from the administrative console unless its node agent is active.

Using the `serverStatus` command

The syntax of the `serverStatus` command is as follows:

```
serverStatus.bat(sh) <server>|-all [options]
```

The first argument is mandatory. The argument is either the name of the server for which status is desired, or the `-all` keyword, which requests status for all servers defined on the node. See Table 4-5 on page 177 for a list of available options.

Table 4-5 Options for serverStatus

Option	Description
-logfile <log file path>	Specify an alternative location for the command's log output, instead of serverStatus.log. The path can be specified in the following forms: absolute, relative, or file name. If the server name is specified, the default location is <profile_home>/logs/<servername>/serverStatus.log. If -all is specified, the default location is <profile_home>/logs/serverStatus.log.
-replacelog	Start a new log, replacing any previous log of the same name. If this argument is not specified, the default behavior is to append the output to the existing file.
-profileName <profile>	Use this profile to run the command against. If the command is run from <was_home>/bin and -profileName is not specified, the default profile is used. If run from <profile_home>/bin, that profile is used.
-trace	Generate trace information into a file for debugging purposes. The output goes to serverStatus.log.
-username <username>	Specify the user name for authentication if WebSphere security is enabled. It is ignored if WebSphere security is disabled.
-password <password>	The password for authentication if WebSphere security is enabled. It is ignored if WebSphere security is disabled.
-help or -?	Prints a usage statement.

Example 4-4 shows an example of using the **serverStatus** command.

Example 4-4 serverStatus example

```
C:\<was_home>\profiles\Node01\bin>serverstatus -all
ADMU0116I: Tool information is being logged in file
           C:\WebSphere\AppServer\profiles\Node01\logs\serverStatus.log
ADMU0128I: Starting tool with the Node01 profile
ADMU0503I: Retrieving server status for all servers
ADMU0505I: Servers found in configuration:
ADMU0506I: Server name: Cserver1
ADMU0506I: Server name: Cserver2
ADMU0506I: Server name: nodeagent
ADMU0506I: Server name: ServerN11
ADMU0506I: Server name: ServerN12
ADMU0509I: The Application Server "Cserver1" cannot be reached. It appears to
           be stopped.
ADMU0509I: The Application Server "Cserver2" cannot be reached. It appears to
           be stopped.
ADMU0508I: The Node Agent "nodeagent" is STARTED
ADMU0509I: The Application Server "ServerN11" cannot be reached. It appears to
           be stopped.
ADMU0509I: The Application Server "ServerN12" cannot be reached. It appears to
           be stopped.
```

4.4.3 Starting an application server

How you start an application server depends largely on personal preference and on whether the application server is stand-alone or managed. Keep in mind that the application server created by an application server profile is always called **server1**. Multiple servers federated in this way are all named **server1**, but reside on different nodes.

Table 4-6 shows the various methods you can use to start an application server.

Table 4-6 Methods to start an application server

Method	Server types:	Summary
Windows service	Managed and stand-alone	Application servers can be registered as a Windows service. You can start the server by starting the service.
First steps menu	Managed and stand-alone	The First Steps menu is located at <code><profile_home>/firststeps/firststeps.bat (.sh)</code> .
Windows Start menu	Managed and stand-alone	Select Start → Programs → IBM WebSphere → Application Server V6.1 → Profiles → <code><profile_name></code> → Start the Server .

Method	Server types:	Summary
Command line	Managed and stand-alone	cd <profile_home>/bin startServer(.sh) server1
Administrative console	Managed	Select Servers → Application Servers . To start a managed server from the administrative console, the node agent must be started.
Administrative console	Clusters	Select Servers → Clusters . Starting a cluster starts each application server in the cluster.
z/OS START command	Managed and stand-alone	START appserver_procname,JOBNAME=server_shortname, ENV=cell_shortname.node_shortname.server_shortname

Using the administrative console to start a managed server

Note: Before managing a server in a distributed server environment using the administrative console, you must make sure that the node agent for the server's node is running. To do this:

1. Select **System Administration** → **Node Agents**.
2. The status of the node agent is in the far right column. If it is not started, you must start it (see 4.5.5, "Starting and stopping nodes" on page 215).

From the administrative console, do the following:

1. Select **Servers** → **Application Servers**.
2. Check the box to the left of each server you want to start.
3. Click **Start**.

If there are any errors, check the log file for the application server process:

<profile_home>/logs/<server_name>/SystemOut.log

Note: By default, all the applications on a server start when the application server starts. To prevent an application from starting, see 4.8.7, "Preventing an enterprise application from starting on a server" on page 234.

Using the startServer command

The syntax of the **startServer** command is as follows:

startServer.bat(sh) <server> [options]

<server> is the name of the server to be started. The first argument is mandatory and case sensitive. The options are listed in Table 4-7.

Table 4-7 Options for startServer

Option	Description
-nowait	Tell the command not to wait for successful startup of the server.
-quiet	Suppress progress information printed to console in normal mode. This option does not affect information written to a file.
-trace	Generate trace information into a file for debugging purposes. The output goes to startServer.log.
-logfile <log file path>	Specify an alternative location for the command's log output instead of startServer.log. The path can be specified in absolute, relative, or file name form. The default location is <profile_home>/logs/startServer.log.
-profileName <profile>	Specify the profile against which to run the command. If the command is run from <was_home>/bin and -profileName is not specified, the default profile is used. If it is run from <profile_home>/bin, that profile is used.
-replacelog	Start a new log, replacing any previous log of the same name. If this argument is not specified, the default behavior is to append output to the existing file.
-script [<script filename>]	Generate a launch script instead of starting the server. The script file name is optional. If the file name is not provided, the default script file name is start_<server>. The script needs to be saved to the bin directory of the node installation.
-username <username>	User name for authentication if WebSphere security is enabled. Ignored if WebSphere security is disabled.
-password <password>	Specify a password for authentication if WebSphere security is enabled. The password is ignored if WebSphere security is disabled.
-timeout <seconds>	Specify the waiting time before server initialization times out and returns an error.

Option	Description
-statusport <portnumber>	Set the port number for server status callback.
-J-<java option>	Specify options to be passed through to the Java interpreter. Options are specified in the form: -D<name>=<value>.
-help or -?	Print a usage statement.

startServer example

Example 4-5 on page 181 shows an example of using the **startServer** command.

Example 4-5 startServer example

```

C:\<was_home>\profiles\<profile_server>\bin>startserver server1
ADMU0116I: Tool information is being logged in file

C:\WebSphere\AppServer\profiles\AppSrv02\logs\server1\startServer.log

ADMU0128I: Starting tool with the AppSrv02 profile
ADMU3100I: Reading configuration for server: server1
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server server1 open for e-business; process id is
2548

```

4.4.4 Stopping an application server

How you stop an application server depends largely on personal preference and on whether the application server is stand-alone or managed. Keep in mind that the application server created by a application server profile is always called **server1**.

Table 4-8 shows several methods to stop an application server.

Table 4-8 Methods to stop an application server

Method	Server types:	Summary
Windows service	Managed and stand-alone	Application servers can be registered as a Windows service. You can stop the server by stopping the service.
First steps menu	Managed and stand-alone	The First Steps menu is located at <code><profile_home>/firststeps/firststeps.bat (.sh)</code> .
Windows Start menu	Managed and stand-alone	For a standalone application server, do the following: Select Start → Programs → IBM WebSphere → Application Server V6.1 → Profiles → <profile_name> → Stop the Server. For a stand-alone or managed application server on a Network Deployment system, do the following: Select Start → Programs → IBM WebSphere → Application Server Network Deployment V6.1 → Profiles → <profile_name> → Stop the Server.
Command line	Managed and stand-alone	For a stand-alone application server: <pre>cd <profile_home>/bin stopServer(.sh) server1</pre> For a managed application server: <pre>cd <profile_home>/bin stopServer(.sh) <server_name></pre>
Administrative console	Managed	Select Servers → Application Servers. To stop a managed server from the administrative console, the node agent must be started.
Administrative console	Managed	Select System Administration → Node Agents → Restart all Servers on the Node. This restarts all the servers on the node.
z/OS STOP command	Managed and standalone	STOP appserver_JOBNAME

Using the administrative console to stop a managed server

Note: These directions assume the node agent for the application server is running.

From the administrative console, you have the following options to stop an application server:

- ▶ Stop quiesces the application server and stops it.
- ▶ Immediate Stop stops the server, but bypasses the normal server quiesce process that supports in-flight requests to complete before shutting down the entire server process. This shutdown mode is faster than the normal server stop processing, but some application clients can receive exceptions.
- ▶ Terminate deletes the application server process. Use this if immediate stop fails to stop the server.

From the administrative console, do the following to stop an application server:

1. Select **Servers** → **Application Servers**.
2. Check the box to the left of each server you want to stop.
3. Click the appropriate stop option.

If there are any errors, check the log file for the application server process:

`<profile_home>/logs/<server_name>/SystemOut.log`

Restarting all servers on a node

If you want to stop, and then restart, all the application servers on a node, you can do the following from the administrative console:

1. Select **System Administration** → **Node Agents**.
2. Check the box to the left of the node agent.
3. Click **Restart all Servers on the Node**.

Restarting all servers in a cluster

If you want to stop, and then restart, all the servers in a cluster, you can do the following from the administrative console:

1. Select **Servers** → **Clusters**.
2. Check the box to the left of the cluster.
3. Click **Ripplestart**.

Using the stopServer command

The syntax of the **stopServer** command is:

`stopServer.bat(sh) <server> [options]`

<server> is the name of the server to be started. The first argument is mandatory and is case sensitive. The options are listed in Table 4-9.

Table 4-9 stopServer command options

Option	Description
-nowait	Tells the command not to wait for the successful stop of the server.
-quiet	Suppress progress information printed to console in normal mode. This option does not affect information written to file.
-trace	Generate trace information into a file for debugging purposes. The output is to stopServer.log.
-logfile <log file path>	Specify an alternative location for the command's log output, instead of stopServer.log. The path can be specified in the following forms: absolute, relative, or file name.
-profileName <profile>	Specify the profile to run the command against. If the command is run from <was_home>/bin and -profileName is not specified, the default profile is used. If run from <profile_home>/bin, that profile is used.
-replacelog	Start a new log, replacing any previous log of the same name. If this argument is not specified, the default behavior is to append the output to the existing file.
-timeout <seconds>	Specify the waiting time before server initialization times out and returns an error.
-conntype <connector type>	Specify the type of JMX connector to use for connection to the deployment manager. Valid values are SOAP or RMI. If not specified, SOAP is assumed.
-port <portnumber>	The server JMX port is used explicitly, so that configuration files do not have to be read to obtain the information.
-statusport <portnumber>	Set the port number for server status callback.
-username <username>	Specify the user name for authentication if WebSphere security is enabled. Ignore the user name if WebSphere security is disabled.
-password <password>	Specify a password for authentication if WebSphere security is enabled. Ignore the password if WebSphere security is disabled.
-help or -?	Print a usage statement.

Table 4-6 shows an example of the **stopServer** command

Example 4-6 stopServer command example

```
C:\<was_home>\profiles\Node01\bin>stopServer ServerN11

ADMU0116I: Tool information is being logged in file
           C:\WebSphere\AppServer\profiles\Node01\logs\ServerN11\stopServer.log
ADMU0128I: Starting tool with the Node01 profile
ADMU3100I: Reading configuration for server: ServerN11
ADMU3201I: Server stop request issued. Waiting for stop status.
ADMU4000I: Server ServerN11 stop
           completed.
```

4.4.5 Viewing run time attributes of an application server

To view run time attributes, do the following:

1. Select **Servers** → **Application Servers** to display the list of servers.
2. Click the server name to access the detail page.

3. If the server is running, you will see both a Configuration tab and Runtime tab. If it is not running, you will see only a Configuration tab. Click the **Runtime** tab. Figure 4-22 on page 186 shows the Runtime tab and the information it provides.

The screenshot shows the 'Application servers' management console. The breadcrumb is 'Application servers > server1'. Below this is a description: 'Use this page to configure an application server. An application server is a server provides services required to run enterprise applications.' There are two tabs: 'Runtime' (selected) and 'Configuration'. The 'Runtime' tab is divided into three main sections: 'General Properties', 'Server messaging', and 'Additional Properties'. 'General Properties' includes fields for 'Process Id' (2344), 'Cell Name' (kcgg1f3Cell01), 'Node Name' (kcgg1f3Node01), and 'State' (Started), with a 'Back' button at the bottom left. 'Server messaging' includes a link for 'Messaging engines'. 'Troubleshooting' includes a link for 'Diagnostic Provider service'. 'Additional Properties' includes links for 'Transaction Service', 'Performance Monitoring Infrastructure (PMI)', and 'Product Information'.

Application servers > server1	
Use this page to configure an application server. An application server is a server provides services required to run enterprise applications.	
Runtime Configuration	
General Properties	Server messaging
Process Id 2344	■ Messaging engines
Cell Name kcgg1f3Cell01	Troubleshooting
Node Name kcgg1f3Node01	⊕ Diagnostic Provider service
State Started	Additional Properties
Back	■ Transaction Service
	■ Performance Monitoring Infrastructure (PMI)
	■ Product Information

Figure 4-22 Application server Runtime tab

4. From the Runtime tab, you have access to the following:
- A list of *messaging engines* that run on this application server. There will be one messaging engine for each bus that the server is a member of.
 - Access to the Diagnostic Provider service, allowing you to query current configuration data, state, and to initiate diagnostic tests.
 - Transaction Service properties allow you to specify settings for the transaction service. You can change the timeout settings while the server is running, but not the transaction log directory setting.

Runtime Configuration

General Properties

Transaction log directory
 C:\WebSphere\ND\profiles\AppSrv01\tranlog\kadw028Cell01\kadw028Node03\server1\transaction

* Total transaction lifetime timeout
 120 seconds

* Async response timeout
 120 seconds

* Client inactivity timeout
 60 seconds

* Maximum transaction timeout
 300 seconds

Manual transactions - [Review](#)
 0

Retry transactions - [Review](#)
 0

Heuristic transactions - [Review](#)
 0

Imported prepared transactions - [Review](#)
 0

☒ Enable file locking

☒ Enable protocol security

OK Cancel

Figure 4-23 Transaction service options and settings

You can also view or act on transactions in the following states by clicking **Review** to the right of the state. This action is not normally necessary, but in an exceptional situation it might be useful.

- Manual transactions

These transactions await administrative completion. For each transaction, the local or global ID is displayed. You can display each transaction resource and its associated resource manager. You can choose also to commit or rollback transactions in this state.

- Retry transactions

These are transactions with some resources being retried. For each transaction, the local or global ID is displayed, and whether the transaction is committing or rolling back. You can display each transaction resource and its associated resource manager. You can choose also to finish, or abandon retrying, transactions in this state.

- Heuristic transactions

These are transactions that have completed heuristically. For each transaction, the local or global ID and the heuristic outcome is displayed. You can display each transaction resource and its associated resource manager. You can also choose to clear the transaction from the list.

- Imported prepared transactions

Transactions that have been imported and prepared but not yet committed. For each transaction, the local or global ID is displayed. You can display each transaction resource and its associated resource manager. You can also choose to commit or rollback transactions in this state.

- Performance Monitoring Service settings allow you to change the instrumentation levels while the server is running.
- Product Information gives you access to extensive information about the product installation and Fix Pack information.

4.4.6 Customizing application servers

When you create a new application server, it inherits most of its configuration settings from the specified template server. To view or modify these settings, select **Servers** → **Application Servers**. A list of application servers defined in the cell appears in the workspace. Click the name of the application server to make a modification.

This section gives you a quick overview of the types of settings you can customize. See Figure 4-24 on page 189 (not all settings are shown due to the size of the configuration window).

Runtime | **Configuration**

General Properties

Name:

Node Name:

☐ Run in development mode

☒ Parallel start

Access to internal server classes:

Server-specific Application Settings

Classloader policy:

Class loading mode:

Container Settings

- [Session management](#)
- ⊕ SIP Container Settings
- ⊕ Web Container Settings
- ⊕ Portlet Container Settings
- ⊕ EJB Container Settings
- ⊕ Container Services
- ⊕ Business Process Services

Applications

- [Installed applications](#)

Server messaging

- [Messaging engines](#)
- [Messaging engine inbound transport](#)
- [WebSphere MQ link inbound transport](#)
- [SIB service](#)

Server Infrastructure

Figure 4-24 Application server configuration

General properties

The general properties consist of a few items that you can see immediately.

- ▶ **Run in development mode:** Enable this option to streamline the startup time of an application server. Do not enable this setting on production servers.
- ▶ **Parallel start:** Select this field to start the server components, services, and applications on multiple threads. This might shorten the startup time.

The order in which the applications start depends on the weights you assigned to each of them. Applications that have the same weight are started in parallel.

- ▶ **Access to internal server classes:** Specifies whether the applications can access many of the server implementation classes.

- ▶ Application classloader policy and class loading mode: These settings allow you to define an application server-specific classloader policy and class loading mode. Class loaders are discussed in Chapter 12, “Understanding class loaders” on page 795.

SIP container settings

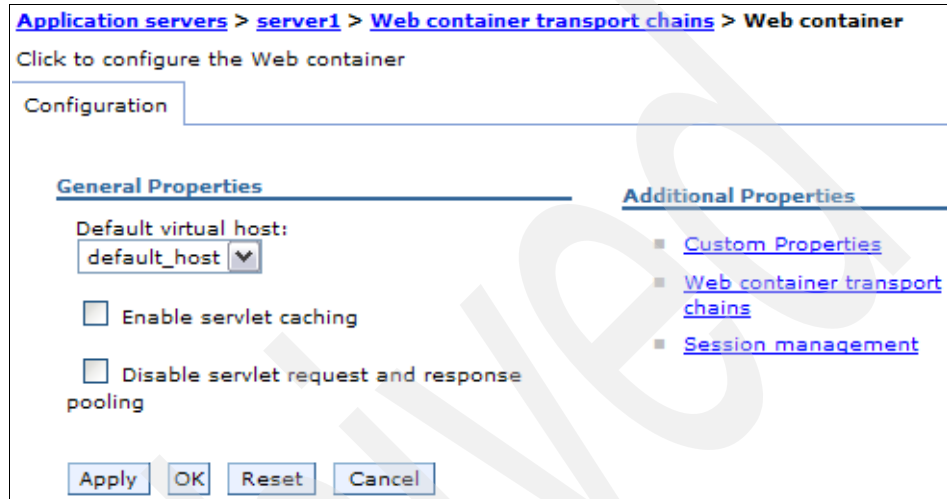
Session Initiation Protocol (SIP) support (new): V6.1 extends the application server to allow it to run SIP applications written to the JSR 116 specification.

Use these items to configure SIP container timers and custom properties.

- ▶ SIP container transport chains: Use this option to manage and create a SIP transport chain. Transport chains represent network protocol stacks operating within a client or server.
- ▶ SIP container: You can use this item to create and manage SIP container timers and custom properties.
 - Maximum application sessions: The maximum number of SIP application sessions that the container manages. When the maximum has been reached, no new SIP conversations are started.
 - Maximum messages per averaging period: Sets the maximum amount of SIP messages per averaging period.
 - Maximum response time: The maximum acceptable response time in milliseconds for an application. After this parameter has been exceeded, the container notifies the clustering framework that it is unavailable.
 - Averaging period: The time period in milliseconds over which averages are calculated.
 - Statistic update rate: The interval at which the container calculates averages and publishes statistics to PMI.
 - Thread pool: The thread pool to use for the SIP container.
 - Custom properties: Specifies additional custom properties for this run time component. Some components use custom configuration properties that can be defined on this option.
 - Session management: Use to configure the session manager that is associated with the Web container and the SIP container.

Web container settings

The Web container serves application requests for servlets and JSPs. The Web container settings allow you to specify the default virtual host, enable servlet caching, specify session manager settings such as persistence and tuning parameters, and HTTP transport properties. See Figure 4-25.



The screenshot shows a web browser window with the address bar displaying the path: [Application servers](#) > [server1](#) > [Web container transport chains](#) > [Web container](#). Below the address bar, there is a text prompt: "Click to configure the Web container". A tab labeled "Configuration" is active. The main content area is divided into two columns. The left column, titled "General Properties", contains three settings: "Default virtual host:" with a dropdown menu showing "default_host", an unchecked checkbox for "Enable servlet caching", and an unchecked checkbox for "Disable servlet request and response pooling". The right column, titled "Additional Properties", contains three links: "Custom Properties", "Web container transport chains", and "Session management". At the bottom of the window, there are four buttons: "Apply", "OK", "Reset", and "Cancel".

Figure 4-25 Web container settings

- Default virtual host: This is the default virtual host to use for applications on the server.
- Enable servlet caching: You can use dynamic cache to improve application performance by caching the output of servlets, commands, and JSPs. This setting allows you to enable dynamic caching for servlets. You must first enable dynamic caching and create the appropriate cache policies in order to use servlet caching.
- Disable servlet request and response pooling: You may want to disable request and response pooling if your application is creating threads inside of the application or if you are concerned about the Web container reusing request and response objects.
- Session management: You can determine how the Web container will manage HTTP session data. This includes settings for the session tracking mechanism (for example, cookies), session timeout, and for the session persistence method. Session management settings are discussed in Chapter 10, "Session management" on page 671.
- Custom Properties: You can specify name/value pairs for configuring internal system properties. Some components can make use of custom configuration properties, which can be defined here. It is not common to

pass information to the Web container this way, but the J2EE specification indicates this as a requirement. Most configuration information can be handled programmatically, or through the deployment descriptor.

- Web container transport chains: Communication to the Web container is handled through the channel framework, which provides a common networking service for WebSphere Application Server components. The channel framework uses a set of configuration settings that describe in layers, how a component communicates to networking ports.

- Port

A port is the component's view of the transport mechanism. A port that uses the channel framework serves as a link between the component and the transport chain.

- Transport chain

A transport chain consists of one or more transport channel types that support a specific I/O protocol.

- Transport channel

A transport channel is specific to an I/O protocol. It contains settings that affect the communication, such as buffer size, timeout settings, TCP/IP port numbers for TCP channels, and other settings required for the communication protocol.

By default, you have four ports, their associated transport chains, and channels defined for a Web container. These are shown in Table 4-10.

Table 4-10 Web container transports

Port	Transport chain	Transport channels
WC_adminhost	WCInboundAdmin ► Enabled ► Host = * ► Port = 9061 ► SSL disabled	TCP Inbound Channel (TCP 1) ► Host = * ► Port = 9061 ► Thread pool=Web container ► Max open connections = 100 ► Inactivity timeout = 60 sec
		HTTP Inbound Channel (HTTP 1) ► Keepalive enabled ► Max persistent requests = 100 ► Read timeout = 60 sec ► Write timeout = 60 sec ► Persistent timeout = 30 sec
		Web Container Inbound Channel (WCC 1) ► Discrimination weight = 1 ► Write buffer size = 32768

Port	Transport chain	Transport channels
WC_adminhost_secure	WCInboundAdminSecure ▶ Enabled ▶ Host = * ▶ Port = 9044 ▶ SSL enabled	TCP Inbound Channel (TCP 1) ▶ Host = * ▶ Port = 9044 ▶ Thread pool=Web container ▶ Max open connections = 100 ▶ Inactivity timeout = 60 sec
		SSL Inbound Channel (SSL 1) ▶ SSL repertoire DMGRNode/DefaultSSLSettings
		HTTP Inbound Channel (HTTP 3) ▶ Keepalive enabled ▶ Max persistent requests = 100 ▶ Read timeout = 60 sec ▶ Write timeout = 60 sec ▶ Persistent timeout = 30 sec
		Web Container Inbound Channel (WCC 1) ▶ Discrimination weight = 1 ▶ Write buffer size = 32768
WC_defaulthost	WCInboundAdminSecure ▶ Enabled ▶ Host = * ▶ Port = 9080 ▶ SSL disabled	TCP Inbound Channel (TCP 2) ▶ Host = * ▶ Port = 9080 ▶ Thread pool=Web container ▶ Max open connections = 20000 ▶ Inactivity timeout = 60 sec
		HTTP Inbound Channel (HTTP 2) ▶ Keepalive enabled ▶ Max persistent requests = 100 ▶ Read timeout = 60 sec ▶ Write timeout = 60 sec ▶ Persistent timeout = 30 sec
		Web Container Inbound Channel (WCC 2) ▶ Discrimination weight = 1 ▶ Write buffer size = 32768

Port	Transport chain	Transport channels
WC_defaulthost_secure	WCInboundDefaultSecure ▶ Enabled ▶ Host = * ▶ Port = 9443 ▶ SSL enabled	TCP Inbound Channel (TCP 4) ▶ Host = * ▶ Port = 9443 ▶ Thread pool=Web container ▶ Max open connections = 20000 ▶ Inactivity timeout = 60 sec
		SSL Inbound Channel (SSL 2) ▶ SSL repertoire DMGRNode/DefaultSSLSettings
		HTTP Inbound Channel (HTTP 4) ▶ Keepalive enabled ▶ Max persistent requests = 100 ▶ Read timeout = 60 sec ▶ Write timeout = 60 sec ▶ Persistent timeout = 30 sec
		Web Container Inbound Channel (WCC 4) ▶ Discrimination weight = 1 ▶ Write buffer size = 32768

TCP channels provide client applications with persistent connections within a Local Area Network (LAN). When configuring a TCP channel, you can specify a list of IP addresses that are allowed to make inbound connections and a list of IP addresses that are not allowed to make inbound connections. You can also specify the thread pool that this channel uses, which allows you to segregate work by the port on which the application server is listening.

HTTP channels are used to enable communication with remote servers. It implements the HTTP 1.0 and 1.1 standards and is used by other channels, such as the Web container channel, to serve HTTP requests and to send HTTP specific information to servlets expecting this type of information.

Web container channels are used to create a bridge in the transport chain between an HTTP inbound channel and a servlet and JavaServer™ Pages™ (JSP™) engine.

SSL channels are used to associate an SSL configuration repertoire with the transport chain. This channel is only available when Secure Sockets Layer (SSL) support is enabled for the transport chain. An SSL configuration repertoire is defined in the security settings in the administrative console.

Portlet container services

Portlet support (new): V6.1 extends the application server to allow it to run JSR 168 compliant portlets.

The portlet container is the run time environment for portlets using the JSR 168 Portlet Specification, in which portlets are instantiated, used, and finally destroyed. The JSR 168 Portlet API provides standard interfaces for portlets. Portlets based on this JSR 168 Portlet Specification are referred to as standard portlets. Use this option to configure the portlet container.

- ▶ **General Properties:** Enable the configure portlet fragment caching to save the output of portlets to the dynamic cache. You must enable the dynamic cache service first.
- ▶ **Additional Properties:** Additional custom properties for this run time component. Some components use custom configuration properties.

EJB container properties

These properties allow you configure the services provided by the EJB container. See Figure 4-26.

[Application servers](#) > [ServerFed](#) > **EJB container**

An EJB container is a component of a J2EE application server that provides runtime services to ejb modules which can be deployed within it.

Configuration

General Properties

* Passivation directory

Inactive pool cleanup interval

Default data source JNDI name

☐ Enable stateful session bean failover using [memory-to-memory replication](#) (Replication domains are defined, but the memory to memory settings have not been selected.)

Initial State

Additional Properties

- [EJB cache settings](#)
- [EJB timer service settings](#)

Figure 4-26 EJB container settings

- ▶ **Passivation Directory:** This attribute provides the directory that you can use to store the persistent state of passivated, stateful session EJBs. If you are using the EJB container to manage session data, you should give WebSphere the ability to swap data to disk when necessary. This directory tells WebSphere where to hold EJB session data when it passivates and activates beans from the pool.
- ▶ **Inactive pool cleanup interval:** Because WebSphere builds a pool of EJBs to satisfy incoming requests, you need to tell it when to remove beans from this pool to preserve resources. This attribute allows you to define the interval at which the container examines the pools of available bean instances to determine if some instances can be deleted to reduce memory usage.

- ▶ Default data source JNDI name: Here you can set a default data source to use for EJBs that have no individual data source defined. This setting is not applicable for EJB 2.x-compliant CMP beans.
- ▶ Initial state: This attribute allows you to identify the state of the container when WebSphere is started. If you have to recycle the application server, this attribute is used to determine whether to start the EJB container at server startup. You would only set this to stopped if you planned on never using the EJB container or EJBs within that specific application server instance.
- ▶ EJB cache settings: You can set up two types of cache settings in WebSphere:
 - Cleanup interval: This attribute allows you to set the interval at which the container attempts to remove unused items from the cache in order to reduce the total number of items in cache to the value we set in the cache size attribute.
 - Cache size: This attribute specifies the number of buckets in the active instance list within the EJB container. This attribute is used by WebSphere to determine how large the cache will be and when to remove components from the cache to reduce its size.
- ▶ EJB timer service settings: Configure and manage the EJB timer service for a specific EJB container.
 - Scheduler type: Specifies a scheduler for the timer service to use.

Container services

The following settings are available under the container services section:

- ▶ Application profiling service: WebSphere Application Server V6 includes a new feature as part of the programming model extensions that provides an extension to access intents. This feature, Application Profiles, lets you identify tasks and access intent to use for a specific task. For information about Application Profiles, refer to the WebSphere Information Center.
Application profiles let you specify externally a set of tasks (a flow of calls in your code), and specify which access intent should be used for a specific task. For information about Application Profiles, refer to the WebSphere Information Center.
- ▶ Transaction service: The transaction service properties allow you to specify settings for the transaction service, as well as manage active transaction locks. The settings include the directory location for the transaction service on the application server to store log files for recovery, the total transaction lifetime timeout, and client inactivity timeout.

When the application server is running, a Runtime tab is available in the Transaction Service properties workspace. From here, you can manage running transactions and modify timeout settings at run time.

- ▶ Dynamic cache service: This page allows you to specify settings for the dynamic cache service of this server.
- ▶ Programming model extensions (PME): These settings are for:
 - Compensation service
 - Internationalization service
 - Object pool service
 - Startup beans service
- ▶ ORB service settings: These settings allow you to specify settings for the Object Request Broker service.

Business process services

The business process settings allow you to manage the following PME features:

- ▶ Activity session service
- ▶ Work area partition service
- ▶ Work area service

Server messaging

The server messaging settings provide configuration settings and information for the messaging services. For information about messaging, see Chapter 8, “Asynchronous messaging” on page 399 and Chapter 9, “Default messaging provider” on page 539.

Server infrastructure

The server infrastructure settings include settings for Java and process management and administration services.

- ▶ Java and Process Management
 - Class loader: Create and configure class loader instances. Class loaders are discussed in Chapter 12, “Understanding class loaders” on page 795.
 - Process definition: You can enhance the operation of an application server, and you can define command-line information for starting or initializing an application server process. These settings define run time properties, such as the program to run, arguments to run the program, and the working directory. Within the process definitions, you will find the JVM definitions, such as the initial and maximum heap sizes, debug options, the process classpath, or different run time options, such as profiler support and heap size.

- Process execution: These include settings such as the process priority, or the user and group that should be used to run the process. These settings are not applicable on the Windows platform.
- Monitoring policy: These properties determine how the node agent will monitor the application server. It includes ping intervals, timeouts, and an initial state setting. These can be used to ensure that the server is started when the node starts and will be restarted in the event of a failure.
- ▶ Administration
 - Custom properties: Specifies additional custom properties for this component.
 - Administration services: This group of settings allows you to specify various settings for administration facility for this server, such as administrative communication protocol settings and timeouts. These settings are not something you would normally be concerned with.
 - Server components: Create an additional run time components that are configurable.
 - Custom Services: Create a custom service classes that run within this server and their configuration properties.

If you plan to extend the administration services by adding custom MBeans, see the *Extending WebSphere Application Server Administrative System with custom MBeans* topic in the Information Center.

Performance

These settings allow you to specify settings for the Performance Monitoring Infrastructure (PMI) and the Runtime Performance Advisor.

Communications

The communications settings include:

▶ Ports

These settings contain the basic port definitions for the server.

You might not ever need to manually change these ports. It is likely, however, that you will want to view these. For example, if you use the **dumpNameSpace** command, you can specify the bootstrap port of the process to dump the name space from. When you federate a node, you will need to know the SOAP connector port of the node or deployment manager. And the inbound communications ports are essential for accessing applications and the administrative console.

Some port settings will be defined to use the channel framework. These will have an associated transport chain. The ports that use the channel

framework include the Web container ports (see “Working with nodes” on page 201), the service integration bus ports (see 9.2.2, “Service integration bus transport chains” on page 563), and the port for Distribution and Consistency Services (DCS) messages.

- ▶ **Message listener service**

The message listener service provides support for WebSphere Application Server V5 message-driven beans applications.

Security

Security settings for the application server allow you to set specific settings at the server level. Security settings are covered in *WebSphere Application Security V6.1 Security Handbook*, SG24-6316.

Troubleshooting

These settings include those for logging and tracing. For information troubleshooting and using these settings, see *WebSphere Application Server V6 Problem Determination for Distributed Platforms*, SG24-6798.

Additional properties

The following settings are defined under the additional properties section:

- ▶ **Class loader viewer service:** Enable or disable service to keep track of classes loaded.
- ▶ **Core group service:** These settings are related to high availability.
- ▶ **Endpoint listeners:** An endpoint listener receives requests from service requester applications within a specific application server or cluster.
- ▶ **Debugging service:** On this page, you can specify settings for the debugging service, to be used in conjunction with a workspace debugging client application, for example, the Application Server Toolkit.
- ▶ **Thread pool:** The thread pool specifies the possible maximum number of concurrently running threads in the Web container. As one thread is needed for every client request, this directly relates to the number of active clients that can possibly access the Web container on this application server at any given time. A timeout value can be specified for the application server to remove threads from the pool based on a timed period of inactivity.

Finally, an option for creating threads beyond the maximum pool size is available. Be careful when using this option. It can have the unexpected effect of allowing the Web container to create more threads than the JVM might be able to process, creating a resource shortage and bringing the application server to a halt.

- Web server plug-in properties: Used to change the HTTP plug-in configuration without having to stop the server and start it again.

4.5 Working with nodes

Managing nodes is a concept specific to a Network Deployment environment. Nodes are managed by the deployment manager through a process known as a *node agent* that resides on each node. In order to manage a node in a Network Deployment environment, the node must be defined and the node agent on each WebSphere Application Server node must be started.

4.5.1 Adding (federating) a node

When you add a node to a cell, the node can be an existing stand-alone application server, or it can be a custom node profile that you have not federated yet.

A custom profile defines a node that can be federated during profile creation, or later using the **addNode** command. For an example of federating a custom profile during profile creation, see 3.3.4, “Creating a custom profile” on page 79. For an example of using **addNode** to federate a custom profile, see 3.3.5, “Federating a custom node to a cell” on page 86.

If you are adding a stand-alone application server installation to a cell, you can do this from the deployment manager administrative console, or you can use the **addNode** command from the node installation. The following examples illustrate using these methods to federate an application server profile to the cell.

Method 1: Using the administrative console

Before you begin, be certain these tasks are completed.

- Make sure the application server is started on the node to be added.
- Open the administrative console for the application server and note the port for the SOAP_CONNECTOR_ADDRESS. You can find this port number by looking in the Communications section in the Details page for the application server.

From the administrative console, do the following to add a node:

1. Select **System Administration** → **Nodes** → **Add Node**.
2. Select **Managed node** and click **Next**. The unmanaged node option is for defining a Web server to the deployment manager (covered later in Chapter 7, “Managing Web servers” on page 365). See Figure 4-27 on page 202.

3. Specify the host name of the node to be added to the cell.
4. Fill in the following fields, as applicable:

Node connection

Host
localhost

JMX connector type
SOAP

JMX connector port
8887

Application server user name

Application server password

Options

☐ Include applications

☐ Include buses

Starting port

☒ Use default

☐ Specify

Port number

Core group name
DefaultCoreGroup

Figure 4-27 Working with nodes

- Host
Specifies the network name of the node to be added to the cell. This value can be an IP address, a domain name server (DNS) name that resolves to an IP address, or the word localhost, if the application server is running on the same machine as the deployment manager. The application server process must be running at the IP address identified by the host field.
- JMX connector type and port
Select the JMX connector type. You can select between SOAP and RMI. If you select SOAP, enter the SOAP_CONNECTOR_PORT number for the

application server. If you select RMI, enter the ORB_LISTENER_ADDRESS number for the application server. These port numbers can be found in serverindex.xml.

- Application server user name

The user ID and password for the application server. If security is enabled at the node you are adding, enter a valid user ID and password to enable the deployment manager to communicate with the remote application server process. *If security is not enabled at the application server, no entry is required.*

- Application server password

Password for the application server user ID entered previously.

- Deployment manager user name

User ID and password for the deployment manager that is required since security is enabled at the deployment manager.

- Deployment manager password

The password for the deployment manager user ID entered previously.

- Config URL

Define the security settings that enables a remote application server to communicate with the deployment manager.

- Include applications

Check this box if you want the applications currently installed on the application server in the node to be included. If you do not check this box, any existing applications on the server will be uninstalled during the process.

- Include buses

If the node you are adding includes a service integration bus and you want to include it in the federation, check this box. The bus name has to be unique within the cell. If there is already a bus by the same name, the node will not be added.

- Starting port

If you want to specify the ports for the node rather than taking the default, you can specify a starting port. The numbers will be incremented from this number. For example, if you specify 3333, the BOOTSTRAP_ADDRESS port will be 3333, CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS will be 3334, and so on.

- Core group name

Specify the core group the node agent will belong to. If you only have one core group (DefaultCoreGroup), you will not see this option.

Click **OK**. The messages will be displayed on the administrative console. See Example 4-7.

Example 4-7 Adding a node from the administrative console - output messages

```
ADMU0505I: Servers found in configuration:
ADMU0506I: Server name: server1
ADMU2010I: Stopping all server processes for node AppSrv02Node
ADMU0510I: Server server1 is now STOPPED
ADMU0024I: Deleting the old backup directory.
ADMU0015I: Backing up the original cell repository.
ADMU0012I: Creating Node Agent configuration for node: AppSrv02Node
ADMU0014I: Adding node AppSrv02Node configuration to cell: Cell01
ADMU0016I: Synchronizing configuration between node and cell.
ADMU0018I: Launching Node Agent process for node: AppSrv02Node
ADMU0020I: Reading configuration for Node Agent process: nodeagent
ADMU0022I: Node Agent launched. Waiting for initialization status.
ADMU0030I: Node Agent initialization completed successfully. Process id is:
1196
ADMU9990I:
ADMU0300I: Congratulations! Your node AppSrv02Node has been successfully
incorporated into the Cell01 cell.
ADMU9990I:
ADMU0306I: Be aware:
ADMU0302I: Any cell-level documents from the standalone CARLAVM2Node03Cell
configuration have not been migrated to the new cell.
ADMU0307I: You might want to:
ADMU0303I: Update the configuration on the Cell01 Deployment Manager with
values from the old cell-level documents.
ADMU9990I:
ADMU0003I: Node AppSrv02Node has been successfully federated.
The new node will not be available in the console until you log in again
Logout from the WebSphere Administrative Console
```

Method 2: Using the addNode command

Before you begin, be certain these tasks are completed.

- ▶ Make sure the application server is started on the node to be added.
- ▶ Open the deployment manager administrative console and note the port specified as the SOAP_CONNECTOR_ADDRESS port for the deployment manager. You can find this port number by looking in the Additional Properties section in the Details page for the deployment manager.

To use the **addNode** command, do the following:

1. Open a command-line window on the system that has the running stand-alone application server.
2. Change the directory to the *<profile_home>/bin* directory of the stand-alone application server installation. On z/OS, the **addNode.sh** command is in the *<was_home>/bin* directory.
3. Execute **addNode**.

The **addNode** command adds a new node to an existing administrative cell.

The actions the command performs are:

1. Connects to the deployment manager process. This is necessary for the file transfers performed to and from the deployment manager in order to add the node to the cell.
2. Attempts to stop all running application servers on the node.
3. Backs up the current stand-alone node configuration to the *<profile_home>/config/backup/base/* directory.
4. Copies the stand-alone node configuration to a new cell structure that matches the deployment manager structure at the cell level.
5. Creates a new local config directory and definition (server.xml) for the node agent.
6. Creates entries (directories and files) in the master repository for the new node's managed servers, node agent, and application servers.
7. Uses the FileTransfer service to copy files from the new node to the master repository.
8. Uploads applications to the cell only if the **-includeapps** option is specified.
9. Performs the first file synchronization for the new node. This pulls everything down from the cell to the new node.
10. Fixes the node's **setupCmdLine** and **wsadmin** scripts to reflect the new cell environment settings.
11. Launches the node agent.

Important: Keep in mind the following points when adding a node to a cell.

- ▶ The cell must already exist.
- ▶ The cell's deployment manager must be running before **addNode** can be executed.
- ▶ The new node must have a unique name. If an existing node in the cell already has the same name, **addNode** will fail.
- ▶ By default, **addNode** does not carry over the applications or service integration buses when added to the cell. The **-includeApps** and **-includebuses** options must be used for this purpose.

Addnode command syntax

The syntax of the **addNode** command is as follows:

```
addNode.bat(sh) <dmgr_host> <dmgr_port> [options]
```

The command must be run from the node's *<profile_home>/bin*. It cannot be run from the deployment manager. The *<dmgr_host>* and *<dmgr_port>* parameters give the location of the deployment manager. The *<dmgr_host>* parameter is required.

The default JMX connector type to use is SOAP and the default port number for SOAP is 8879. If this is how you want to connect, and the SOAP_CONNECTOR_ADDRESS is 8879 for the deployment manager, you do not need to specify the *<dmgr_port>* parameter.

For options, see Table 4-11.

Table 4-11 Options for addNode

Option	Description
-nowait	Tell the command not to wait for successful completion of the node addition.
-quiet	Suppress progress information printed to the console in normal mode. This option does not affect information written to file.
-trace	Generate trace information into a file for debugging purposes. The output goes to addNode.log.
-logfile <log file path>	Specify an alternative location for command's log output, instead of addNode.log. The path can be specified in the following forms: absolute, relative, or file name. The default is <i><profile_home>/logs/addNode.log</i> .

Option	Description
-replacelog	Start a new log, replacing any previous log of the same name. If this argument is not specified, the default behavior is to append output to the existing file.
-conntype <type>	Specify the JMX connector to use for connection. Valid values are SOAP or RMI. If not specified, SOAP is assumed. If RMI is specified, then the deployment manager's correct RMI/IOP JMX connector port must be specified by the <dmgr_port> argument.
-profileName <profile>	Specify the profile to run the command against. If the command is run from <was_home>/bin and -profileName is not specified, the default profile is used. If it is run from <profile_home>/bin, that profile is used.
-username <username>	Specify a user name for authentication if WebSphere security is enabled. The user name is ignored if WebSphere security is disabled.
-password <password>	Specify a password for authentication if WebSphere security is enabled. The password is ignored if WebSphere security is disabled.
-includeapps	Attempt to include the applications in the incorporation of the base node into a cell. The default is not to include the applications.
-includebuses	If the node contains one or more service integration buses, carry these into the new configuration.
-startingport <port>	Used as the starting/base IP port number for the node agent created for this new node.
-portprops <qualified-filename>	Passes the name of the file that contains key-value pairs of explicit ports that you want the new node agent to use.
-nodeagentshortname <name>	Specify the short name to use for the new node agent.
-nodegroupname <name>	Specify the node group in which to add this node. If you do not specify, the node is added to the DefaultNodeGroup.
-registerservice -serviceusername <name> -servicepassword <password>	In Windows only, this option registers the node agent as a Windows service with the specified user ID and password.
-coregroupname <name>	Specify the core group in which to add this node. If you do not specify this option, the node will be added to the DefaultCoreGroup.
-statusport <port>	Set the port number for server status callback.
-noagent	Indicates that the new node agent (generated as part of adding the node to a cell) is not to be started at the end. The default setting is to start the node agent.
-help or -?	Print a usage statement.

Example 4-8 shows an example of using the **addNode** command to add a custom node to a cell.

Example 4-8 addNode usage examples

```
C:\<was_base>\profiles\Node02\bin>addnode carlavm2 8879 -startingport 3333

ADMU0116I: Tool information is being logged in file
           C:\WebSphere\AppServer\profiles\Node02\logs\addNode.log
ADMU0128I: Starting tool with the Node02 profile
ADMU0001I: Begin federation of node Node02 with Deployment Manager at
           carlavm2:8879.
ADMU0009I: Successfully connected to Deployment Manager Server: carlavm2:8879
ADMU0507I: No servers found in configuration under:
C:\<was_base>\profiles\Node02\config\cells\CARLAVM2Node02Cell\nodes\Node02\serv
ers
ADMU2010I: Stopping all server processes for node Node02
ADMU0024I: Deleting the old backup directory.
ADMU0015I: Backing up the original cell repository.
ADMU0012I: Creating Node Agent configuration for node: Node02
ADMU0014I: Adding node Node02 configuration to cell: Cell01
ADMU0016I: Synchronizing configuration between node and cell.
ADMU0018I: Launching Node Agent process for node: Node02
ADMU0020I: Reading configuration for Node Agent process: nodeagent
ADMU0022I: Node Agent launched. Waiting for initialization status.
ADMU0030I: Node Agent initialization completed successfully. Process id is:
           1072
ADMU9990I:
ADMU0300I: Congratulations! Your node Node02 has been successfully incorporated
           into the Cell01 cell.
ADMU9990I:
ADMU0306I: Be aware:
ADMU0302I: Any cell-level documents from the standalone CARLAVM2Node02Cell
           configuration have not been migrated to the new cell.
ADMU0307I: You might want to:
ADMU0303I: Update the configuration on the Cell01 Deployment Manager with
           values from the old cell-level documents.
ADMU9990I:
ADMU0306I: Be aware:
ADMU0304I: Because -includeapps was not specified, applications installed on
           the standalone node were not installed on the new cell.
ADMU0307I: You might want to:
ADMU0305I: Install applications onto the Cell01 cell using wsadmin $AdminApp or
           the Administrative Console.
ADMU9990I:
ADMU0003I: Node Node02 has been successfully federated.
C:\<was_base>\profiles\Node02\bin>
```

Federating a node on z/OS

The zPMT tool has an option that leads you through the process of generating jobs that will federate a stand-alone server to a cell.

4.5.2 Removing a node

There are two ways of removing a node from a network distributed administration cell.

Note: When a node is removed, it is restored to its original configuration, except when it was added to the cell.

Method 1: Using the administrative console

From the administrative console, do the following:

1. Select **System Administration** → **Nodes**.
2. Place a check mark in the check box beside the node you want to remove and click **Remove Node**.

This method runs the **removeNode** command in the background.

Method 2: Using the removeNode command

The **removeNode** command detaches a node from a cell and returns it to a stand-alone configuration.

To use the command, do the following:

1. Change the directory to the `<profile_home>/bin` directory.
2. Run **removeNode**. All parameters are optional for this command.

In a distributed environment on z/OS, the **removeNode.sh** command is in the `<was_home>/bin` directory. You will need to specify the `-profileName` parameter to specify the profile for the node you want to remove.

The command performs the following operations:

1. Connects to the deployment manager process to read the configuration data.
2. Stops all of the running server processes of the node, including the node agent process.
3. Removes servers in the node from clusters.
4. Restores the original stand-alone node configuration. This original configuration was backed up when the node was originally added to the cell.

5. Removes the node's configuration from the master repository of the cell. The local copy of the repository held on each node will get updated at the next synchronization point for each node agent. Although the complete set of configuration files are not pushed out to other nodes, some directories and files are pushed out to all nodes.
6. Removes installed applications from application servers in the cell that are part of the node being removed.
7. Copies the original application server cell configuration into the active configuration.

Unlike the **addNode** command, **removeNode** always uses the SOAP JMX connector of the deployment manager. There is no option provided for specifying the RMI JMX connector.

The command provides the **-force** option to force the local node's configuration to be decoupled from the cell even if the deployment manager cannot be contacted. However, if this situation occurs, the cell's master repository will then have to be separately updated to reflect the node's removal, for example, through manual editing of the master repository configuration files.

removeNode command

The command syntax is as follows:

```
removeNode [options]
```

Table 4-12 shows the **removeNode** parameters.

Table 4-12 removeNode parameters

Parameter	Description
-quiet	Suppress the printing of progress information.
-logfile <fileName>	Specify the location of the log file to which information is written. The default is <profile_home>/logs/removeNode.log.
-profileName <profile>	Specify the profile to run the command against. If the command is run from <was_home>/bin and -profileName is not specified, the default profile is used. If it is run from <profile_home>/bin, that profile is used.
-replacelog	Replace the log file instead of appending to the current log.
-trace	Generate trace information into the log file for debugging purposes.
-statusport <portNumber>	Set the port number for node agent status callback.
-username <name>	Specify the user name for authentication if security is enabled in the server.
-password <password>	Specify the password for authentication if security is enabled.

Parameter	Description
-force	Clean up the local node configuration, regardless of whether you can reach the deployment manager for cell repository cleanup. Note: After using the -force parameter, you might need to use the cleanupNode command on the deployment manager.
-help or -?	Print command syntax information.

Example

Table 4-9 shows an example of using the **removeNode** command.

Example 4-9 removeNode example

```

C:\<was_base>\bin>removeNode -profileName Custom01
ADMU0116I: Tool information is being logged in file
           C:\WebSphere\AppServer\profiles\Custom01\logs\removeNode.log
ADMU0128I: Starting tool with the Custom01 profile
ADMU2001I: Begin removal of node: CustomNode
ADMU0009I: Successfully connected to Deployment Manager Server:
           CARLAVM2.itso.ral.ibm.com:8879
ADMU0505I: Servers found in configuration:
ADMU0506I: Server name: Cserver1
ADMU0506I: Server name: Cserver2
ADMU0506I: Server name: nodeagent
ADMU2010I: Stopping all server processes for node CustomNode
ADMU0512I: Server Cserver1 cannot be reached. It appears to be stopped.
ADMU0512I: Server Cserver2 cannot be reached. It appears to be stopped.
ADMU0512I: Server nodeagent cannot be reached. It appears to be stopped.
ADMU2021I: Removing all servers on this node from all clusters in the cell.
ADMU2014I: Restoring original configuration.
ADMU2017I: The local original configuration has been restored.
ADMU9990I:
ADMU0306I: Be aware:
ADMU2031I: Any applications that were uploaded to the DMCell cell configuration
           during addNode using the -includeapps option are not uninstalled by
           removeNode.
ADMU0307I: You might want to:
ADMU2032I: Use wsadmin or the Administrative Console to uninstall any such
           applications from the Deployment Manager.
ADMU9990I:
ADMU2024I: Removal of node CustomNode is
complete.

```

4.5.3 Renaming a node

renameNode (new): The **renameNode** command allows you to modify the node name of a federated server.

To run the command, do the following:

1. Change to the *<profile_home>/bin* directory of the deployment manager.
2. Run the **renameNode** command.

The command:

1. Connects to the deployment manager.
2. Stops all servers.
3. Changes the node configuration on the deployment manager.
4. Synchronizes the node.

renameNode command

The command syntax is as follows:

```
renameNode.sh <dmgr_host> <dmgr_port> <node_name> [options]
```

The parameters for the command are shown in Table 4-13.

Table 4-13 *renameNode* parameters

Parameter	Description
-nodeshortname <name>	Short name of the node.
-conntype <type>	Specifies the JMX connector type to use for connecting to the deployment manager. Valid types are SOAP or RMI.
-trace	Generate trace information into the log file for debugging purposes.
-username <name>	Specify the user name for authentication if security is enabled in the server.
-password <password>	Specify the password for authentication if security is enabled.
-logfile <filename>	Specify the location of the log file to which information is written. The default is <i><profile_home>/logs/renameNode.log</i> .
-help or -?	Print command syntax information.

4.5.4 Node agent synchronization

Configuration synchronization between the node and the deployment manager is enabled by default. During a synchronization operation, a node agent checks with the deployment manager to see if any configuration documents that apply to the node have been updated. New or updated documents are copied to the node repository, and deleted documents are removed from the node repository. Configure the interval between synchronizations in the administrative console by doing the following:

1. Expand **System Administration** → **Node Agents** in the administrative console.
2. Select the node agent process on the appropriate server to open the Properties page.
3. In the Additional Properties section, click **File Synchronization Service**.
4. Configure the synchronization interval. By default, the synchronization interval is set to one minute.

Explicit synchronization can be forced by selecting **System Administration** → **Nodes**. Select a node and click **Synchronize** or **Full Synchronization**.

Synchronize performs an immediate synchronization on the selected node.

The Full Synchronization option disregards any synchronization optimization settings and ensures that the node and cell configuration are identical.

Tip: Increase the synchronization interval in a production environment to reduce the overhead.

Using the syncNode command

The **syncNode** command can be used to force the synchronization of a node's local configuration repository with the master repository on the deployment manager node.

Note: To use the **syncNode** command, the node agent must be stopped. You can use the **-stopservers** and **-restart** options on the **syncNode** command to stop the node agent and application servers, and then restart the node agent.

The syntax of the **syncNode** command is as follows:

```
syncNode.bat(sh) <dmgr_host> [dmgr_port] [options]
```

The first argument is mandatory. The options are listed in Table 4-14.

Table 4-14 Options for syncNode

Option	Description
-nowait	Tell the command not to wait for successful synchronization of the node.
-quiet	Suppress progress information printed to the console in normal mode. This option does not affect information written to file.
-trace	Generate trace information into a file for debugging purposes. The output goes to syncNode.log.
-profileName <profile>	Specify the profile to run the command against. If the command is run from <was_home>/bin and -profileName is not specified, the default profile is used. If it is run from <profile_home>/bin, that profile is used.
-conntype <type>	Specify the JMX connector type to use for connection to the deployment manager. Valid values are SOAP or RMI. If not specified, SOAP is assumed.
-stopservers	Indicate that the node agent and all managed servers of the node should be stopped prior to synchronizing the node's configuration with the cell.
-restart	Indicate that the node agent is to be restarted after synchronizing the node's configuration with the cell.
-logfile <log file path>	Specify an alternative location for the command's log output, instead of syncNode.log. The path can be specified in the following forms: absolute, relative, or file name. The default location is <profile_home>/logs/syncNode.log
-replacelog	Start a new log, replacing any previous log of the same name. If this argument is not specified, the default behavior is to append the output to the existing file.
-username <username>	Specify a user name for authentication if WebSphere security is enabled. Ignore it if WebSphere security is disabled.
-password <password>	Specify a password for authentication if WebSphere security is enabled. Ignore it if WebSphere security is disabled.
-localusername <localusername>	Specifies the user name for authentication for existing application servers on the node that you want to federate. This parameter is only applicable if security is enabled for the application server.

Option	Description
-localpassword <localpassword>	Specifies the password for authentication for existing application servers on the node that you want to federate. The password that you choose must be one that is associated with a preexisting user name. This parameter is only applicable if security is enabled for the application server.
-help or -?	Print a usage statement.

Example 4-10 shows an example of using the **syncNode** command. This example was run on a Windows system.

Example 4-10 syncNode usage examples

```
C:\<was_base>\profiles\Node01\bin>stopnode

ADMU0116I: Tool information is being logged in file
           C:\WebSphere\AppServer\profiles\Node01\logs\nodeagent\stopServer.log
ADMU0128I: Starting tool with the Node01 profile
ADMU3100I: Reading configuration for server: nodeagent
ADMU3201I: Server stop request issued. Waiting for stop status.
ADMU4000I: Server nodeagent stop completed.

C:\<was_base>\profiles\Node01\bin>syncnode carlavm2

ADMU0116I: Tool information is being logged in file
           C:\WebSphere\AppServer\profiles\Node01\logs\syncNode.log
ADMU0128I: Starting tool with the Node01 profile
ADMU0401I: Begin syncNode operation for node Node01 with Deployment Manager
           carlavm2: 8879
ADMU0016I: Synchronizing configuration between node and cell.
ADMU0402I: The configuration for node Node01 has been synchronized with
           Deployment Manager carlavm2: 8879
```

4.5.5 Starting and stopping nodes

A node consists of the node agent and the servers. There are several ways to start and stop a node and node agent, or stop them individually. Before using any of these methods, be sure to note whether it affects the entire node, including servers, or just the node agent.

Starting a node agent

When a node agent is stopped, the deployment manager has no way to communicate with it. Therefore, the node agent has to be started with the **startNode** command run from on the profile node system.

From a command prompt, type the following command:

- Windows: **<profile_home>\bin\startNode**
- UNIX and z/OS: **<profile_home>/bin/startNode.sh**

startNode command

The command syntax is as follows:

startNode.bat(sh) [options]

The parameters are shown in Table 4-15.

Table 4-15 startNode parameters

Parameter	Description
-nowait	Do not wait for successful initialization of the node agent process.
-quiet	Suppress the printing of progress information.
-logfile <fileName>	Specify the location of the log file to which information gets written. The default is <profile_home>/logs/nodeagent/startServer.log.
-profileName <profile>	Specify the profile to run the command against. If the command is run from <was_home>/bin and -profileName is not specified, the default profile is used. If it is run from <profile_home>/bin, that profile is used.
-replacelog	Replace the log file instead of appending to the current log.
-trace	Generate trace information into the log file for debugging purposes.
-timeout <seconds>	Specify the wait time before node agent initialization times out and returns an error.
-statusport <portNumber>	Set the port number for node agent status callback.
-script [<script fileName>] -background	Generate a launch script with the startNode command instead of launching the node agent process directly. The launch script name is an optional argument. If you do not provide the launch script name, the default script file name is start_<nodeName>, based on the name of the node. The -background parameter is an optional parameter that specifies that the generated script will run in the background when you execute it.
-J-<java_option>	Specify options to pass through to the Java interpreter.
-help	Prints command syntax information

See Example 4-11 for an example of the **startNode** command.

Example 4-11 startNode command

```
C:\<was_base>\profiles\<profile_name>\bin>startnode
ADMU0116I: Tool information is being logged in file
C:\WebSphere\AppServer\profiles\Custom01\logs\nodeagent\startServer.log
ADMU0128I: Starting tool with the Custom01 profile
ADMU3100I: Reading configuration for server: nodeagent
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server nodeagent open for e-business; process id is 1816
```

Starting a node on z/OS using the START command

To start a node agent on z/OS using the START command, use the following format:

```
START nodeagent_procname,JOBNAME=server_shortname,
ENV=cell_shortname.node_shortname.server_shortname
```

For example:

```
START CHACRA,JOBNAME=CHAGNTA,ENV=CHCELL.CHNODEA.CHAGNTA
```

Stopping a node agent

To stop the node agent and leave the servers running, do the following, depending on your preferred method.

From the administrative console, do the following:

1. From the administrative console, select **System Administration** → **Node Agents**.
2. Check the box beside the node agent for the server and click **Stop**.

From a command prompt, type the following command:

- ▶ Windows: `<profile_home>\bin\stopNode`
- ▶ UNIX and z/OS: `<profile_home>/bin/stopNode.sh`

Note: Once you stop the node agent, the deployment manager has no way to communicate with the servers on that node. The servers might be up and running, but the administrative console is not able to determine their status.

stopNode command

The command syntax is as follows:

```
stopNode [options]
```

The parameters are shown in Table 4-16 on page 218.

Table 4-16 *stopNode* parameters

Parameter	Description
-nowait	Do not wait for successful initialization of the node agent process.
-quiet	Suppress the printing of progress information.
-logfile <fileName>	Specify the location of the log file to which information gets written. The default is <profile_home>/logs/nodeagent/stopServer.log.
-profileName <profile>	Specify the profile to run the command against. If the command is run from <was_home>/bin and -profileName is not specified, the default profile is used. If run from <profile_home>/bin, that profile is used.
-replacelog	Replace the log file instead of appending to the current log.
-trace	Generate trace information into the log file for debugging purposes.
-timeout <seconds>	The wait time before node agent shutdown times out and returns an error.
-statusport <portNumber>	Set the port number for node agent status callback.
-username <name>	Specify the user name for authentication if security is enabled in the server.
-password <password>	Specify the password for authentication if security is enabled.
-stopservers	Stop all application servers on the node before stopping the node agent.
-conntype <type>	Specify the JMX connector type to use for connecting to the deployment manager. Valid types are SOAP or RMI.
-port <portNumber>	Specify the node agent JMX port to use explicitly, so that you can avoid reading configuration files to obtain the information.
-help	Print command syntax information.

See Example 4-12 for an example and sample output of the **stopNode** command.

Example 4-12 stopNode command

```
C:\<was_base>\profiles\<profile_name>\bin>stopNode
ADMU0116I: Tool information is being logged in file
C:\<was_base>\profiles\<profile_name>\logs\nodeagent\stopServer.log
ADMU0128I: Starting tool with the Custom01 profile
ADMU3100I: Reading configuration for server: nodeagent
ADMU3201I: Server stop request issued. Waiting for stop status.
ADMU4000I: Server nodeagent stop
completed.
```

Stopping a node on z/OS using the STOP command

To stop a node agent on z/OS, you can use the following command:

```
STOP nodeagent_JOBNAME
```

For example:

```
STOP CHAGNTA
```

Stopping a node (the node agent and servers)

You can use the administrative console to stop a node and its servers with one action:

1. From the administrative console, select **System Administration** → **Nodes**.
2. Check the box beside the node and click **Stop**.

Restarting a node agent

You can restart a running node agent from the administrative console by doing the following from the administrative console:

1. Select **System Administration** → **Node Agents**.
2. Check the box beside the node agent for the server and click **Restart**.

4.5.6 Node groups

You can have nodes in cells with different capabilities. Currently, this means having a cell with nodes on both distributed platforms and z/OS nodes. In the future, there might be other situations that fit this criteria. However, there are still restrictions on how the nodes can coexist. For example, you cannot have mixed nodes in a cluster. Node groups are created to group nodes of similar capability together to allow validation during system administration processes.

A default node group called `DefaultNodeGroup` is automatically created for you when the deployment manager is created, based on the deployment manager platform. New nodes on similar platforms are automatically added to the default group. A node must belong to at least one node group, but can belong to more than one.

As long as you have nodes in a cell with similar platforms, you do not need to do anything with node groups. New nodes are automatically added to the node group. However, before adding a node on a platform that does not have the same capabilities as the deployment manager platform, you will need to create the new node group.

Working with node groups

You can display the default node group and its members by selecting **System Administration** → **Node Groups**. See Figure 4-28.

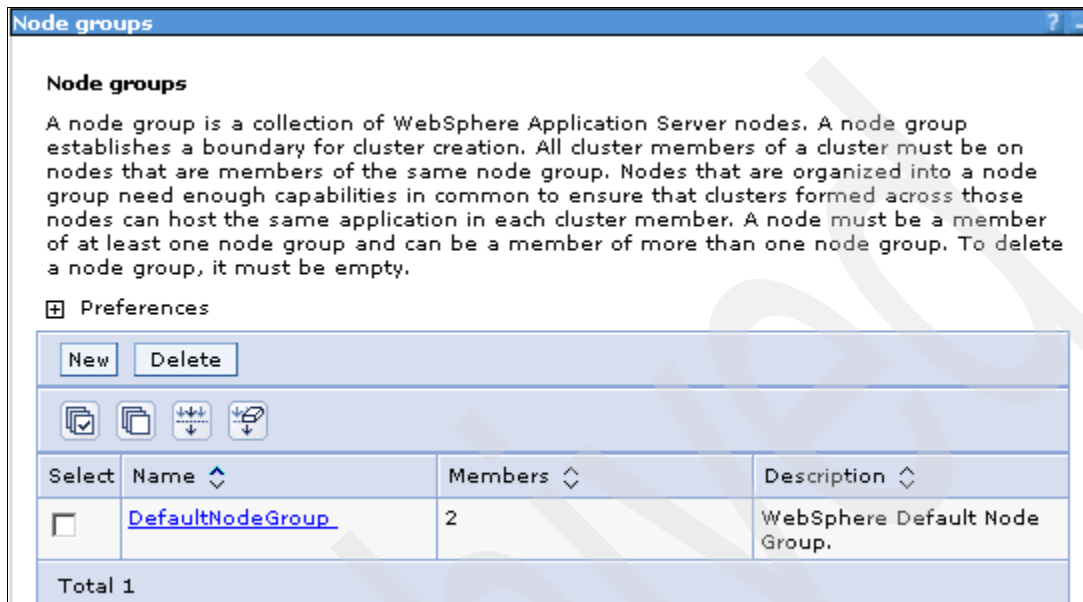


Figure 4-28 Display a list of node groups

- ▶ To create a new node group, click **New**. The only thing you need to enter is the name of the new node group. Click **OK**.
- ▶ To delete a node group, check the box to the left of the node group name and select **Delete**.
- ▶ To display a node group, click the node group name. For example, in Figure 4-30, we have displayed the DefaultNodeGroup.

General Properties	Additional Properties
Name DefaultNodeGroup	■ Custom properties ■ Node group members
Members 2	
Description WebSphere Default Node Group.	
Apply OK Reset Cancel	

Figure 4-29 Node group properties

- To add a node to a node group, display the node group and click **Node group members** in the Additional Properties section. When the list appears, select **Add**. You will be able to select from a list of nodes.

Node groups

[Node groups](#) > [DefaultNodeGroup](#) > **Node group members**

View and configure the nodes in this node group.

Preferences

Select	Name
☐	DMGRNode
☐	Node01

Total 2

Figure 4-30 Displaying node group members

4.6 Working with clusters

This section discusses creating, configuring, and managing clusters using the administrative console. Clustering is an option in a distributed server environment only.

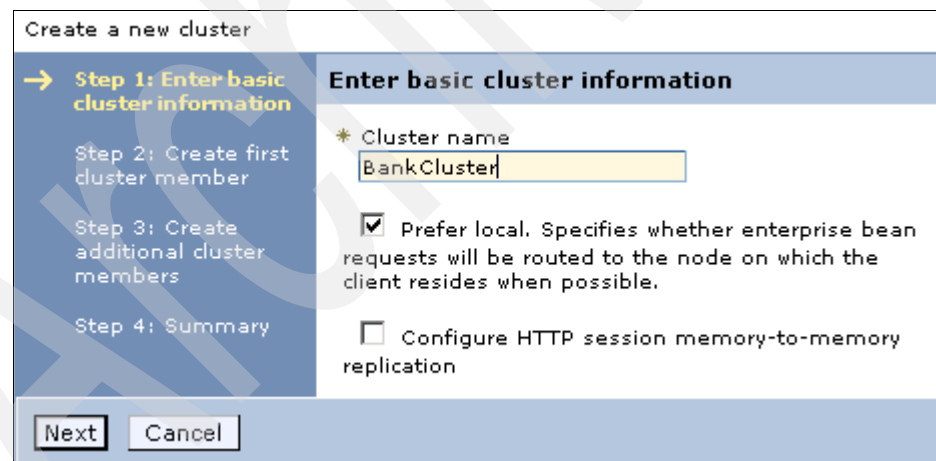
4.6.1 Creating clusters

Clusters consist of one or more application servers. When you create a cluster, you can choose one existing application server to add to the cluster. The rest of the servers must be new and can be created when you create the cluster or later.

When creating a cluster, it is possible to select the template of an existing application server for the cluster without adding that application server into the new cluster. For this reason, consider creating an application server with the server properties that you want as a standard in the cluster first, then use that server as a template or as the first server in the cluster.

To create a new cluster:

1. Select **Servers** → **Clusters**.
2. Click **New**. See Figure 4-31 on page 222.



Create a new cluster

→ Step 1: Enter basic cluster information

Step 2: Create first cluster member

Step 3: Create additional cluster members

Step 4: Summary

Enter basic cluster information

* Cluster name
BankCluster

☒ Prefer local. Specifies whether enterprise bean requests will be routed to the node on which the client resides when possible.

☐ Configure HTTP session memory-to-memory replication

Next Cancel

Figure 4-31 Creating a new cluster

3. Enter the information for the new cluster:
 - Cluster name: Enter a cluster name of your choice. On z/OS, you will also be asked for the short name for the cluster.

- Prefer local: This setting indicates that a request to an EJB should be routed to an EJB on the local node if available.
 - Configure HTTP session memory-to-memory replication: WebSphere Application Server supports session replication to another WebSphere Application Server instance. In this mode, sessions can replicate to one or more WebSphere Application Server instances to address HTTP Session single point of failure.
4. Create first cluster member: The first cluster member determines the server settings for the cluster members.

Step 1: Enter basic cluster information

→ Step 2: Create first cluster member

Step 3: Create additional cluster members

Step 4: Summary

Create first cluster member

The first cluster member determines the server settings for the cluster members. A server configuration template is created from the first member and stored as part of the cluster data. Additional cluster members are copied from this template.

* Member name
BankServer1

Select node
kadw028Node03(ND 6.1.0.0)

* Weight
2 (0..20)

☒ Generate unique HTTP ports

Core Group
DefaultCoreGroup

Select basis for first cluster member:

☒ Create the member using an application server template.
default

☐ Create the member using an existing application server as a template.
kadw028Cell01/kadw028Node03(ND 6.1.0.0)/server1

☐ Create the member by converting an existing application server.
kadw028Cell01/kadw028Node03(ND 6.1.0.0)/server1

☐ None. Create an empty cluster.

Previous Next Cancel

Figure 4-32 First cluster member

- **Member Name:** Type a name of the new server to be added to the cluster. On z/OS, you will also be asked for the short name for the server.
- **Select Node:** Specifies the node on which this new cluster member is created.
- **Server weight:** The value for this field determines how workload is distributed. For example, if all cluster members have identical weights, work is distributed among the cluster members equally. Servers with higher weight values are given more work. A rule of thumb formula for determining routing preference would be:

$$\% \text{ routed to Server1} = \text{weight1} / (\text{weight1} + \text{weight2} + \dots + \text{weight n})$$

In the formula, n represents the number of cluster members in the cluster.
- **Generate unique HTTP ports:** Generates unique port numbers for every transport that is defined in the source server, so that the resulting server that is created will not have transports that conflict with the original server or any other servers defined on the same node.
- **Select basis for first cluster member:**
 - If you select **Create the member using an application server template**, the settings for the new application server are identical to the settings of the application server template you select from the list of available templates.
 - If you select **Create the member using an existing application server as a template**, the settings for the new application server are identical to the settings of the application server you select from the list of existing application servers.
 - If you select **Create the member by converting an existing application server**, the application server you select from the list of available application servers becomes a member of this cluster.
 - If you select **None. Create an empty cluster**, a new cluster is created, but it does not contain any cluster members.

Click **Next**.

5. **Create additional cluster members:** Use this page to create additional members for a cluster. You can add a member to a cluster when you create the cluster or after you create the cluster. A copy of the first cluster member that you create is stored as part of the cluster data and becomes the template for all additional cluster members that you create.

To add a member, enter a new server name, select the node, and click **Add Member**. See Figure 4-33.

Step 1: Enter basic cluster information

Step 2: Create first cluster member

→ Step 3: Create additional cluster members

Step 4: Summary

Create additional cluster members

Enter information about this new cluster member, and click Add Member to add this cluster member to the member list. A server configuration template is created from the first member and stored as part of the cluster data. Additional cluster members are copied from this template.

* Member name

Select node

* Weight
 (0..20)

☒ Generate unique HTTP ports

Use the Edit function to edit the properties of a cluster member that is already included in this list. Use the Delete function to remove a cluster member from this list. You are not allowed to edit or remove the first cluster member or an already existing cluster member.

☒
☐

Select	Member name	Nodes	Version	Weight
☐	BankServer1	kadw028Node03	ND 6.1.0.0	2

Figure 4-33 Additional cluster members

- When all the servers have been entered, click **Next**.
- A summary page shows you what will be created.
- Click **Finish** to create the cluster and new servers.
- Save the configuration.

4.6.2 Viewing cluster topology

The administrative console provides a graphical view of the existing clusters and their members. To see the view, do the following:

1. Select **Servers** → **Cluster Topology**.
2. Expand each category. See Figure 4-34.

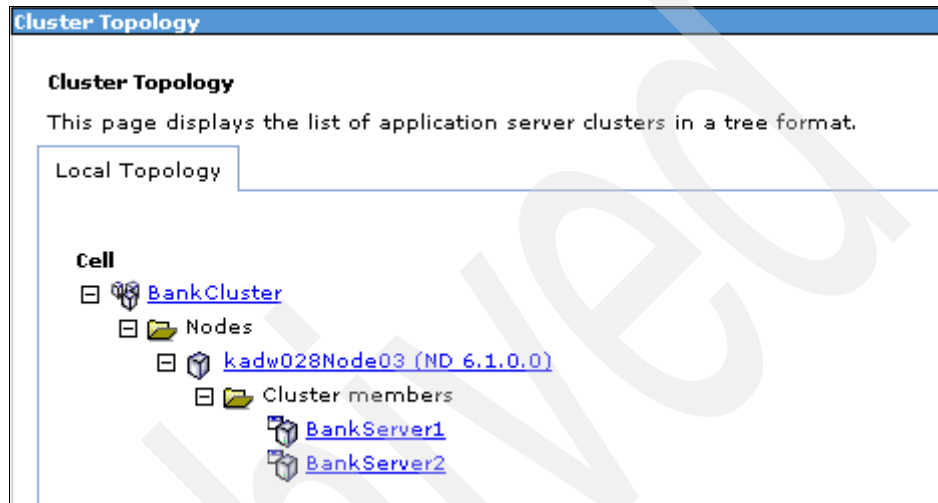


Figure 4-34 Cluster topology view

3. Selecting a server will take you to the configuration window for the application server.

4.6.3 Managing clusters

Application servers within a cluster can be managed as independent servers. A second option is to manage all the servers in the cluster using a single button:

1. Select **Servers** → **Clusters**.
2. Check each cluster you want to work with and select one of the following options:
 - Start: Use this option to start all servers in the cluster.
 - Stop: Use this option to stop all servers in the cluster. This allows the server to finish existing requests and allows failover to another member of the cluster.
 - Ripplestart: Use this option to Stop, then start all servers in the cluster.
 - ImmediateStop: Stop all servers immediately.

4.7 Working with virtual hosts

Note: For an example of defining and using a new virtual host, see 14.1.4, “Defining the Plants by WebSphere virtual host” on page 901.

A *virtual host* is a configuration enabling a single host machine to resemble multiple host machines. It consists of a host alias or aliases, which consist of a host name and a port number. If you specify an asterisk (*) as a host name, all host names and IP addresses that the Web server can receive will be mapped to that virtual host.

There are two virtual hosts defined during installation: `default_host` and `admin_host`.

- ▶ The *default_host* virtual host is intended for access to user applications, either through the HTTP transport or through a Web server. At installation time, it is configured as the default virtual host for the `server1` application server. It is configured to match requests to ports 80, 9080, and 9443 for any host name.
- ▶ The *admin_host* virtual host is used for access to the WebSphere administrative console. It is configured to match requests to the secure ports 9090 (HTTP transport) and 9043 (Web server) for any host name.
- ▶ The *proxy_host* virtual host includes default port definitions, port 80 and 443, which are typically initialized as part of the proxy server initialization. Use this proxy host as appropriate with routing rules associated with the proxy server.

When you install an application, you associate a virtual host with each Web module in the application. By associating a virtual host with a Web module, requests that match the host aliases for the virtual host should be processed by servlets/JSPs in this Web module. The Web server plug-in also checks the URI of the request against the URIs for the Web module to determine whether the Web module can handle them or not.

A single virtual host can be associated with multiple Web modules unless each application has unique URIs. If there are same URIs among applications, different virtual hosts must be created and associated with each of the applications.

4.7.1 Creating a virtual host

By default, default_host is associated with all user application requests. There are some cases in which multiple virtual hosts should be created, for example:

- ▶ Applications having conflicting URIs
- ▶ Support for extra ports, such as port 443 for SSL
- ▶ Keep clear independence of each virtual host for applications and servers

The configuration of a virtual host is applied to an entire cell. To create a new virtual host, do the following:

1. Select **Environment** → **Virtual Hosts** and then click **New**.
2. Enter a name for the virtual host and click **Apply**.
3. Click **Host Aliases** in the Additional Properties pane.
4. Click **New**.
5. Enter values for the Host Name and Port fields and click **OK**.

The host aliases are not necessarily the same as the host name and port number of the WebSphere Application Servers. They are the host names and port numbers that the Web server plug-in is expecting to receive from the browser. The Web server plug-in will send the request to the application server using the host name and port number in the transport setting for that server. If the Web server is running on a separate machine from WebSphere, then the host aliases are for Web server machines.

Mapping HTTP requests to host aliases is case sensitive and the match must be alphabetically exact. Also, different port numbers are treated as different aliases.

For example, the request `http://www.myhost.com/myservlet` does *not* map to any of the following:

```
http://myhost/myservlet
http://www.myhost.com/MyServlet
http://www.myhost.com:9876/myservlet
```

If the Web server plug-in receives a request that does not match one of the virtual hosts, then an HTTP error will be returned to the user.

Simple wild cards can be used on the host aliases. A * can be used for the host name, the port or both. It means that any request will match this rule.

Note: If the virtual host is used in a cluster environment, all host aliases used by servers in the cluster should be registered in the virtual host. For information about how to do this, see 7.3.1, “Regenerating the plug-in configuration file” on page 386.

6. Multi-Purpose Internet Mail Extensions (MIME) mappings associate a file name extension with a type of data file, such as text, audio, or image. A set of MIME types is automatically defined for you when you create a virtual host. To see or alter the MIME types associated with this new virtual host, click **MIME Types** in the Additional Properties section of the virtual host.
7. Click **New** to add a MIME type.
8. Enter the MIME type and extension. Click **Apply** to continue adding new types or click **OK** if you are finished.
9. Click **Save** on the taskbar and save your changes.

Important: If you create, delete, or update virtual hosts, you need to regenerate the Web server plug-in.

4.8 Managing applications

Applications can be managed using the following methods:

- Using **wsadmin**

Using scripts to manage applications is more complicated than using the other methods. It requires skill in at least one of the supported scripting languages and a complete understanding of the WebSphere Application Server configuration. However, scripting can offer a greater degree of control and can be quite useful in situations where you are performing the same administrative tasks multiple times, or when the tasks are to be done by multiple administrators.

Information about using **wsadmin** scripts is found in Chapter 5, “Administration with scripting” on page 249.

- Using WebSphere Rapid Deployment

The rapid deployment tools in WebSphere Rapid Deployment provides a shortcut to installing, uninstalling, and updating applications. You can place full J2EE applications (EAR files), application modules (WAR files or EJB JAR files), or application artifacts (Java source files, Java class files, images, JSPs, and so on) into a configurable location on your file system, referred to as the *monitored*, or *project*, directory. The rapid deployment tools then automatically detect added or changed parts of these J2EE artifacts and performs the steps necessary to produce a running application on an application server.

For information about using this feature, see *Rapid deployment of J2EE applications* topic in the Information Center.

- Using the administrative console

Using the administrative console is an easy way to install or update an application. Wizards take you through the process and provide help information at each step.

This is the method discussed in this section at a high level. A detailed example of it can be found in Chapter 14, “Deploying applications” on page 893.

4.8.1 Using the administrative console to manage applications

To view and manage applications using the administrative console, select **Applications** → **Enterprise Applications**.

In the window, you see the list of installed applications and options for performing application management tasks. Select one or more applications by checking the box to the left of the application name, and then click an action to perform. The exception to this is the Install option, which installs a new application, and requires no existing application to be selected. See Figure 4-35 on page 230.

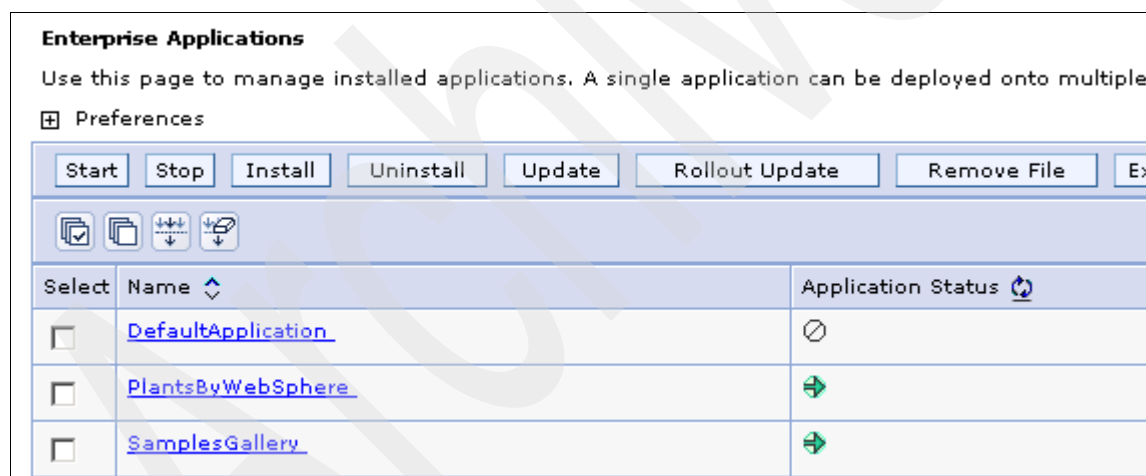


Figure 4-35 Working with enterprise applications

The following list describes the actions you can choose on this window.

- Start

Applications normally start when the server to which they are mapped starts. Exceptions to this include when the application has just been installed, and when the application has been stopped manually.

- ▶ **Stop**
You can stop an application manually without affecting the rest of the application server processes. This is common when you are updating an application or want to make it unavailable to users.
- ▶ **Install**
The install option takes you through the process of installing a new enterprise application EAR file.
- ▶ **Uninstall**
Use this to uninstall an application. This removes it from the application servers and from the configuration repository.
- ▶ **Update or Rollout Update**
Applications can be updated in several ways. The update options include full application, single module, single file, and partial application.
- ▶ **Remove file**
With this option, you can remove a single file from an application.
- ▶ **Export**
Use this option to export an EAR file of the application.
- ▶ **Export DDL**
Use this option to export DDL files found in the application.

4.8.2 Installing an enterprise application

Adding a new cluster member: When an application server is added as a member to a server cluster, the modules installed on other members are also installed on the new member. You do not need to re-install or upgrade the application.

To install an enterprise application into a WebSphere configuration, you must install its modules onto one or more application servers. Follow these steps for this task:

1. Select **Applications** → **Enterprise Applications** → **Install**, or **Applications** → **Install New Application**.

2. Specify the location of the EAR file to install, as shown in Figure 4-36 on page 232.

The EAR file that you are installing can be either on the client machine running the Web browser, or on any of the nodes in the cell.

Preparing for the application installation

Specify the EAR, WAR, JAR, or SAR module to upload and install.

Path to the new application

☒ Local file system

Full path
C:\MyApps\BankApp.ear

☐ Remote file system

Full path

Context root
 Used only for standalone Web modules (.war files) and SIP modules (.sar files)

How do you want to install the application?

☒ Prompt me only when additional information is required.

☐ Show me all installation options and parameters.

Figure 4-36 Installing an enterprise application

Streamline™ installation process (new): Note the new V6.1 Prompt me... option that allows you to streamline the installation process.

Click **Next**.

3. The first window has settings used during the installation. These settings primarily determine whether default settings will be used or if you will override them during the installation. You can choose to view all installation options and parameters or just prompt when additional information is required.

Click **Next**.

4. The rest of the installation process is done in steps. The steps can vary, depending on the contents of the EAR file. The following steps are a typical sequence for the option Show me all installation options and parameters:
 - a. Provide options to perform the installation. This includes an option to use embedded configuration values in an Enhanced EAR and the option to pre-compile JSPs.
 - b. Map modules to servers
 - c. Provide JSP reloading options for Web modules
 - d. Map shared libraries
 - e. Initialize parameters for servlets
 - f. Provide JNDI names for beans
 - g. Map resource references to resources
 - h. Map virtual hosts for Web modules
 - i. Map context roots for Web modules
 - j. Map security roles to users or groups
 - k. Summary
5. Click **Finish** to install the application.
6. Save the configuration.

For information about where the application files are stored, see 2.4.3, “Application data files” on page 42.

4.8.3 Uninstalling an enterprise application

To uninstall a no longer needed enterprise application, do the following:

1. Select **Applications** → **Enterprise Applications**.
2. Check the application you want to uninstall and click **Uninstall**.

4.8.4 Exporting an enterprise application

If you have modified the binding information of an enterprise application, you might want to export the changed bindings to a new EAR file. To export an enterprise application to an EAR file:

1. Select **Applications** → **Enterprise Applications**.
2. Check the application you want to export and click **Export**.
3. Click the link for the file you want to export.
4. Click **Save**.
5. Specify the directory on the local machine and click **Save**.

4.8.5 Starting an enterprise application

An application starts automatically when the application server to which it is mapped starts. You only need to start an application immediately after installing it, or if you have manually stopped it.

Application startup: Starting an application server starts the applications mapped to that server. The order in which the applications start depends on the weights you assigned to each them. The application with the lowest starting weight is started first. Applications that have the same weight are started in no particular order. Enabling the parallel start option for the application server means start applications with the same weight in parallel.

To view or change the application starting weight, select **Applications** → **Enterprise Applications**. To find the Starting weight field, open the configuration page for the application by clicking the application name.

An application can be started by following these steps from the administrative console:

1. Select **Applications** → **Enterprise Applications**.
2. Check the application you want and click **Start**.

Note: In order to start an application, the application server that contains the application has to be started. If not, the application displays in the administrative console as unavailable and you are not able to start it.

4.8.6 Stopping an enterprise application

An application can be stopped using the administrative console.

1. From the administrative console, do the following.
 - a. Select **Applications** → **Enterprise Applications**
 - b. Check the application you want to stop and click **Stop**.

4.8.7 Preventing an enterprise application from starting on a server

By default, an application will start when the server starts. The only way to prevent this is to disable the application from running on the server.

1. From the administrative console:
 - a. Select **Applications** → **Enterprise Applications**.
 - b. Click the application to open the configuration.

- c. Select **Target specific application status** in the Detail Properties table.
- d. Select the server for which you want to disable the application.
- e. Click the **Disable Auto Start** button.
- f. Save the configuration.

4.8.8 Viewing application details

The administrative console does not display the deployed servlets, JSPs, or EJBs directly on the console. However, you can use the console to display XML deployment descriptors for the enterprise application, Web modules, and EJB modules.

To view the application deployment descriptor for an application, do the following:

1. From the console navigation tree, select **Applications** → **Enterprise Applications**.
2. Click the application that you are interested in.
3. Under the Configuration tab, select **View Deployment Descriptor** under Detail Properties.

Figure 4-37 shows the deployment descriptor window for the PlantsByWebSphere enterprise application. The Configuration tab shows you the structure defined by the deployment descriptor:

- ▶ The name and description of the enterprise application
- ▶ The Web modules or WAR files and their context roots
- ▶ The EJB modules and their associated JAR files
- ▶ The security roles associated with the enterprise application



Figure 4-37 Enterprise application deployment descriptor

Viewing EJB modules

To see the EJBs that are part of an enterprise application:

1. Select **Applications** → **Enterprise Applications**.
2. Click the application that you are interested in.

3. Select **Manage Modules** under Modules Items.
4. Click the EJB module you want to view. See Figure 4-38.

The screenshot shows the 'Enterprise Applications' console. The breadcrumb path is 'Enterprise Applications > PlantsByWebSphere > Manage Modules > PlantsByWebSphereEJB.jar'. Below the path, it says 'Specifies a server-module installation binding for an EJB module.' There is a 'Configuration' tab selected. The configuration is divided into two sections: 'General Properties' and 'Additional Properties'. Under 'General Properties', there is a field for 'URI' with the value 'PlantsByWebSphereEJB.jar', an empty field for 'Alternate deployment descriptor', and a 'Starting weight' field with a value of '5000'. At the bottom of the 'General Properties' section are buttons for 'Apply', 'OK', 'Reset', and 'Cancel'. The 'Additional Properties' section contains four links: 'Stateful session bean failover settings', 'View Module Class Loader', 'Target specific application status', and 'View Deployment Descriptor'.

Figure 4-38 Viewing an EJB module configuration

5. Click **View Deployment Descriptor** under Additional Properties to see the EJB deployment descriptor.

Viewing Web modules

To see the servlets and JSPs that are part of an enterprise application:

1. Select **Applications** → **Enterprise Applications**.
2. Click the application that you are interested in.
3. Select **Manage Modules** under Modules.

4. Click the Web module you want to view. See Figure 4-39.

[Enterprise Applications](#) > [PlantsByWebSphere](#) > [Manage Modules](#) > **PlantsByWebSphere.war**

Use this page to configure an instance of a deployed Web module in the application. This page contains deployment-specific information for a Web module and session management settings.

Configuration

General Properties

* URI

Alternate deployment descriptor

* Starting weight

* Class loader order

Web Services Properties

- [Web services client bindings](#)
- [View Web services client deployment descriptor extension](#)

Web Services Security Properties

- [Web services: Client security bindings](#)

Additional Properties

- [View Module Class Loader](#)
- [Target specific application status](#)
- [View Deployment Descriptor](#)
- [View Portlet Deployment Descriptor](#)
- [Session Management](#)
- [Web Module Proxy Configuration](#)

Figure 4-39 View a Web module

5. Click **View Deployment Descriptor** to see the details of the Web module content.

4.8.9 Finding a URL for a servlet or JSP

The URL for a servlet or JSP is the path used to access it from a browser. The URL is partly defined in the deployment descriptor provided in the EAR file and partly defined in the deployment descriptor for the Web module containing the servlet or JSP.

To find the URL for a servlet or JSP:

1. Find the context root of the Web module containing the servlet.
2. Find the URL for the servlet.
3. Find the virtual host where the Web module is installed.

4. Find the aliases by which the virtual host is known.
5. Combine the virtual host alias, context root, and URL pattern to form the URL request of the servlet/JSP.

For example, to look up the URL for the snoop servlet:

1. Find the context root of the Web module DefaultWebApplication of the DefaultApplication enterprise application. This Web module contains the snoop servlet.
 - a. From the console navigation tree, select **Applications** → **Enterprise Applications**.
 - b. Click the application that you are interested in, in our case, **DefaultApplication**.
 - c. On the Configuration tab, select **Context Root for Web Modules**. (Figure 4-40). You can see:
 - i. There is only one Web module in this application, DefaultWebApplication.
 - ii. The context root for the DefaultWebApplication Web module is `/`. We will use this later.

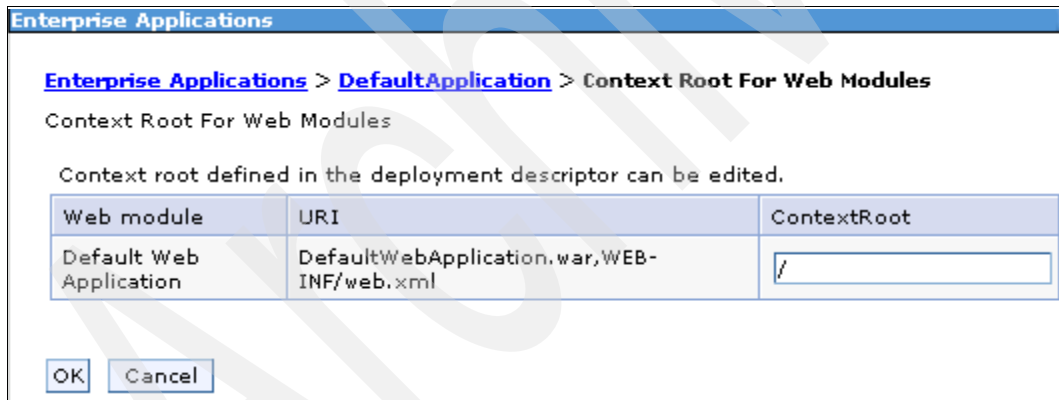


Figure 4-40 Context root for the Web modules in DefaultApplication

- d. Click **OK** to return to the DefaultApplication configuration.
2. Find the URL for the snoop servlet:
 - a. In the DefaultApplication configuration page, select **Manage Modules**.
 - b. Click the **DefaultWebApplication** Web module to see the general properties.

c. Click **View Deployment Descriptor**.

This displays the Web module properties window, as shown in Figure 4-41. Note that the URL pattern for the snoop servlet starting from the Web module context root is `/snoop/*`. The Web module context root was `/`.



Figure 4-41 DefaultWebApplication Web module deployment descriptor

d. Note that as you navigate through the windows, a navigation path is displayed below the Messages area. Click **DefaultApplication** to return to the application configuration page.

3. Find the virtual host where the DefaultWebApplication Web module is installed:
 - a. In the DefaultApplication configuration page, select **Virtual hosts** under Web Module Properties.

This will display all of the Web modules contained in the enterprise application, and the virtual hosts in which they have been installed. See Figure 4-42. Note that the DefaultWebApplication Web module has been installed on the default_host virtual host.

Enterprise Applications > DefaultApplication > Virtual hosts

Virtual hosts

Specify the virtual host where you want to install the Web modules that are contained in your application. You can install Web modules on the same virtual host or disperse them among several hosts.

☐ Apply Multiple Mappings

Select	Web module	Virtual host
☐	Default Web Application	default_host

Figure 4-42 List of virtual hosts

4. Find the host aliases for the default_host virtual host.
 - a. From the console navigation tree, select **Environment** → **Virtual Hosts**.
 - b. Click **default_host**.

- c. Select **Host Aliases** under Additional Properties.

This shows the list of aliases by which the default_host virtual host is known. See Figure 4-43.

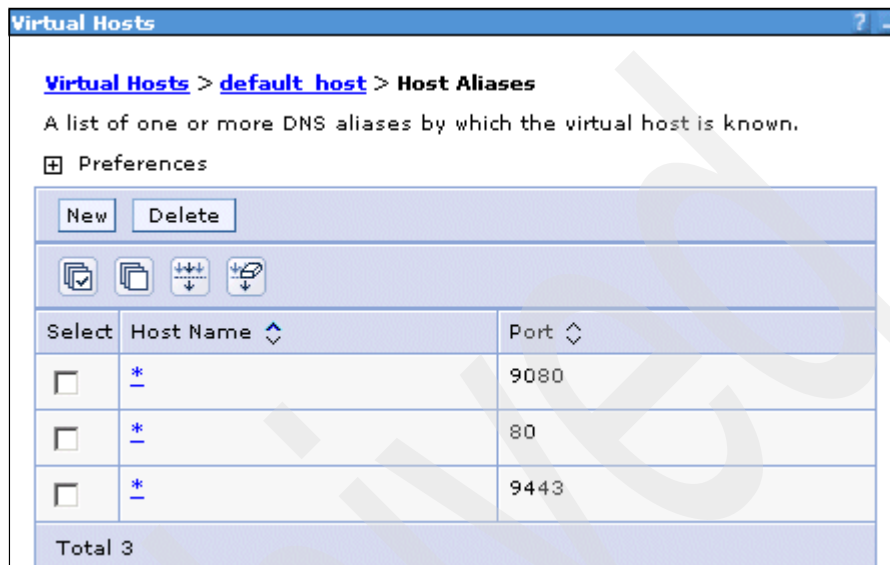


Figure 4-43 Default_host virtual host aliases

Note that the aliases are composed of a DNS host name and a port number. The host aliases for the default_host virtual host are *:80, *:9080 and *:9443, meaning any host name.

5. Combine the virtual host alias, context root and URL pattern to form the URL request of the snoop servlet. Requests for the servlet with any of the following URLs will map to the default_host virtual host:

```
http://<hostname>:80/snoop
http://<hostname>:9080/snoop
https://<hostname>:9443/snoop
```

4.9 Managing your configuration files

This section summarizes some of the most common system management tasks:

- ▶ Backing up a profile
- ▶ Restoring a profile
- ▶ Exporting and importing profiles

4.9.1 Backing up a profile

Use the **backupConfig** command to back up a profile. The command will zip the configuration file and store it in the current directory or a specified file name. The zip file can be restored using the **restoreConfig** command. By default, **backupConfig** will stop all servers in the configuration before performing the backup.

- ▶ Executing **backupConfig** from the *<was_home>/bin* directory without the **-profileName** parameter will back up the default directory.
- ▶ Executing **backupConfig** from the *<profile_home>/bin* directory without the **-profileName** parameter will back up that profile.
- ▶ To back up a node configuration, specify the node profile in the **-profileName** parameter.
- ▶ To back up a cell configuration, specify the deployment manager profile in the **-profileName** parameter.
- ▶ To back up a stand-alone application server, specify the application server profile in the **-profileName** parameter.

Syntax:

```
backupConfig <backup_file> [options]
```

The **backup_file** parameter specifies the file where the backup is to be written. If you do not specify a backup file name, a unique name is generated and the file is stored in the current directory. If you specify a backup file name in a directory other than the current directory, the specified directory must exist.

The parameters are shown in Table 4-17.

Table 4-17 *backupConfig* parameters

Parameter	Description
-nostop	Servers are not to be stopped before backing up the configuration.
-quiet	Suppresses the printing of progress information.
-logfile <fileName>	Name of the log file to which information gets written. The default is <i><profile_home>/logs/backupConfig.log</i>
-profileName <profile>	Profile to run the command against. If the command is run from <i><was_home>/bin</i> and -profileName is not specified, the default profile is used. If run from <i><profile_home>/bin</i> , that profile is used.
-replacelog	Replaces the log file instead of appending to the current log.
-trace	Generates trace information into the log file for debugging purposes.

Parameter	Description
-username <name>	User name for authentication if security is enabled in the server.
-password <password>	Specifies the password for authentication if security is enabled.
-help or -?	Prints command syntax information.

Example

Example 4-13 shows an example of backing up a deployment manager.

Example 4-13 backupConfig example

```

C:\WebSphere\ND\bin>backupConfig c:\WASbackups\Dmgr01\Dmgr01Aug2506
-profileName Dmgr01 -logfile c:\WASbackups\logs\Dmgr01Aug2506
ADMU0116I: Tool information is being logged in file
           c:\WASbackups\logs\Dmgr01Aug2506
ADMU0128I: Starting tool with the Dmgr01 profile
ADMU5001I: Backing up config directory C:\WebSphere\ND\profiles\Dmgr01\config
           to file C:\WASBackups\Dmgr01\Dmgr01Aug2506
ADMU0505I: Servers found in configuration:
ADMU0506I: Server name: dmgr
ADMU2010I: Stopping all server processes for node kadm028CellManager01
.....
ADMU5002I: 627 files successfully backed up.

```

4.9.2 Restoring a profile

Use the **restoreConfig** command to restore a profile configuration using an archive previously generated using **backupConfig**. If the configuration to be restored exists, the config directory is renamed to config.old (then config.old_1, etc.) before the restore begins. The command then restores the entire contents of the *<profile_home>/config* directory. By default, all servers on the node stop before the configuration restores so that a node synchronization does not occur during the restoration.

- ▶ Executing **restoreConfig** from the *<was_home>/bin* directory without the **-profileName** parameter will restore the default directory.
- ▶ Executing **restoreConfig** from the *<profile_home>/bin* directory without the **-profileName** parameter will restore that profile.

Syntax:

```
restoreConfig <backup_file> [options]
```


where backup_file specifies the file to be restored. If you do not specify one, the command will not run.

The parameters are shown in Table 4-18.

Table 4-18 restoreConfig parameters

Parameter	Description
-nowait	Do not wait for the servers to be stopped before backing up the configuration.
-quiet	Suppresses the printing of progress information.
-location <directory_name>	Location of the backup file.
-logfile <fileName>	Location of the log file to which information gets written. The default is <profile_home>/logs/backupConfig.log.
-profileName <profile>	Profile to run the command against. If the command is run from <was_home>/bin and -profileName is not specified, the default profile is used. If run from <profile_home>/bin, that profile is used.
-replacelog	Replaces the log file instead of appending to the current log.
-trace	Generates trace information into the log file for debugging purposes.
-username <name>	User name for authentication if security is enabled in the server.
-password <password>	Specifies the password for authentication if security is enabled.
-help or -?	Prints command syntax information.

Example

Example 4-14 shows an example of restoring an application server profile.

Example 4-14 restoreConfig example

```
C:\<was_base>\bin>restoreconfig d:\wasbackups\appsrv01Nov022004 -profileName
AppSrv01
ADMU0116I: Tool information is being logged in file
           C:\WebSphere\AppServer\profiles\AppSrv01\logs\restoreConfig.log
ADMU0128I: Starting tool with the AppSrv01 profile
ADMU0505I: Servers found in configuration:
ADMU0506I: Server name: server1
ADMU2010I: Stopping all server processes for node AppSrvNode01
ADMU0512I: Server server1 cannot be reached. It appears to be stopped.
ADMU5502I: The directory C:\WebSphere\AppServer\profiles\AppSrv01\config
           already exists; renaming to
           C:\WebSphere\AppServer\profiles\AppSrv01\config.old
ADMU5504I: Restore location successfully renamed
ADMU5505I: Restoring file d:\wasbackups\appsrv01Nov022004 to location
           C:\WebSphere\AppServer\profiles\AppSrv01\config
.....
ADMU5506I: 187 files successfully restored
ADMU6001I: Begin App Preparation -
ADMU6009I: Processing complete.
```

4.9.3 Exporting and importing profiles

WebSphere Application Server V6.x provides a mechanism that allows you to export certain profiles, or server objects from a profile to an archive. The archive can be distributed and imported to other installations.

An exported archive is a zip file of the config directory with host-specific information removed. The recommended extension of the zip file is .car. The exported archive can be the complete configuration or a subset. Importing the archive creates the configurations defined in the archive.

The target configuration of an archive export / import can be a specific server or an entire profile.

To use an archive, you would:

1. Export a WebSphere configuration. This creates a zip file with the configuration.
2. Unzip the files for browsing or update for use on other systems. For example, you might need to update resource references.

3. Send the configuration to the new system. An import can work with the zip file or with the expanded format.
4. Import the archive. The import process requires that you identify the object in the configuration you want to import and the target object in the existing configuration. The target can be the same object type as the archive or its parent:
 - If you import a server archive to a server configuration, the configurations are merged.
 - If you import a server archive to a node, the server is added to the node.

Server archives

The following command in **wsadmin** can be used to create an archive of a server:

```
$AdminTask exportServer {-archive <archive_location> -nodeName <node>
-serverName <server>}
```

This process removes applications from the server that you specify, and breaks the relationship between the server that you specify and the core group of the server, cluster, or bus membership. If you export a single server of a cluster, the relation to the cluster is eliminated.

To import a server archive, use the following command:

```
$AdminTask importServer {-archive <archive_location> [-nodeInArchive
<node>] [-serverInArchive <server>] [-nodeName <node>] [-serverName
<server>]}
```

When you use the **importServer** command, you select a configuration object in the archive as the source and select a configuration object on the system as the target. The target object can match the source object or be its parent. If the source and target are the same, the configurations are merged.

Profile archives

You can create a configuration archive (CAR) file containing the configuration of a stand-alone application server profile for later restoration. A CAR file can be used to clone the original profile to another machine or system. CAR files can be bundled in a customized installation package for use with the Installation Factory feature. For more information about using the Installation Factory, refer to the Information Center.

You can only create an archive of an unfederated profile (standalone application server).

The following commands in **wsadmin** can be used to create an archive of a profile:

```
$AdminTask exportWasprofile {-archive <archive_location>}
```

Administration with scripting

In this chapter, we introduce the WebSphere scripting solution called **wsadmin** and describe how some of the basic tasks that are performed by WebSphere administrators can be done using the scripting solution. There are two types of tasks: the operational task and the configurational task. The operational tasks deal with currently running objects in WebSphere installation and the configurational tasks deal with the configuration of WebSphere installations.

This chapter contains the following topics:

- ▶ Overview of WebSphere scripting
- ▶ Using wsadmin
- ▶ Common operational tasks using wsadmin
- ▶ Common configuration tasks
- ▶ Help creating wsadmin scripts
- ▶ Using Java for administration

The system management operations used in WebSphere Application Server V6.1 are based on the model used on V.6.0.x. All V.6.0.x commands continue to work as before. There have been some improvements, as we will point out in the relevant sections.

The examples shown in this chapter were written in the Jython language.

5.1 Overview of WebSphere scripting

WebSphere Application Server provides a scripting interface based on the *Bean Scripting Framework (BSF)* called *wsadmin*. BSF is an open source project to implement an architecture for incorporating scripting into Java applications and applets. The BSF architecture works as an interface between Java applications and scripting languages. Using BSF allows scripting languages to do the following:

- ▶ Look up a pre-registered bean and access a pre-declared bean
- ▶ Register a newly created bean
- ▶ Perform all bean operations
- ▶ Bind events to scripts in the scripting language

Because *wsadmin* uses BSF, it can make various Java objects available through language-specific interfaces to scripts. Figure 5-1 shows the major components involved in the *wsadmin* scripting solution.

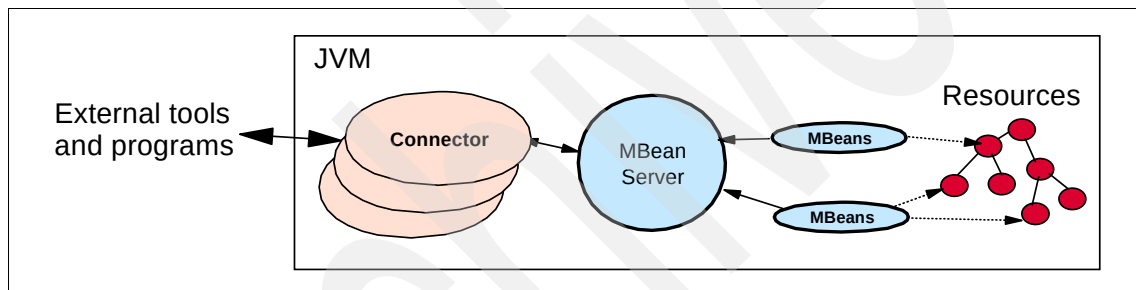


Figure 5-1 *wsadmin* scripting

5.2 Using wsadmin

In this section, we describe how to configure and launch *wsadmin*. We then describe the *wsadmin* objects used to manage WebSphere Application Server.

5.2.1 Jacl versus Jython

Jacl deprecated (new): WebSphere Application Server V6.1 represents the start of the deprecation process for the Jacl syntax.

Both the Jacl and Jython scripting languages are supported by the *wsadmin* tool. This chapter shows scripting using Jython.

If you have existing Jacl scripts and would like to start migrating to Jython, the Application Server Toolkit V6.1 provides a tool that converts the Jacl scripts to Jython, *Jacl2Jython*. In most cases, the resulting conversion is syntactically equivalent, and is usually run time equivalent.

However Jacl and Jython language differences can result in a few lines of code that are difficult to automatically convert, and in almost all such cases these preliminary converted lines are flagged `#!PROBLEM?`. This helps developer's focus on manual verification or alteration of these lines to ensure the intended run time result is maintained. While the developer needs to manually review and verify all the converted script, the `#!PROBLEM?` comment flags help identify the most likely problem lines.

Jacl2Jython is a conversion assistant which typically does 95-98% of a preliminary conversion, but the developer must manually verify all of the preliminary conversion, and typically must also manually convert or modify some code to make it function as originally intended. Even if the preliminary conversion superficially appears correct, it always requires a complete line-by-line manual review and verification.

5.2.2 Launching wsadmin

The **wsadmin.bat** (Windows) or **.sh** (UNIX) command file resides in the bin directory of every profile for an application server, deployment manager, and managed node instance. Start **wsadmin** from a command prompt with the command:

```
<was_home>\profiles\<profile_name>\bin\wsadmin.bat (.sh)
```

Note that the **wsadmin** command also exists in the bin directory of every `<profile_home>` directory. Starting **wsadmin** from this location is not recommended because you have to be very careful to specify the right profile to work with. The default profile will be chosen.

To get syntax-related help, type **wsadmin.bat -?** and press Enter. Example 5-1 shows the output. Some of these options have an equivalent in the properties file. Any options specified on the command line will override those set in the properties file.

Example 5-1 wsadmin command-line options

```
C:\<was_home>\profiles\<profile_name>\bin>wsadmin -?
wsadmin
[ -h(elp) ]
[ -? ]
[ -c <command> ]
[ -p <properties_file_name>]
[ -profile <profile_script_name>]
[ -f <script_file_name>]
[ -javaoption java_option]
[ -lang language]
[ -wsadmin_classpath classpath]
[ -profileName profile]
[ -conntype
    SOAP
        [-host host_name]
        [-port port_number]
        [-user userid]
        [-password password] |
    RMI
        [-host host_name]
        [-port port_number]
        [-user userid]
        [-password password] |
    NONE
]
[ -jobid <jobid_string>]
[ -tracefile <trace_file>]
[ -appendtrace <true/false>]
[ script parameters ]
```

5.2.3 Configuring wsadmin

The properties that determine the scripting environment for **wsadmin** can be set using either the command line or a properties file. Properties can be set in the following three ways:

- Use the profile or system default properties file:
 <profile_home>/properties/wsadmin.properties
 or

<was_home>/properties/wsadmin.properties

- Use a customized properties file placed in the location pointed to by the WSADMIN_PROPERTIES environment variable. You can copy the default properties file to this location and modify it.
- Specify the -p argument to the **wsadmin** command.

The properties to note are listed in Table 5-1.

Table 5-1 *wsadmin properties*

Property	Value
com.ibm.ws.scripting.connectionType	SOAP, RMI or NONE
com.ibm.scripting.port	TCP port of target system
com.ibm.scripting.host	Host name of target system
com.ibm.ws.scripting.defaultLang	Jython or Jacl
com.ibm.ws.scripting.echoparams	Determines whether parameters or arguments are output to STDOUT or to the wsadmin trace file
com.ibm.ws.scripting.traceFile	File for trace information
com.ibm.ws.scripting.validationOutput	Location of validation reports
com.ibm.ws.scripting.traceString	=com.ibm.*=all=enabled
com.ibm.ws.scripting.appendTrace	Appends to the end of the existing log file
com.ibm.ws.scripting.profiles	List of profiles to be run before running user commands, scripts, or an interactive shell
com.ibm.ws.scripting.emitWarningForCustomSecurityPolicy	Controls whether message WASX7207W is emitted when custom permissions are found
com.ibm.ws.scripting.tempdir	Store temporary files when installing applications
com.ibm.ws.scripting.validationLevel	Level of validation to use when configuration changes are made from the scripting interface
com.ibm.ws.scripting.crossDocumentValidationEnabled	Determines whether the validation mechanism examines other documents when changes are made to one document

Property	Value
<code>com.ibm.ws.scripting.classpath</code>	List of paths to search for classes and resources

Some of the listed properties in the **wsadmin.properties** file are commented out by default. An example is `com.ibm.ws.scripting.traceString`. If you want to trace **wsadmin** execution, remove the comment sign `#` from the properties file.

Similarly, some of the properties contain values. For example, `com.ibm.ws.scripting.connectionType` has a default value of `SOAP`. This means that when a scripting process is invoked, a SOAP connector is used to communicate with the server.

The **wsadmin** command can operate in either connected or local mode. In connected mode, all operations are performed by method invocations on running JMX MBeans. In local mode, the application server (MBeans server) is not started and the **wsadmin** objects are limited to configuring the server by means of directly manipulating XML configuration documents. When operating in local mode, it is very important to specify the correct profile for performing the administration tasks or starting the tool from the correct profile directory. Remember that each application server instance is configured from a set of XML documents that is stored in separate directories for every server instance (the application server profile).

When performing configuration changes in local mode in a distributed server environment, care should be taken to make configuration changes at the deployment manager level. Changes made directly to the node configuration will be lost at server startup or at configuration replication.

In addition to the properties file and configuration profile, you should also take note of the script profile file. This is not to be confused with the server configuration profile. A script profile is a script that is invoked before the main script or before invoking **wsadmin** in interactive mode. The purpose of the script profile is to customize the environment on which scripts run. For example, a script profile can be set for Java Command Language (Jacl) scripting language that makes Jacl-specific variables or procedures available to the interactive session or main script.

5.2.4 Command and script invocation

The **wsadmin** commands can be invoked in three different ways. This section details the different ways in which command invocation is performed.

Invoking a single command (-c)

The `-c` option is used to execute a single command using `wsadmin` in Example 5-2. In the example, we use the `AdminControl` object to query the node name of the WebSphere server process.

Example 5-2 Running a single command in wsadmin

```
C:\<was_home>\profiles\<profile_name>\bin>wsadmin -c AdminControl.getNode()  
WASX7209I: Connected to process "dmgr" on node kcgglf3CellManager01 using SOAP  
connector; The type of process is: DeploymentManager  
'kcgglf3CellManager01'
```

Invoking commands interactively

The command execution environment can be run in interactive mode, so you can invoke multiple commands without having the overhead of starting and stopping the `wsadmin` environment for every single command. Run the `wsadmin` command without the command (`-c`) and script file (`-f`) options to start the interactive command execution environment, as shown in Example 5-3.

Example 5-3 Starting the wsadmin interactive command execution environment

```
C:\<was_home>\profiles\<profile_name>\bin>wsadmin  
WASX7209I: Connected to process "dmgr" on node kcgglf3CellManager01 using SOAP  
connector; The type of process is: DeploymentManager  
WASX7031I: For help, enter: "print Help.help()"  
wsadmin>
```

From the `wsadmin>` prompt, the WebSphere administrative objects and built-in language objects can be invoked, as shown in Example 5-4. Type the commands as shown in bold.

Example 5-4 Interactive command invocation

```
wsadmin>AdminControl.getNode()  
'kcgglf3CellManager01'  
wsadmin>
```

End the interactive execution environment by typing **quit** and pressing the Enter key.

Running script files (-f)

The `-f` option is used to execute a script file. Example 5-5 shows a two-line Jython script named `myScript.py`. The script has a `.py` extension to reflect the Jython language syntax of the script. The extension plays no significance in `wsadmin`; the `com.ibm.ws.scripting.defaultLang` property and `-lang` command-line option is used to determine the language used. If the property setting is not correct, use the `-lang` option to identify the scripting language, because the default is Jacl.

Example 5-5 Jython script

```
print "This is an example Jython script"
print ""+ AdminControl.getNode()+""
```

Example 5-6 shows how to execute the script.

Example 5-6 Running a Jython script in wsadmin

```
C:\<was_home>\profiles\<profile_name>\bin>wsadmin -f myScript.py
WASX7209I: Connected to process "dmgr" on node kcgglf3CellManager01 using SOAP
connector; The type of process is: DeploymentManager
This is an example Jython script
kcgglf3CellManager01
```

Using a profile (-profile)

The `-profile` command-line option can be used to specify a profile script. The profile can be used to perform whatever standard initialization is required. Several `-profile` options can be used on the command line and those are invoked in the order given.

Specifying a properties file (-p)

Use the `-p` option to specify a properties file other than `wsadmin.properties` either located in the `<profile_home>/properties` directory, `<was_home>/properties` directory, or in the `$user_home` directory.

Figure 5-7 shows an example of invoking `wsadmin` to execute a script file using a specific properties file.

```
C:\<was_home>\profiles\<profile_name>\bin>wsadmin -f c:\myScript.py -p
c:\temp\custom.properties
WASX7209I: Connected to process "dmgr" on node kcgglf3CellManager01 using SOAP
connector; The type of process is: DeploymentManager
This is an example Jython script
kcgglf3CellManager01
```

5.2.5 Overview of wsadmin objects

The **wsadmin** command exposes four objects used for managing the WebSphere environment, as well as a help object:

- ▶ AdminControl
- ▶ AdminConfig
- ▶ AdminApp
- ▶ AdminTask
- ▶ Help

AdminControl

The AdminControl scripting object is used for operational control. It communicates with MBeans that represent live objects running a WebSphere server process. It includes commands to query existing running objects and their attributes and invoke operations on the objects. In addition to the operational commands, the AdminControl object supports commands to query information about the connected server, convenient commands for client tracing, reconnecting to a server, and starting and stopping a server.

AdminConfig

The AdminConfig object is used to manage the configuration information that is stored in the repository. This object communicates with the WebSphere Application Server configuration service component to make configuration inquiries and changes. You can use it to query existing configuration objects, create configuration objects, modify existing objects, and remove configuration objects. In a distributed server environment, the AdminConfig commands are available only if a scripting client is connected to the deployment manager. When connected to a node agent or a managed application server, the AdminConfig commands will not be available because the configuration for these server processes are copies of the master configuration that resides in the deployment manager.

AdminApp

The AdminApp object can update application metadata, map virtual hosts to Web modules, and map servers to modules for applications already installed.

Changes to an application, such as specifying a library for the application to use or setting session management configuration properties, are performed using the AdminConfig object.

AdminTask

The AdminTask object is used to access a set of task-oriented administrative commands that provide an alternative way to access the configuration commands and the running object management commands. The administrative commands run simple and complex commands. The administrative commands are discovered dynamically when the scripting client is started. The set of available administrative commands depends on the edition of WebSphere Application Server you install. You can use the AdminTask object commands to access these commands.

Two run modes are always available for each administrative command, namely the batch and interactive mode. When you use an administrative command in interactive mode, you go through a series of steps to collect your input interactively. This process provides users a text-based wizard and a similar user experience to the wizard in the administrative console. You can also use the help command to obtain help for any of the administrative commands and the AdminTask object.

Help

The Help object provides information about the available methods for the four management objects as well as information about operations and attributes of running MBeans. For example, to get a list of the public methods available for the AdminControl object, enter the command as shown:

```
wsadmin>print Help.AdminControl()
```

To get a detailed description of a specific object method and the parameters it requires, invoke the help method of the target object with the method name as the option to the help method, as shown in Example 5-8 on page 259.

Example 5-8 Getting method-specific help

```
wsadmin>print AdminControl.help('completeObjectName')  
WASX7049I: Method: completeObjectName
```

Arguments: object name, template

Description: Returns a String version of an object name that matches the "template." For example, the template might be "type=Server,*"
If there are several MBeans that match the template, the first match

Similarly, you can get a detailed methods help for the AdminConfig, AdminApp, and AdminTask objects.

Obtaining operations and attributes information from the Help object are discussed in "Finding attributes and operations for running MBeans" on page 262.

Execution environment

The AdminConfig, the AdminTask, and the AdminApp objects all handle configuration functionality. You can invoke configuration functions with or without being connected to a server. Only the AdminControl object requires the server to be started because its commands can only be invoked on running JMX MBeans.

If a server is running, it is not recommended that the scripting client be started in local mode because configuration changes made in local mode are not reflected in the running server configuration. The reverse is also true. In connected mode, the availability of the AdminConfig commands depend on the type of server to which the scripting client is attached to. Performing configuration changes to a node agent or managed application server is not advised.

Note: For the purposes of this discussion, we will refer to the methods of the AdminControl, AdminConfig, AdminApp, AdminTask, and Help objects as *commands*.

5.2.6 Management using wsadmin objects

Administration can be performed from **wsadmin** on JMX MBean objects from the AdminControl object. Configuration management is done with the AdminConfig object. the AdminTask is used for performing common types of administrative and configurative tasks without in-depth knowledge of the JMX framework and the WebSphere XML configuration structure. The following sections explain these **wsadmin** objects in more detail.

Administration using AdminControl

In order to invoke administrative methods on running JMX MBeans, a reference to the target MBean object is required, by means of an *Object Name*. As explained previously, MBeans represent running components in the WebSphere run time environment and can be used to query and alter state and configuration. Each WebSphere server instance contains an MBean server that registers and provides the run time environment for all MBeans in that server.

Use the **queryNames** command to list the object names of all MBeans registered and running in the MBean server. The simplest form of this command in Jython is:

```
AdminControl.queryNames('*')
```

The list contains all object names of all MBeans currently running in the MBean server. Depending on the server your scripting client is attached to, this list might contain MBeans that are running in remote servers. This is because every MBean server provides management capabilities for all the node agents and managed application servers that is manageable from this level in the cell hierarchy. The MBeans running on the remote MBean server are manageable by means of a proxy MBean, transparent to the scripting client.

- ▶ If the client is attached to a stand-alone WebSphere Application Server, the list contains only MBeans running on that server.
- ▶ If the client is attached to a node agent, the list contains MBeans running in the node agent as well as MBeans running on all application servers on that node.
- ▶ If the client is attached to a deployment manager, the list contains MBeans running in the deployment manager, in all node agents communicating with that deployment manager, and all application servers on all the nodes served by those node agents.

Example 5-9 on page 261 shows a Jython script that collects information about running MBeans into a file named `mbean.txt`. The list returned by the **queryNames** command is a single Jython string object that separates every object name with two new line control characters for clear readability. The new line character is used for creating a Jython list structure that is written to the `mbean.txt` file with an `ObjectName: prefix`. Note that because the list is created based on new line (`line.separator`) information, every other entry from the `mbList` object is empty.

Example 5-9 Finding information for running MBeans

```
file = "mbean.txt"
logFile = open( file, "a" )
mbStr = AdminControl.queryNames("*:*)
mbList = mbStr.split(java.lang.System.getProperty("line.separator"))
for item in mbList:
    if (item != ""):
        print >>logFile, "ObjectName: "+item
    #endIf
#endFor
logFile.close()
```

An example of object name item returned by the **queryNames** command could look like Example 5-10.

Example 5-10 Returned object name item

```
ObjectName:
WebSphere:name=dmgr,process=dmgr,platform=proxy,node=kcgg1f3CellManager01,j2eeType=J2EEServer,version=6.1.0.0,type=Server,mbeanIdentifier=cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr/server.xml#Server_1,cell=kcgg1f3Cell01,spec=1.0,processType=DeploymentManager
```

This represents a deployment manager (dmgr) running in cell kcgg1f3Cell01 on node kcgg1f3CellManager01. WebSphere includes the following key properties on its object names:

- ▶ Name
- ▶ Type
- ▶ Cell
- ▶ Node
- ▶ Process
- ▶ mbeanIdentifier

You can use any of these key properties to narrow the scope of the list returned by **queryNames**. For example you can list all MBeans that represent server objects on the node kcgg1f3CellManager01, as follows:

```
wsadmin>AdminControl.queryNames('WebSphere:type=Server,node=kcgg1f3CellManager01,*')
```

Note: Be aware of the following when using **AdminControl.queryNames**.

- ▶ You will get an empty list back if you do not use the ***** wildcard at the end of the **ObjectName**.
- ▶ **WebSphere:** represents the domain and is assumed if you do not include it.

An alternative way to obtain the object name is by using the **completeObjectName** command. This command only returns the first object name matching the pattern specified. For patterns specifying the exact object needed or the *top level* MBean, for example, the deployment manager, the **completeObjectName** command could be a better choice. For example, this command would obtain the deployment manager object name:

```
wsadmin>AdminControl.completeObjectName('type=DeploymentManager,node=kcgg1f3Cel
1
```

Javadoc™: All MBean types are documented in the Javadoc format in the web\mbeanDocs directory from within the WebSphere target installation directory. The starting point is the index.html file. For a default installation, the location of the index.html file is in this directory in a Windows environment:

```
C:\<was_home>\web\mbeanDocs\index.html
```

Finding attributes and operations for running MBeans

The Help object can be used to list attributes and operations available for any running MBean. The object name of the running MBean is needed in order to complete the query. The object name can be obtained by use of the AdminControl **completeObjectName** command.

Example 5-11 shows how to find attributes information for a server MBean. The first command initializes the variable serv to the object name of a running server on the kcgg1f3CellManager01, as found by the **completeObjectName** command. Note that the object name returned is the first found by **completeObjectName**. The **attributes** command of the Help object lists all the available attributes for the particular server MBean found.

Example 5-11 Finding attributes for a running MBean

```
wsadmin>serv =
AdminControl.completeObjectName('type=Server,node=kcgg1f3CellManager01,*')
wsadmin>print Help.attributes(serv)
```

Attribute	Type	Access
name	java.lang.String	RO
shortName	java.lang.String	RO
threadMonitorInterval	int	RW
threadMonitorThreshold	int	RW
threadMonitorAdjustmentThreshold	int	RW
pid	java.lang.String	RO
cellName	java.lang.String	RO
cellShortName	java.lang.String	RO
deployedObjects	java.lang.String;	RO
javaVMs	java.lang.String;	RO
nodeName	java.lang.String	RO

nodeShortName	java.lang.String	RO
processType	java.lang.String	RO
resources	java.lang.String;	RO
serverVersion	java.lang.String	RO
serverVendor	java.lang.String	RO
state	java.lang.String	RO
platformName	java.lang.String	RO
platformVersion	java.lang.String	RO
internalClassAccessMode	java.lang.String	RO
objectName	java.lang.String	RO
stateManageable	boolean	RO
statisticsProvider	boolean	RO
eventProvider	boolean	RO
eventTypes	java.lang.String;	RO

Attribute values for any specific MBean can be read with the **getAttribute** command of the AdminControl object. Depending on the access policy for the individual attribute (Read only (RO) or Read and Write (RW), as listed with the **attributes** Help command), attribute values can be modified with the **setAttribute** command. For example, the process ID (pid) from the server MBean can be retrieved by running:

```
wsadmin>AdminControl.getAttribute(serv,'pid')
```

Similar to the **attributes** command, the **operations** command can be used to list the operations supported by a particular MBean. Example 5-12 shows the usage of the **operation** command and its output.

Example 5-12 Finding operations for a running MBean (partial list of operations)

```
wsadmin>print Help.operations(serv)
Operation
java.lang.String getName()
java.lang.String getShortName()
int getThreadMonitorInterval()
void setThreadMonitorInterval(int)
int getThreadMonitorThreshold()
void setThreadMonitorThreshold(int)
int getThreadMonitorAdjustmentThreshold()
void setThreadMonitorAdjustmentThreshold(int)
java.lang.String getPid()
java.lang.String getCellName()
java.lang.String getCellShortName()
java.lang.String; getDeployedObjects()
void stopImmediate()
void stop(java.lang.Boolean, java.lang.Integer)
void restart()
java.lang.String getObjectNamesStr()
boolean isStateManageable()
boolean isStatisticsProvider()
boolean isEventProvider()
java.lang.String; getEventTypes()
```

MBean operations are invoked by use of the **invoke** command of the AdminControl object. For example, this is the syntax for invoking the `getVersionsForAllProducts` operation:

```
wsadmin>print AdminControl.invoke(serv,'getVersionsForAllProducts')
```

For viewing and invoking MBean attributes and operations visually, the graphical tool *MBeanInspector (MBI)* is recommended. With MBeanInspector, all JMX MBeans are listed in a parent-child tree structure and with the **wsadmin** invocation syntax displayed for most operations.

Even though MBI was not available for WebSphere Application Server Version 6.x, the current version for Version 5 works fine with Version 6. However, MBI is not profile-aware. With security enabled, it uses the generic `sas.properties` file from the root of the WebSphere install tree instead of the `sas.properties` file from the current profile. For more information, see *MBeanInspector for WebSphere Application Server* on alphaWorks® at:

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

Configuring using AdminConfig

The AdminConfig and AdminTask objects are used to manage configuration information for the WebSphere environment. This section discusses the use of the AdminConfig object.

The AdminConfig object communicates with the configuration service of the WebSphere process to query and update the configuration. All modifications done with the AdminConfig commands are stored to a temporary workspace until you invoke the **save** command.

The AdminConfig object performs a series of tasks for configuration changes:

1. Identify the configuration type and the corresponding attributes.
2. Query an existing configuration object to obtain the configuration ID of the object to modify.
3. Modify the existing configuration object or overwrite with a new object.
4. Save the configuration.

The next sections discuss these steps in more detail. Be warned that configuring WebSphere by use of the AdminConfig object requires a good understanding of the WebSphere XML configuration documents and the config directory content. A starting point would be to look through a default WebSphere configuration profile and understand the defined elements, attributes, and namespaces listed in the Javadoc configuration documentation.

The types command

The WebSphere configuration consists of element types and attribute names structured in a set of XML documents. The WebSphere configuration is managed from the AdminConfig object by obtaining a reference to an existing element type or by instantiating or removing element types from the configuration. In **wsadmin**, every element type is managed as a configuration object with a unique configuration ID. All available configuration objects can be listed by using the **types** command. Example 5-13 shows the partial output of the **types** command.

Example 5-13 Partial output of types command

```
wsadmin>print AdminConfig.types()
AccessPointGroup
ActivationSpec
ActivationSpecTemplateProps
ActivitySessionService
AdminObject
AdminObjectTemplateProps
AdminServerAuthentication
AdminService
Agent
AllActivePolicy
AllAuthenticatedUsersExt
Application
ApplicationClientFile
ApplicationConfig
ApplicationContainer
ApplicationDeployment
ApplicationManagementService
ApplicationProfileService
ApplicationServer
```

Every configuration object is used for configuring a specific part of the overall cell. For example, the **ApplicationServer** object is used for defining application servers in the environment. As the application server provides configurable features, attributes defined from within the object are used to configure the application server features. The available attributes for the **ApplicationServer** object can be listed by use of the **AdminConfig attributes** command, this will be discussed in detail in “Input and output of configuration object attributes” on page 269.

An object can contain other objects. Therefore, a parent-to-child relationship exists in the configuration. For example, a node type object contains server type objects, making the node object a parent to the server objects. To identify possible objects in where a given configuration object can reside, use the **parents** command. Locate the parent configuration objects for the **ApplicationServer** object by issuing:

```
wsadmin>AdminConfig.parents('ApplicationServer')
```

The getid command

The **getid** command returns the configuration name for a configuration object. Configuration objects are named with a combination of the display name for the object and its configuration ID. The ID uniquely identifies the object and can be used in any configuration command that requires a configuration object name.

Example 5-14 shows how to obtain the configuration name for **dmgr**. The string argument passed to the command identifies the node and server to get the name for. The **/** is used to separate one set of object type and value from another. The **:** is used to separate the value from the object type in an object type and value pair.

Example 5-14 Finding configuration name of an object

```
wsadmin>AdminConfig.getid('/Node:kcgg1f3CellManager01/Server:dmgr/')  
'dmgr(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#Server_1)'
```

Example 5-14 illustrates how the parent-to-child relationship of configuration objects comes into play. As the configuration object name for the **dmgr** residing on the **kcgg1f3CellManager01** is needed, the specification of both child and parent objects are required.

Note: Configuration objects are named using a combination of the display name and its configuration ID. The display name comes first, followed by the configuration ID in parentheses. An example of such an object name is:

```
server1(cells/MyCell/nodes/MyNode/servers/server1|server.xml#Server_1)
```

For those pieces of configuration data that do not have display names, the name of the object simply consists of the configuration ID in parentheses. An example of such an object name is as follows:

```
(cells/MyCell/nodes/MyNode/servers/server1|server.xml#ApplicationServer_1)
```

Because the ID portion is completely unique, a user can always use it without the prepended display name in any command that requires a configuration object name.

The list command

The **list** command returns a list of objects for a given type. In a WebSphere Application Server environment, there are several object types and many objects configured that have the same object type.

Example 5-15 list all objects of the DataSource object type in the test environment. The list command returns two objects of the DataSource type to the server1 server. Note how this command lists all objects regardless of scope. From the administrative console, you would have to collect this information by querying at each scope level.

Example 5-15 Finding objects of the same object type

```
wsadmin>print AdminConfig.list('DataSource')
"Default
Datasource(cells/kcgg1f3Cell01/nodes/kcgg1f3Node01/servers/server1|resources.xml#DataSource_1153406381923)"
DefaultEJBTimerDataSource(cells/kcgg1f3Cell01/nodes/kcgg1f3Node01/servers/server1|resources.xml#DataSource_1000001)
```

The defaults command

The **defaults** command displays a table of attributes, their types, and defaults, if any, for the configuration object. Each object has an object type and each object type has attributes that might or might not have default values.

Example 5-16 shows the usage of the **defaults** command to list the attributes and default values for those attributes for object type DynamicCache.

Example 5-16 Finding attributes and default values for an object type

```
wsadmin>print AdminConfig.defaults('DynamicCache')
```

Attribute	Type	Default
enable	boolean	false
defaultPriority	int	1
hashSize	int	0
cacheSize	int	2000
enableCacheReplication	boolean	false
replicationType	ENUM	NONE
pushFrequency	int	1
enableDiskOffload	boolean	false
diskOffloadLocation	String	
flushToDiskOnStop	boolean	false
enableTagLevelCaching	boolean	false
diskCachePerformanceLevel	ENUM	BALANCED
diskCacheSizeInGB	int	0
diskCacheSizeInEntries	int	0
diskCacheEntrySizeInMB	int	0
diskCacheCleanupFrequency	int	0

context	ServiceContext
properties	Property
cacheGroups	ExternalCacheGroup
cacheReplication	DRSSettings
diskCacheCustomPerformanceSettings	DiskCacheCustomPerformanceSettings
diskCacheEvictionPolicy	DiskCacheEvictionPolicy

Input and output of configuration object attributes

The AdminConfig **attributes** command is part of the **wsadmin** online help feature. The information displayed does not represent any particular configuration object, but represents configuration object types or object metadata. The metadata is used to show, modify, and create real configuration objects. In this section, we discuss the interpretation of the output of those commands.

The **attributes** command displays the type and name of each attribute defined for a given type of configuration object. The name of each attribute is always a string, generally beginning with a lowercase letter. But the types of attributes vary.

Example 5-17 shows the output of the **attributes** command for the configuration object DynamicCache. There are 15 attributes listed, four simple integer attributes, five Boolean attributes, and one String attribute.

The cacheGroups and properties objects are lists of objects indicated by * at the end of ExternalCacheGroup and Property(TypedProperty), respectively. These are nested attributes. Another attributes invocation can be used to see the composition of these nested attributes.

Example 5-17 Output of attribute command of AdminConfig object

```
wsadmin>print AdminConfig.attributes('DynamicCache')
cacheGroups ExternalCacheGroup*
cacheReplication DRSSettings
cacheSize int
context ServiceContext@
defaultPriority int
diskCacheCleanupFrequency int
diskCacheCustomPerformanceSettings DiskCacheCustomPerformanceSettings
diskCacheEntrySizeInMB int
diskCacheEvictionPolicy DiskCacheEvictionPolicy
diskCachePerformanceLevel ENUM(HIGH, CUSTOM, BALANCED, LOW)
diskCacheSizeInEntries int
diskCacheSizeInGB int
diskOffloadLocation String
enable boolean
enableCacheReplication boolean
enableDiskOffload boolean
enableTagLevelCaching boolean
flushToDiskOnStop boolean
hashSize int
properties Property(TypedProperty, DescriptiveProperty)*
pushFrequency int
replicationType ENUM(PULL, PUSH, PUSH_PULL, NONE)

wsadmin>print AdminConfig.attributes('ExternalCacheGroup')
members ExternalCacheGroupMember*
name String
type ENUM(SHARED, NOT_SHARED)

wsadmin>print AdminConfig.attributes('TypedProperty')
description String
name String
required boolean
type String
validationExpression String
value String
```

In Example 5-17, the properties attribute of the DynamicCache object has a value that is also a list of objects of the Property type. The Property type is a generic type, so its sub-types are listed, that is TypedProperty. The replicationType and diskCachePerformanceLevel attribute are an ENUM type attribute whose value must be one of the four possible values listed in parentheses.

The **show** command of the AdminConfig object can be used to display the top-level attributes of a given object. In Example 5-18, the top-level attributes for the SocratesServer1 object is shown by use of the **show** command.

Example 5-18 Finding top-level attributes for a given object

```
wsadmin>print
AdminConfig.show(AdminConfig.getid('/Node:kcgg1f3CellManager01/Server:dmgr/'))
[components
"[(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#NameS
erver_1) "Deployment
Manager(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#
CellManager_1)" "WorkloadManagement
Server(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#W
orkloadManagementServer_1)" "Network Deployment
Server(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#A
pplicationServer_1)"]"]
[customServices []]
[developmentMode false]
[errorStreamRedirect
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#StreamR
edirect_1)]
[name dmgr]
[outputStreamRedirect
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#StreamR
edirect_2)]
[parallelStartEnabled true]
[processDefinitions
[(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#JavaPr
ocessDef_1)]]
[serverType DEPLOYMENT_MANAGER]
[services
"[(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#PMISe
rvice_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#AdminSe
rvice_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#TraceSe
rvice_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#Diagnos
ticProviderService_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#RASLogg
ingService_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#CoreGro
upBridgeService_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#ObjectR
equestBroker_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#Transpo
rtChannelService_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#ThreadP
```

```
oolManager_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#HTTPAcc
essLoggingService_1)]"]
[stateManagement
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#StateMa
nageable_1)]
[statisticsProvider
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#Statist
icsProvider_1)]
```

The value for a particular attribute can be retrieved with the **showAttribute** command. In Example 5-19, the values for the name attribute and the services attribute of SocratesServer1 server object are listed.

Example 5-19 Retrieving attribute values for a given object

```
wsadmin>serv = AdminConfig.getid('/Node:kcgg1f3CellManager01/Server:dmgr/')

wsadmin>print AdminConfig.showAttribute(serv,'name')
dmgr

wsadmin>print AdminConfig.showAttribute(serv,'services')
[(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#PMISer
vice_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#AdminSe
rvice_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#TraceSe
rvice_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#Diagnos
ticProviderService_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#RASLogg
ingService_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#CoreGro
upBridgeService_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#ObjectR
equestBroker_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#Transpo
rtChannelService_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#ThreadP
oolManager_1)
(cells/kcgg1f3Cell01/nodes/kcgg1f3CellManager01/servers/dmgr|server.xml#HTTPAcc
essLoggingService_1)]
```

Another useful command to list all attributes and their values in the AdminConfig object is the **showall** command. This command returns names and values for all attributes of a given object, including nested attributes.

Tip: Scripting examples for managing the WebSphere Application Server configuration are available from the IBM *WebSphere Developer Domain (WSDD)* library in the samples collection. Even though the samples are for WebSphere Application Server V5, they are just as useful for WebSphere Application Server V6.1.

Configuring using AdminTask

Use of the AdminConfig and AdminControl requires some knowledge of the JMX framework and WebSphere XML configuration structure. For performing various scripted administrative tasks without knowledge of the underlying infrastructure, the AdminTask object has been introduced.

The AdminTask object commands are more like wizards, providing a step-by-step guide to performing management operations. The AdminTask commands can either be invoked interactively, in order to prompt the user for the parameters required, or be invoked batch-like, with all input specified as part of the invocation. The AdminTask commands offered are direct reflections of the tasks each component provides through the command framework. As the command set is discovered dynamically on `wsadmin` startup, the number of commands can differ, depending on the server environment and WebSphere Application Server package.

Overview of AdminTask commands

The AdminTask object provides a large number of commands that perform simple and complex administrative tasks. In order to find a command for a specific task, commands have been logically grouped into command groups.

For example, to find AdminTask commands related to service integration bus administration, the commands of the SIBAdminCommands group can be listed. All the command groups and the commands in the SIBAdminCommands group are listed in Example 5-20.

Example 5-20 AdminTask Groups and SIBAdmin commands (partial list)

```
wsadmin>print AdminTask.help('-commandGroups')
WASX8005I: Available admin command groups:

AdminReports - Admin configuration reports
AuthorizationGroupCommands - Authorization Group
AutoGen Commands - Commands for autogenerating LTPA password and server Id.
CertificateRequestCommands - Command that manage certificate request.
ChannelFrameworkManagement - A group of admin commands that help in configuring
the WebSphere Transport Channel Service
ClusterConfigCommands - Commands for configuring application server clusters
and cluster members.
ConfigArchiveOperations - A command group that contains various config archive
related operations.
ConfigLimits - No description available
CoreGroupBridgeManagement - A group of administrative commands that help in
configuring core groups.
CoreGroupManagement - A set of commands for modifying core groups
CreateWebServerByHostNameCommands - Specify the configuration properties for
IBM HTTP Server.
DescriptivePropCommands - Commands to configure Descriptive Properties.

wsadmin>print AdminTask.help('SIBAdminCommands')
WASX8007I: Detailed help for command group: SIBAdminCommands
Description: A group of commands that help configure SIB queues and messaging
engines.
Commands:
addSIBPermittedChain - Adds the specified chain to the list of permitted chains
for the specified bus.
addSIBusMember - Add a member to a bus.
createSIBDestination - Create bus destination.
createSIBEngine - Create a messaging engine.
createSIBForeignBus - Create a SIB foreign bus.
createSIBLink - Create a new SIB link.
createSIBMQLink - Create a new WebSphere MQ link.
createSIBMediation - Create a mediation.
createSIBWMQServer - Create a new WebSphere MQ server.
createSIBus - Create a bus.
deleteSIBDestination - Delete bus destination.
```

All the available AdminTask commands can be retrieved in one list with the **help** command:

```
wsadmin>print AdminTask.help('-commands')
```

For example, in order to invoke the **createSIBus** command, a number of options are needed. To list the options, invoke help on the command:

```
wsadmin>print AdminTask.help('createSIBus')
```

An example of creating a service integration bus interactively is shown in Example 5-21 on page 276. The batch invocation of the command is displayed at the end of the interactive guide with all the correct options added. This command can be used to form scripted creations of additional service integration buses. It is a means to help the script developer become familiar with the command invocation of the AdminTask object. Using the interactive approach for obtaining the correct invocation syntax can be very useful when developing automated scripted installations and configurations.

Tip: The AdminTask batch command syntax is displayed at the time of command invocation. To obtain the command syntax without changing the master WebSphere configuration repository, the change need not be saved from the local workspace to the repository. The change to the workspace can be reversed with use of the AdminConfig **reset** command:

```
wsadmin>AdminConfig.reset()
```

Example 5-21 Interactive invocation of AdminTask

```
wsadmin>AdminTask.createSIBus('-interactive')
Create a bus

Create a bus.

*Bus name (bus): WSBus
Description of bus (description): Web Services cell wide bus
Security (Deprecated) (secure): false
Inter-engine authentication alias (interEngineAuthAlias):
Mediations authentication alias (mediationsAuthAlias):
Protocol (protocol):
Discard messages after queue deletion (discardOnDelete): [false]
Max bus queue depth (highMessageThreshold):
Dynamic configuration reload enabled (configurationReloadEnabled): [true]
Enable bus security (busSecurity):
Script Compatibility (scriptCompatibility):

Create a bus

F (Finish)
C (Cancel)

Select [F, C]: [F] F
WASX7278I: Generated command line: AdminTask.createSIBus('[-bus WSBus
-description "Web Services cell wide bus" -secure false ]')
'WSBus(cells/kcgg1f3Cell101/buses/WSBus|sib-bus.xml#SIBus_1153949210676)'
```

As some configuration tasks are dependent on other resources to exist, the task commands can provide a means for configuring related resources for completing the intended task. Such tasks are split into steps. An example of a multi-step task is the **createCluster** command, which provides steps to create a replication domain and convert servers to cluster members as part of the cluster creation. See the help text for the **createCluster** command in Example 5-22.

Example 5-22 createCluster help text

```
wsadmin>print AdminTask.help('createCluster')
WASX8006I: Detailed help for command: createCluster
Description: Creates a new application server cluster.
Target object:  None
Arguments:
  None
Steps:
  clusterConfig - Specifies the configuration of the new server cluster.
  replicationDomain - Specifies the configuration of a replication domain for
this cluster. Used for HTTP session data replication.
  convertServer - Specifies an existing server will be converted to be the
first member of cluster.
  eventServiceConfig - Specifies the event service configuration of the new
server cluster.
  promoteProxyServer - If a proxy server was specified for convertServer,
apply its proxy settings to the cluster.
```

Some steps are required for performing the intended task, while others are optional. When starting the command task in interactive mode, the steps are numbered with an optional marker prefixed to the number. A prefix of:

- ▶ The asterisk (*) character indicates a required step.
- ▶ A parentheses () indicates a step that is disabled.
- ▶ No denotation indicates an optional step.
- ▶ An arrow (→) indicates the current step in process.

New in V6.1: Several new high level commands have been added to **wsadmin** to ease administrative tasks. These commands are exposed in **wsadmin** as **AdminTask** commands. Table 5-2 shows these new commands.

Table 5-2 New high level commands

High level command group	Commands
Data Source Management	createJDBCProvider, createDataSource, listJDBCProviders, and listDatasources
Server Management	New commands to modify and view server configurations
Variable Configuration	setVariable, removeVariables, and showVariables
Port Management	listServerPorts, modifyServerPort, and listApplicationPorts
Report Generation Commands	ReportConfigInconsistencies, ReportConfiguredPorts

New in V6.1: Table 5-3 on page 278 shows some utility commands introduced in WebSphere Application Server V6.1.

Table 5-3 New utility commands

Utility	Function
isFederated	Check if the system is a single server or network deployment.
getDmgrProperties	Return the name of the deployment manager.
changeHostName	Change the host name of a node.
renameNode	This new command line utility is used to rename a federated node in a network deployment environment.

5.3 Common operational tasks using wsadmin

In this section, we describe how you can use **wsadmin** to perform common WebSphere operations. This section discusses a general approach for operational tasks and gives specific examples of common administrative tasks.

5.3.1 General approach for operational tasks

In order to invoke an operation on a running MBean, you first need to know the name of the running object. Then you invoke the method on a fully qualified object name. This means that invoking operations usually involves two types of commands:

- ▶ Find the object name.
- ▶ Invoke the operation.

In simple cases, the two commands can be combined.

Similarly, in order to change an attribute of a running object, you first need to know the object name of that running object. This means that getting or setting attributes involves a sequence of two commands:

- ▶ Find the object name of the running object/MBeans.
- ▶ Get or set attributes for that running object.

Note: You can use the **queryNames** and **completeObjectName** commands of the AdminControl object to identify the name of a running object. See “Help” on page 258 for information about how to do this.

5.3.2 Examples of common administrative tasks

Common operational tasks performed using `wsadmin` include:

- ▶ Starting and stopping the deployment manager
- ▶ Starting and stopping nodes
- ▶ Starting and stopping application servers
- ▶ Starting, stopping, and viewing enterprise applications
- ▶ Starting and stopping clusters
- ▶ Generating the Web server plug-in configuration file
- ▶ Enabling tracing for WebSphere components

Note: Some of the examples used in this section need Network Deployment installed. To show the command syntax, we used the WebSphere sample applications.

The elements of our distributed server environment include:

- ▶ Server node: `kcgg1f3Node01`
- ▶ Deployment manager node: `dmgr`
- ▶ Node agent server: `nodeagent`
- ▶ Servers: `server1`, `server2`

5.3.3 Managing the deployment manager

This section describes how to start and stop tasks on the deployment manager using `wsadmin`.

Starting the deployment manager

`wsadmin` works on MBeans. Because the MBean representing the deployment manager is not available unless the process is running, you have to start the deployment manager using other methods (see 4.3.2, “Starting and stopping the deployment manager” on page 166).

Stopping the deployment manager

The deployment manager can be stopped using the `AdminControl` object and invoking the `stopServer` command. To invoke `stopServer`, you must provide the deployment manager name and the node name. Example 5-23 shows an example of stopping the deployment manager.

Example 5-23 Stopping deployment manager using a single line command

```
wsadmin>AdminControl.stopServer('dmgr','kcgg1f3CellManager01')
WASX7337I: Invoked stop for server "dmgr" Waiting for stop completion.
WASX7264I: Stop completed for server "dmgr" on node "kcgg1f3CellManager01"
```

The stop operation can also be performed by invoking the stop method of the AdminControl object on the MBean representing the deployment manager. To do this, you need to identify the MBean that represents the deployment manager using the **completeObjectName** command of AdminControl object.

Example 5-24 shows the command to query the MBeans information and the command to stop the deployment manager. First, the variable named dmgr is assigned to the DeploymentManager Server MBean; subsequently, this variable is used for starting the **invoke** command.

Example 5-24 Getting MBean information and stopping the deployment manager

```
wsadmin>dmgr =  
AdminControl.completeObjectName('type=Server,processType=DeploymentManager,*')  
wsadmin>AdminControl.invoke(dmgr,'stop')
```

5.3.4 Managing nodes

This section describes how to perform common administration tasks on nodes and their node agent using **wsadmin**.

Starting a node agent

As with the deployment manager, the node agent cannot be started with **wsadmin** because there are no MBeans available yet. Use the **startNode** command to start the node agent. For information, see 4.5.5, “Starting and stopping nodes” on page 215.

Stopping a node agent

The node agent process controls all of the WebSphere managed processes on a node. Therefore, stopping a node agent limits the ability to issue any further commands against managed servers. There is one node agent per node.

You can stop node agents by invoking the **stopServer** command of the AdminControl object. The name of the node agent server and the name of the node need to be supplied as arguments. Example 5-25 shows the command to stop a node agent.

Example 5-25 Single line command to stop a node agent

```
wsadmin>AdminControl.stopServer('nodeagent','kcgg1f3Node01')  
WASX7337I: Invoked stop for server "nodeagent" Waiting for stop completion.  
WASX7264I: Stop completed for server "nodeagent" on node "kcgg1f3Node01"
```

The stop operation of the node agent can also be performed by invoking the stop operation on the MBean representing the node agent. You first need to identify the Server MBean for the node agent using the **completeObjectName** command.

Example 5-26 shows the command syntax to query MBean information for the node agent Server object and to invoke the stop method on the identified MBean.

Example 5-26 Getting MBean information for a node agent Server object

```
wsadmin>naServer =
AdminControl.completeObjectName('type=Server,node=kcgg1f3Node01,name=nodeagent,
*')
wsadmin>AdminControl.invoke(naServer,'stop')
```

5.3.5 Managing application servers

This section describes how to perform common administration tasks on application servers using **wsadmin**.

Starting an application server

In a distributed server environment, the node agent can start an application server. Example 5-27 shows the command for starting the server2 application server by use of the **startServer** command.

Example 5-27 Start an application server

```
wsadmin>AdminControl.startServer('server2','kcgg1f3Node01')
'WASX7262I: Start completed for server "server2" on node "kcgg1f3Node01"'
```

You can also use the **launchProcess** operation on the NodeAgent object to start server2. Example 5-28 on page 281 shows the command syntax to query the MBean information for the NodeAgent object and to invoke the **launchProcess** operation from the identified MBean.

Example 5-28 Getting MBean information for a node agent NodeAgent object

```
wsadmin>naMain =
AdminControl.completeObjectName('type=NodeAgent,node=kcgg1f3Node01,name=NodeAge
nt,*')
wsadmin>AdminControl.invoke(naMain,'launchProcess','server2')
'true'
```

Stopping an application server

Example 5-29 shows the command for stopping the server2 application server.

Example 5-29 Stop an application server

```
wsadmin>AdminControl.stopServer('server2','kcgg1f3Node01')
WASX7337I: Invoked stop for server "server2" Waiting for stop
completion.
WASX7264I: Stop completed for server "server2" on node "kcgg1f3Node01"
```

You can also use the **launchProcess** operation on the **NodeAgent** to start the application server. Example 5-30 on page 282 shows the command syntax to query the MBean information for the **NodeAgent** object and to invoke the **launchProcess** operation from the identified MBean.

Example 5-30 Getting MBean information for a node agent NodeAgent object

```
wsadmin>naMain = AdminControl.queryNames('*',type=NodeAgent')
wsadmin>AdminControl.invoke(naMain,'launchProcess','server2')
'true'
```

If there are multiple application servers running on a node, you can stop all the servers from a single script. Example 5-31 on page 282 shows a script that stops all application servers on the **SocratesNode** node. In this example, the node name is hardcoded, but it is also possible to write Jython code that accepts the node name from the command line or a menu.

To invoke the script from a command, type the following:

```
cd \<was_home>\profiles\<profile_name>\bin
wsadmin -f <script_filename>
```

Example 5-31 Stopping all application servers on a node

```
servername =
AdminControl.queryNames('node=kcgg1f3Node01,type=Server,processType=ManagedProc
ess,*').split(lineSeparator)
for item in servername:
    shortname = AdminControl.getAttribute(item,'name')
    completename =
AdminControl.completeObjectName('type=Server,node=kcgg1f3Node01,name='+shortnam
e+',*')
    print 'Stopping server : '+shortname
    AdminControl.invoke(completename, "stop")
#endFor
```

5.3.6 Managing enterprise applications

This section describes how to perform common administration tasks on enterprise applications.

New in V6.1

The application management functions of install, edit, and update provided by AdminApp have been simplified using the regular expression pattern instead of use of all parameters. In order to specify target for all Web modules, one can specify `*.war*` as module URI pattern in the MapModulesToServer step.

Install and update commands now support the `server` or `node` option to specify a default target for installation or updating. Multiple targets can be specified in a single command, avoiding the need to repeat a command for each target. A `+` or `-` leading delimiter is used on AdminApp install or edit operations to add or remove deployment targets. Lack of leading delimiter replaces existing targets with specified ones, which is the behavior in V6.0.x.

Viewing installed applications

Use the AdminApp object to view the applications installed on an application server. Example 5-32 shows the use of the list command and the resulting output.

Example 5-32 Listing installed applications

```
wsadmin>AdminApp.list()  
'DefaultApplication'
```

You can also do this by querying the MBeans for running applications on a node. Example 5-33 shows you how to perform this task.

Example 5-33 Listing applications by MBeans query

```
wsadmin>AdminControl.queryNames('type=Application,node=kcgg1f3Node01,*')  
'WebSphere:name=DefaultApplication,process=server1,platform=dynamicproxy,node=kcgg1f3Node01,J2EEName=DefaultApplication,Server=server1,version=6.1.0.0,type=Application,mbeanIdentifier=cells/kcgg1f3Cell01/applications/DefaultApplication.ear/deployments/DefaultApplication/deployment.xml#ApplicationDeployment_1153406359260,cell=kcgg1f3Cell01,spec=1.0'
```

If an object is not running, the MBean for that object does not exist. Based on this, we can write a simple Jython script that will display running applications.

Example 5-34 shows a script using the AdminApp object that lists the installed applications. The data obtained is configurational data and cannot be interrogated to determine run time status. Use **queryNames** for each installed application to see if an MBean exists, if the application is running. If the application is running, **queryNames** returns a name; otherwise, **queryNames** returns a null value.

Example 5-34 Script to display the status of applications

```
application = AdminApp.list().split(lineSeparator)
for app in application:
    objName = AdminControl.queryNames('type=Application,name='+ app +',*')
    if (len(objName) == 0):
        print 'The Application ' + app + ' is not running'
    else:
        print 'The Application ' + app + ' is running'
    #end if
#end for
```

Stopping a running application

To stop a running application, we use the AdminControl object and invoke the stopApplication method on the MBean of the running application. Example 5-35 shows the sequence of commands used to query the MBean and stop the application.

Example 5-35 Stopping a running application

```
wsadmin>appservername =
AdminControl.queryNames('type=ApplicationManager,node=kcgg1f3Node01,process=server1,*')
wsadmin>AdminControl.invoke(appservername,'stopApplication','DefaultApplication')
```

Starting a stopped application

To start a stopped application, we use the AdminControl object and invoke the startApplication method on the stopped application. This requires the identity of the application server MBean. Example 5-36 shows the sequence of commands used to start the DefaultApplication application.

Example 5-36 Starting a stopped application

```
wsadmin>appservername =
AdminControl.queryNames('type=ApplicationManager,node=kcgg1f3Node01,process=server1,*')
wsadmin>AdminControl.invoke(appservername,'startApplication','DefaultApplication')
```

5.3.7 Managing clusters

This section describes how to perform common administration tasks on clusters using **wsadmin**.

Starting a cluster

Example 5-37 shows the sequence of commands needed to start a cluster. The first command lists the configured clusters in the cell. In this case, there is only one cluster, `testCluster`. The second command initializes a variable named `tstClst` with the cluster object name. The final command invokes the start operation on the cluster object.

Example 5-37 Start a cluster

```
wsadmin>AdminControl.queryNames('type=Cluster,*')
'WebSphere:name=testCluster,process=dmgr,platform=common,node=kcgg1f3CellManage
r01,version=6.1.0.0,type=Cluster,mbeanIdentifier=testCluster,cell=kcgg1f3Cell01
,spec=1.0'

wsadmin>tstClst =
AdminControl.completeObjectName('type=Cluster,name=testCluster,*')
wsadmin>AdminControl.invoke(tstClst,'start')
```

Stopping a cluster

Example 5-38 shows the sequence of commands used to stop a cluster. The first command lists the configured clusters in the cell. The second command initializes a variable named `tstClst` with the cluster object name. The final command invokes the stop operation on the cluster object.

Example 5-38 Stopping a cluster

```
wsadmin>AdminControl.queryNames('type=Cluster,*')
'WebSphere:name=testCluster,process=dmgr,platform=common,node=kcgg1f3CellManage
r01,version=6.1.0.0,type=Cluster,mbeanIdentifier=testCluster,cell=kcgg1f3Cell01
,spec=1.0'

wsadmin>tstClst =
AdminControl.completeObjectName('type=Cluster,name=testCluster,*')

wsadmin>AdminControl.invoke(tstClst,'stop')
```

5.3.8 Generating the Web server plug-in configuration

Example 5-39 shows the sequence of commands used to generate the Web server plug-in configuration file. The first command identifies the MBean for the Web server plug-in configuration file generator on a node. The second command generates the Web server plug-in configuration file.

Example 5-39 Generating the Web server plug-in configuration file

```
wsadmin>pluginGen =  
AdminControl.completeObjectName('type=PluginCfgGenerator,*')  
wsadmin>AdminControl.invoke(pluginGen,'generate',"C:/PROGRAM~1/IBM/WebSphere/App  
Server/profiles/Dmgr01/config kcgg1f3Cell01 kcgg1f3CellManager01 dmgr  
plugin-cfg.xml")  
WASX7435W: Value plugin-cfg.xml is converted to a boolean value of false.
```

The argument for the generate command includes:

- ▶ Install root directory
- ▶ Configuration root directory
- ▶ Cell name
- ▶ Node name
- ▶ Server name
- ▶ Output file name

You can use `null` as an argument for the node and server name options. The generate operation generates a plug-in configuration for all the nodes and servers residing in the cell. The output file, `plugin-cfg.xml`, is created in the configuration root directory, in this case on `C:\Program Files\IBM\WebSphere\AppServer\profiles\Dmgr01\config\cells\kcgg1f3Cell01\nodes\kcgg1f3CellManager01\servers\dmgr`.

5.3.9 Enabling tracing for WebSphere components

This section illustrates how to enable tracing for a server process using the **setAttribute** command on the `TraceService` MBean.

In a Network Deployment environment, there are multiple server processes and therefore multiple `TraceService` MBeans. Example 5-40 shows how to use **queryNames** to list the `TraceService` MBeans.

Example 5-40 List of TraceService MBeans

```
wsadmin>print AdminControl.queryNames('type=TraceService,*')
WebSphere:name=TraceService,process=dmgr,platform=proxy,node=kcgg1f3CellManager
01,version=6.1.0.0,type=TraceService,mbeanIdentifier=cells/kcgg1f3Cell01/nodes/
kcgg1f3CellManager01/servers/dmgr/server.xml#TraceService_1,cell=kcgg1f3Cell01,
spec=1.0
WebSphere:name=TraceService,process=nodeagent,platform=proxy,node=kcgg1f3Node01
,version=6.1.0.0,type=TraceService,mbeanIdentifier=cells/kcgg1f3Cell01/nodes/kc
gg1f3Node01/servers/nodeagent/server.xml#TraceService_1120677326772,cell=kcgg1f
3Cell01,spec=1.0
WebSphere:name=TraceService,process=server2,platform=proxy,node=kcgg1f3Node01,v
ersion=6.1.0.0,type=TraceService,mbeanIdentifier=cells/kcgg1f3Cell01/nodes/kcgg
1f3Node01/servers/server2/server.xml#TraceService_1154007376682,cell=kcgg1f3Cel
l01,spec=1.0
```

To start tracing for a server, you need to locate the TraceService MBean for the server process using the **completeObject** command. Example 5-41 shows how to do this using a variable named **ts**, which is set to the value of the tracing service MBean. In the second step, the **setAttribute** command is used to enable the tracing.

Example 5-41 Enable tracing using TraceService mbean

```
wsadmin>ts =
AdminControl.completeObjectName('type=TraceService,process=server1,*')
wsadmin>AdminControl.setAttribute(ts,'traceSpecification','com.ibm.ejs.*=all')
```

The SystemOut.log file for the Server reflects this new trace specification, as the TraceService has logged this statement:

```
TRAS0018I: The trace state has changed. The new trace state is
*=info:com.ibm.ejs.*=all
```

Note that setting trace level with use of the AdminControl object only changes the current trace specification of the TraceService. The specification is not stored to the WebSphere configuration repository. To change the configuration permanently, use the **modify** command of the AdminConfig object to change the traceSpecification attribute of the TraceService configuration object.

5.4 Common configuration tasks

In this section, we describe how to use **wsadmin** to create, modify, and change the WebSphere Application Server configuration. The section is described in two parts as follows:

- ▶ General approach for configuration tasks
- ▶ Specific examples of WebSphere configuration tasks

5.4.1 General approach for configuration tasks

There are many possible configuration tasks that can be performed in a WebSphere environment. Rather than document every possible modification, we describe a general approach to use when performing configuration tasks and then give a few specific examples.

This general approach has three steps:

1. Find the object you want to change using **AdminConfig.getId()**.
2. Change or create a configuration using **AdminConfig.modify()** or **create()**.
3. Save the changes using **AdminConfig.save()**.

The **create** and **modify** commands use an attribute list. In general, the attribute is supplied as a list of Jython lists. A Jython list can be constructed using name and value pairs as follows:

```
[ ['name1', 'value1'], ['name2', 'value2'], ['name3', 'value3']... ]
```

The attributes for a WebSphere configuration object are often deeply nested. If you need to modify a nested attribute, you can get the ID of the object and modify it directly. This is the preferred method, although it requires more lines of scripting.

5.4.2 Specific examples of WebSphere configuration tasks

This section describes how a variety of typical configuration tasks can be done using the **wsadmin** objects, including:

- ▶ Application server
 - Create or remove an application server.
- ▶ Enterprise application
 - Install or uninstall an enterprise application.
 - Change attributes of an enterprise application.
- ▶ Configure and modify WebSphere configuration

- Configure virtual hosts.
- Configure JDBC providers.
- Edit an application server.
- Create a cluster.
- Add member to a cluster.

Creating an application server

With the introduction of the AdminTask object, there are now two ways of creating an application server. The AdminTask provides the interactive approach, and is shown in Example 5-42. Notice the batch invocation of the **createApplicationServer** command shown at the end of the input.

Notice the extra step after collecting the configuration values for the server creation. This extra step provides the ability to configure ConfigCoreGroup options for the server being created. The → arrow in front of the line indicates this to be the current step of the interactive guide. To input a core group name for this server, type S (for select), then press Enter. To skip configuration of a core group for this server, type F (as shown).

Example 5-42 Creating an application server using AdminTask

```
wsadmin>AdminTask.createApplicationServer('-interactive')
Create Server
```

Command that creates a server

```
*Node Name: kcgglf3Node01
*Server Name (name): server4
Template Name (templateName):
Generate Unique Ports (genUniquePorts): [true]
template location (templateLocation):
server specific short name (specificShortName):
server generic short name (genericShortName):
Create Server
```

Command that creates a server

```
-> 1. No description available (ConfigCoreGroup)
```

```
S (Select)
F (Finish)
C (Cancel)
H (Help)
```

```
Select [S, F, C, H]: [F] F
```

```
WASX7278I: Generated command line:
AdminTask.createApplicationServer('kcgg1f3Node01', '[-name server4 ]')
'server4(cells/kcgg1f3Cell01/nodes/kcgg1f3Node01/servers/server4|server.xml#Server_1154109581505)'
wsadmin>AdminConfig.save()
```

The alternative approach to using AdminTask for creating an application server is using the AdminConfig object. Example 5-43 illustrates application server creation using AdminConfig. The first command initializes a variable named node to set the value of the node configuration ID. The second command creates the server on the node.

Example 5-43 Creating an application server using AdminConfig

```
wsadmin>AdminConfig.getId('/Node:kcgg1f3Node01/')
'kcgg1f3Node01(cells/kcgg1f3Cell01/nodes/kcgg1f3Node01|node.xml#Node_1)'

wsadmin>node = AdminConfig.getId('/Node:kcgg1f3Node01/')
wsadmin>AdminConfig.create("Server", node, [{"name", "server5"}])
'server5(cells/kcgg1f3Cell01/nodes/kcgg1f3Node01/servers/server5|server.xml#Server_1154115062206)'
wsadmin>AdminConfig.save()
```

Removing an application server

As with creating application servers, an application server can be removed by either using the AdminTask object or the AdminConfig object. Example 5-44 illustrates removing an application server using AdminTask.

Example 5-44 Remove an application server using AdminTask

```
wsadmin>AdminTask.deleteServer('-interactive')
Delete Server

Delete a server configuration

*ADMG0106I (serverName): server5
*ADMG0104I (nodeName): kcgg1f3Node01
Delete Server

Delete a server configuration

-> 1. No description available (ConfigCoreGroup)
   2. No description available (CleanupSIBuses)

S (Select)
N (Next)
F (Finish)
```

C (Cancel)
H (Help)

```
Select [S, N, F, C, H]: [F] F
WASX7278I: Generated command line: AdminTask.deleteServer('[-serverName server5
-nodeName kcgglf3Node01 ]')
wsadmin>AdminConfig.save()
```

The general syntax for removing an application server using the AdminConfig object is:

```
AdminConfig.remove('<server Config id>')
```

Installing an enterprise application

There are two options for installing an application:

- Perform an interactive installation using the **installInteractive** command. The interactive install prompts you for options. The syntax is:

```
AdminApp.installInteractive('<ear_file_location>')
```

For example:

```
wsadmin>AdminApp.installInteractive('C:/PROGRA~1/IBM/WebSphere/AppServer/in
stallableApps/ivtApp.ear')
```

Note: In Windows, use either a forward slash (/), or double backslashes (\\) when specifying the path to the .ear file.

- Perform a non-interactive installation using the **install** command.

Using the install command

The general syntax for installing an enterprise application is as follows:

```
AdminApp.install('<location of the ear file> {task or non-task option}')
```

There are two types of options that can be specified when using the **install** command:

- To see a list of install task options, use the following syntax:

```
AdminApp.options()
```

The list includes options for specifying server name, cluster name, install directory, and so on.

- To see a list of application-specific options, use the following syntax:

```
AdminApp.options('<ear_file_location>')
```

Here is a sample output for application-specific options:

```
wsadmin>print
AdminApp.options('C:/PROGRA~1/IBM/WebSphere/AppServer/installableApps/ivtAp
p.ear')
```

The list of options includes things that define application information, security role mapping, module-to-virtual host mapping, and whether to pre-compile JSPs.

Note: All options supplied for the `install` command must be supplied in a single string. In Jython, a single string is formed by collecting all options within curly braces or double quotes:

```
AdminApp.install("c:/temp/application.ear", [:"-server", "serv2",
"-appname", "-TestApp"]])
```

Example 5-45 shows an example of installing a new application named `ivtApp` on a server named `server1` inside cluster `testCluster`.

Example 5-45 Installing an application

```
wsadmin>AdminApp.install('C:/PROGRA~1/IBM/WebSphere/AppServer/installableApps/i
vtApp.ear', [:"-server", 'server1', '-node', 'kcgg1f3Node01', '-cluster', 'testCluster
', '-appname', 'IVT App'])
....
wsadmin>AdminConfig.save()
```

Uninstalling an enterprise application

The general syntax for uninstalling an enterprise application is:

```
AdminApp.uninstall('<application name>')
```

Example 5-46 shows an example of uninstalling an application, remember that the application name is case sensitive.

Example 5-46 Uninstalling an enterprise application

```
wsadmin>AdminApp.uninstall('IVT App')
ADMA5017I: Uninstallation of IVT App started.
ADMA5104I: The server index entry for
WebSphere:cell=kcgg1f3Cell01,node=kcgg1f3N
ode01 is updated successfully.
ADMA5102I: The configuration data for IVT App from the configuration repository
is deleted successfully.
ADMA5011I: The cleanup of the temp directory for application IVT App is
complete.
ADMA5106I: Application IVT App uninstalled successfully.
wsadmin>AdminConfig.save()
```


Editing an enterprise application

Editing of an enterprise application can be done either interactively or non-interactively. The following commands are available for editing:

- Interactively, use the **editInteractive** command, which prompts you for input. The syntax is:

```
AdminApp.editInteractive('<application name>')
```

- Non-interactively, you can use the **edit** command.

Using the edit command

The general syntax for editing an enterprise application in non-interactive mode is:

```
AdminApp.edit(<application_name>, [-taskname [['item1a',  
'item2a','item3a']] ['item1b','item2b','item3b'].....])
```

In Example 5-47, you can see how to change the module to server mapping for an application. The options are the same as those you would use during installation with the **install** command.

Example 5-47 Edit an enterprise application

```
wsadmin>AdminApp.edit("IVT App", ["-MapModulesToServers", [{"IVT Application",  
"ivt_app.war,WEB-INF/web.xml","WebSphere:cell=kcgg1f3Cell01,node=kcgg1f3Node01,  
server=server1,cluster=testCluster"}]] )  
wsadmin>AdminConfig.save()
```

Preventing the startup of an application

To prevent the startup of a specific enterprise application when starting the application server, change the configuration property to enable the enterprise application on the deployed target. In Example 5-48, the steps to locate, modify, and save the target property are outlined.

Example 5-48 Disable of enterprise application on target server

```
wsadmin>import java.lang.System as sys  
wsadmin>lineSeparator = sys.getProperty('line.separator')  
wsadmin>eaBk =  
AdminConfig.list('ApplicationDeployment').split(lineSeparator)[0]  
wsadmin>print AdminConfig.showAttribute(eaBk,'targetMappings')  
["(cells/kcgg1f3Cell01/applications/IVT App.ear/deployments/IVT  
App|deployment.xml#DeploymentTargetMapping_1154118924159)"]  
wsadmin>AdminConfig.modify('(cells/kcgg1f3Cell01/applications/IVT  
App.ear/deployments/IVTApp|deployment.xml#DeploymentTargetMapping_11541  
18924159)',[['enable','false']])  
wsadmin>print AdminConfig.queryChanges()
```

```
'WASX7146I: The following configuration files contain unsaved changes:
cells/kcgg1f3Cell101/applications/IVT App.ear/deployments/IVT
App/deployment.xml '
wsadmin>AdminConfig.save()
```

Creating a virtual host

The command to create a virtual host is:

```
AdminConfig.create('VirtualHost', <cell object>, [['name', '<vhost name>']])
```

First, you need to find the ID of the object you want to change. The virtual host is a WebSphere resource defined in a cell. Therefore, by creating a virtual host, we are modifying the configuration of the cell object. Example 5-49 shows the command syntax for retrieving the configuration ID of the cell object and creating the virtual host resource. Finally, save the changes to the WebSphere configuration repository.

Example 5-49 Find an object using the AdminConfig command

```
wsadmin>cell = AdminConfig.getid('/Cell:kcgg1f3Cell101/')
wsadmin>AdminConfig.create('VirtualHost', cell, [['name', 'IVTVHost']])
'IVTVHost(cells/kcgg1f3Cell101|virtualhosts.xml#VirtualHost_1154362727831)'
wsadmin>AdminConfig.save()
```

Modifying a virtual host

Modify the virtual host configuration with the **modify** command in the AdminConfig object. Example 5-50 shows an example of modifying a virtual host. The example gets the ID of the IVTVHost virtual host, then uses that ID in the **modify** command to redefine the list of aliases.

Example 5-50 Modifying a virtual host

```
wsadmin>IVTVHost = AdminConfig.getid('/VirtualHost:IVTVHost/')
wsadmin>AdminConfig.modify(IVTVHost, [['aliases', [['hostname', '*'], ['port",
9082]], [['hostname', '*'], ['port", 80]]]])
wsadmin>print AdminConfig.queryChanges()
WASX7146I: The following configuration files contain unsaved changes:
  cells/kcgg1f3Cell101/virtualhosts.xml
wsadmin>AdminConfig.save()
```

Modifying an application server

Modify an application server configuration using the AdminConfig object. The **modify** command is used for changing the attribute values for configuration objects. As the AdminConfig commands interacts with the configuration service, changes are written to the WebSphere configuration repository (XML

documents). All services within the WebSphere run time environment read from the configuration repository at startup only. As a result, changes made with the AdminConfig commands take effect only after restarting the service or WebSphere run time.

Tip: To find the parent-child relationships for configuration objects placed in the application server configuration hierarchy, use the output from the **showall** command. To use **showall**, use the following syntax:

```
AdminConfig.showall(<object id of application server>)
```

Also, the layout of the WebSphere administrative console presents some kind of logical progression from parent to child. For example, to change the PingInterval you would need to select **Application Server** → **<server_name>** → **Monitoring Policy** → **Ping Interval**.

Example 5-51 shows an example of changing the ping interval for a server named server1.

Example 5-51 Modifying an application server

```
wsadmin>AdminControl.stopServer('server1','kcgg1f3Node01')
WASX7337I: Invoked stop for server "server1" Waiting for stop completion.
'WASX7264I: Stop completed for server "server1" on node "kcgg1f3Node01"'

wsadmin>srv = AdminConfig.getid('/Node:kcgg1f3Node01/Server:server1/')

wsadmin>prcDef = AdminConfig.list('ProcessDef',srv)

wsadmin>monPol = AdminConfig.list('MonitoringPolicy',prcDef)

wsadmin>AdminConfig.modify(monPol, [{"pingInterval", 120}] )

wsadmin>AdminConfig.save()

wsadmin>AdminControl.startServer('server1','kcgg1f3Node01')
'WASX7262I: Start completed for server "server1" on node "kcgg1f3Node01"'
```

Creating a cluster

To create a new cluster use either the AdminTask or AdminConfig object. In Example 5-52, the AdminTask object is used for creating a cluster named T4SCluster adding an existing server named server2 as a cluster member.

Example 5-52 Create a server cluster

```
wsadmin>AdminTask.createCluster('-interactive')
Create Server Cluster
```

Creates a new application server cluster.

```
-> *1. Cluster Configuration (clusterConfig)
    2. Replication Domain (replicationDomain)
    3. Convert Server (convertServer)
    4. Configure the event service during cluster creation.
    (eventServiceConfig)

    5. Promote Proxy Server Settings To Cluster (promoteProxyServer)

S (Select)
N (Next)
C (Cancel)
H (Help)
```

```
Select [S, N, C, H]: [S] S
```

```
Cluster Configuration (clusterConfig)
```

```
*Cluster Name (clusterName):
Prefer Local (preferLocal): [true]
Cluster Type (clusterType):
Short Name of Cluster (shortName):
```

```
Select [C (Cancel), E (Edit)]: [E]
*Cluster Name (clusterName): testCluster2
Prefer Local (preferLocal): [true]
Cluster Type (clusterType):
Short Name of Cluster (shortName):
Create Server Cluster
```

Creates a new application server cluster.

```
1. Cluster Configuration (clusterConfig)
-> 2. Replication Domain (replicationDomain)
    3. Convert Server (convertServer)
```

4. Configure the event service during cluster creation.
(eventServiceConfig)

5. Promote Proxy Server Settings To Cluster (promoteProxyServer)

S (Select)
N (Next)
P (Previous)
F (Finish)
C (Cancel)
H (Help)

Select [S, N, P, F, C, H]: [F] N
Create Server Cluster

Creates a new application server cluster.

1. Cluster Configuration (clusterConfig)
2. Replication Domain (replicationDomain)
-> 3. Convert Server (convertServer)
4. Configure the event service during cluster creation.
(eventServiceConfig)

5. Promote Proxy Server Settings To Cluster (promoteProxyServer)

S (Select)
N (Next)
P (Previous)
F (Finish)
C (Cancel)
H (Help)

Select [S, N, P, F, C, H]: [F] S

Convert Server (convertServer)

Converted Server Node Name (serverNode):
Converted Server Name (serverName):
Member Weight (memberWeight):
Node Group (nodeGroup):
enable data replication (replicatorEntry):

Select [C (Cancel), E (Edit)]: [E] E
Converted Server Node Name (serverNode): kcgglf3Node01
Converted Server Name (serverName): server2
Member Weight (memberWeight):

```
Node Group (nodeGroup):
enable data replication (replicatorEntry):
Create Server Cluster
```

Creates a new application server cluster.

```
1. Cluster Configuration (clusterConfig)
2. Replication Domain (replicationDomain)
3. Convert Server (convertServer)
-> 4. Configure the event service during cluster creation.
(eventServiceConfig)

5. Promote Proxy Server Settings To Cluster (promoteProxyServer)
```

```
S (Select)
N (Next)
P (Previous)
F (Finish)
C (Cancel)
H (Help)
```

```
Select [S, N, P, F, C, H]: [F] F
WASX7278I: Generated command line: AdminTask.createCluster('[-clusterConfig
[-clusterName testCluster2] -convertServer [-serverNode kcgg1f3Node01
-serverName server2]]')
'testCluster2(cells/kcgg1f3Cell101/clusters/testCluster2|cluster.xml#ServerClust
er_1154374243841)'
wsadmin>AdminConfig.save()
```

The **AdminConfig** object provides a different means of creating a cluster. Use the **convertToCluster** command to create a cluster with an existing server added. Use the **create** command to create an empty cluster with the **ServerCluster** type object.

Adding a member to an existing cluster

As with creating a cluster, both **AdminTask** and **AdminConfig** objects provide the means for creating a new cluster members. Servers have to be created as cluster members from the start; they cannot be joined to a cluster later.

Example 5-53 shows how to create a new server, **server4**, and make it a member of a cluster, **testCluster2**, by use of the batch invocation of the **createClusterMember** command from the **AdminTask**.

Example 5-53 Create a new cluster member

```
wsadmin>AdminTask.createClusterMember(["-clusterName", "testCluster2",  
"-memberConfig", [{"kcgg1f3Node01", "server4", "", "", "true", "false"}]] )  
'server4(cells/kcgg1f3Cell01/clusters/testCluster2|cluster.xml#ClusterMember_11  
54375381547)'  
wsadmin>AdminConfig.save()
```

Deleting a member from a cluster

To delete a member from a cluster, use the AdminTask `deleteClusterMember` command. Example 5-54 shows how to delete a cluster member.

Example 5-54 Delete a cluster member

```
wsadmin>AdminTask.deleteClusterMember( ["-clusterName", "testCluster2",  
"-memberNode", "kcgg1f3Node01", "-memberName", "server4"] )  
'ADMG9239I: Cluster member server4 on node kcgg1f3Node01 deleted from cluster  
testCluster2.'  
wsadmin>AdminConfig.save()
```

Configuring JDBC providers

Example 5-55 on page 300 shows a common method for creating a JDBC provider. The provider is created based on a template.

Using templates: A group of templates are supplied with the WebSphere installation as XML files in the `<profile_home>/config/templates` directory. Within each XML file, you will find multiple entries. To use a template, you specify the XML file and the entry within the file that you want to use.

Templates are especially useful when using the AdminConfig object for configuration purposes. The template reduces the amount of typed input required, speeding up the process and reducing the probability of syntax errors.

The `listTemplates` command of the AdminConfig object prints a list of templates matching a given type. These templates can be used with the `createUsingTemplate` command.

In Example 5-55, the JDBC provider is added at the cluster scope, so the first command gets the configuration ID for the cluster and assigns it to a variable named `cluster` to hold the ID. The second command uses `listTemplates` to set the `JDBCtempl` variable to the template ID. The third command creates the JDBC provider using the template.

Example 5-55 Configuring a JDBC driver

```
wsadmin>cluster = AdminConfig.getid('/ServerCluster:testCluster/')
wsadmin>JDBCtempl = AdminConfig.listTemplates("JDBCProvider", "Cloudscape JDBC
Provider (XA)" ).split(lineSeparator)[1]
wsadmin>AdminConfig.createUsingTemplate("JDBCProvider", cluster, [{"name",
"testDriver"}], JDBCtempl )
'testDriver(cells/kcgg1f3Cell01/clusters/testCluster|resources.xml#JDBCProvider
_1154378721689)'
wsadmin>AdminConfig.save()
```

5.5 Help creating wsadmin scripts

Assistance with scripting (new): V6.1 has added the command assistance feature in the administrative console to show the corresponding scripting commands when you perform certain activities. The list of activities that show the corresponding commands will grow over time. You also have the option to send these as notifications to the Application Server Toolkit, where you can use the new Jython editor to build scripts.

Command assistance in the administrative console maps your administrative activities to `wsadmin` scripting commands, so that you can capture your console activities and apply them to `wsadmin` scripts. To enable this feature, go to **Console Preferences** and check the box **Enable command assistance notifications**. After performing an administrative task, the help portlet will show the corresponding command in Jython. You can also enable an option to log command assistance commands.

The new Jython editor in Application Server ToolKit V6.1 is used to perform a variety of tasks, such as the following:

- ▶ Develop Jython script files.
- ▶ Edit Jython script files.
- ▶ Import existing Jython files for structured viewing.
- ▶ Set breakpoints for debugging your scripts.

The Jython Editor is also integrated with the WebSphere Administrative Script Launcher and Debugger tools, so you can run and debug script files directly from the editor.

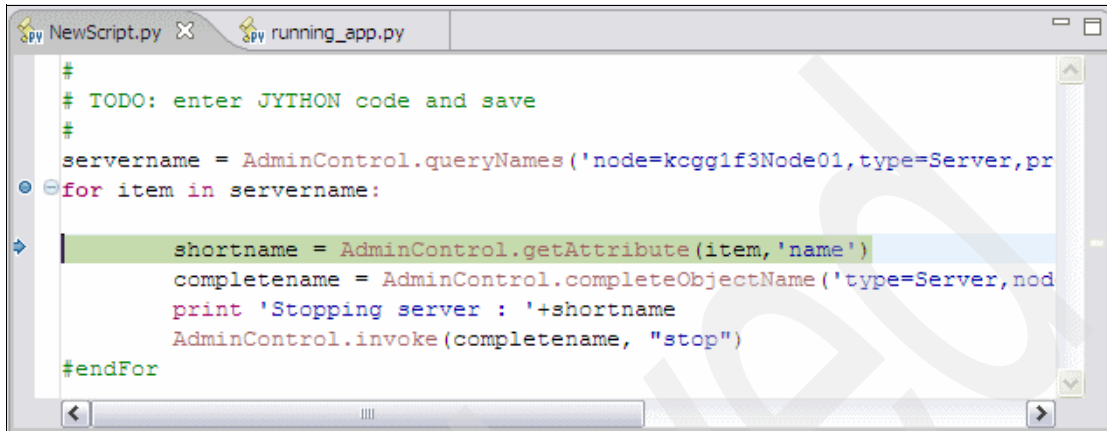


Figure 5-2 Jython editor running on debug mode.

The Jython editor has many text editing features, such as syntax highlighting, unlimited undo or redo, and automatic tab indentation. When you tag a comment in a Jython script with "#TODO", the editor automatically creates a corresponding task as a reminder in the Tasks view. Then, if you open the task later, the editor automatically synchronizes to that TODO entry in the script source. Other helpful features are content assist and tips, which provides a list of acceptable continuations depending on where the cursor is located in a Jython script file, or what you have just typed. The Jython editor is not integrated to a compiler. As a result, the Jython editor does not perform syntax verification on your scripts.

5.6 Using Java for administration

An alternative way of managing the WebSphere environment from a programmatic point of view is to develop a Java client that attaches to the WebSphere JMX infrastructure directly. Every administrative task can be performed with the use of MBean resources, just as the administrative console and `wsadmin` administrative objects use MBeans to do their tasks. The advantage of using Java for developing the administrative client is that the language is well-adopted in the WebSphere community. Every administrative aspect can be highly-customized. The disadvantage is that the developer needs to have a very detailed understanding of the WebSphere infrastructure and every administrative

task has to be built directly from the MBean resources. This means that **wsadmin** object functionality has to be programmed by the developer.

The Information Center has more on this topic. Also, the *IBM WebSphere Developer Technical Journal* article series *System Administration for WebSphere Application Server V5* discussed this subject in detail.

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ WebSphere Application Server Information Center
<http://www.ibm.com/software/webservers/appserv/was/library/>
See *Scripting: Resources for learning*
- ▶ *MBeanInspector for WebSphere Application Server*.
<http://www.alphaworks.ibm.com/tech/mbeaninspector>
- ▶ *Sample Scripts for WebSphere Application Server Versions 5 and 6*:
<http://www.ibm.com/developerworks/websphere/library/samples/SampleScripts.html>
- ▶ Tcl Developer Xchange
<http://www.tcl.tk/>
- ▶ IBM WebSphere Developer Technical Journal
<http://www.ibm.com/developerworks/websphere/techjournal/>

Configuring WebSphere resources

Resource providers are a class of objects that provide resources needed by running Java applications, and J2EE applications in particular. For example, if an application requires database access through a data source, you would need to install a JDBC data source provider and then configure a data source to be used by your application.

This chapter discusses the following application server resource providers:

- ▶ JDBC resources
- ▶ JCA resources
- ▶ JavaMail resources
- ▶ URL providers
- ▶ Resource environment providers
- ▶ Resource authentication

6.1 WebSphere resources

WebSphere Application Server provides a number of resources that you can define for applications to use. The resource types can be seen in the administrative console under the Resources category, as in Figure 6-1.

New in V6.1: The path to access resources from the administrative console has been shortened in some cases. For example, you can list the data sources without selecting a JDBC provider first. A new option for scope (All scopes) allows you to display all of the selected resource types as opposed to only those defined at a specific scope.

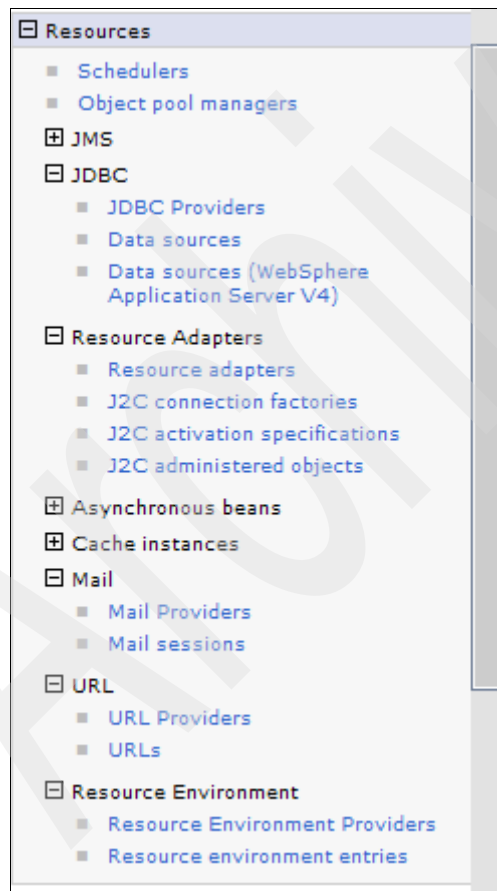


Figure 6-1 WebSphere Application Server resource types

In this chapter, we discuss the following topics:

- ▶ JDBC resources
- ▶ Resource adapters
- ▶ Mail providers
- ▶ URL providers
- ▶ Resource environment providers

For information about configuring JMS resources, see Chapter 8, “Asynchronous messaging” on page 399.

For information about dynamic cache, including servlet cache, and object cache configuration, see *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392.

Asynchronous beans, object pools, and schedulers are programming model extensions that have previously been available only in WebSphere Application Server Enterprise and in WebSphere Business Integration Server Foundation. These programming model extensions are not covered in this IBM Redbook. Information about them can be found in the Information Center. Conceptual information and examples of these at the previous versions can be found in:

- ▶ *WebSphere Application Server Enterprise V5 and Programming Model Extensions*, SG24-6932
- ▶ *WebSphere Business Integration Server Foundation V5.1 Handbook*, SG24-6318

6.2 JDBC resources

The JDBC API provides a programming interface for data access of relational databases from the Java programming language. The JDBC 3.0 API is comprised of two packages:

- ▶ The `java.sql` package (the JDBC 3.0 core API)
- ▶ The `javax.sql` package (the JDBC 3.0 Standard Extension API)

This package provides data source and connection pooling functionality.

In the next sections, we explain how to create and configure data source objects for use by JDBC applications. This is the recommended way of getting a connection to a database, and the only way if you are looking to use connection pooling and distributed transactions.

The following database platforms are supported for JDBC:

- ▶ DB2 family
- ▶ Oracle
- ▶ Sybase
- ▶ Informix
- ▶ SQL Server
- ▶ Cloudscape / Derby (test and development only)
- ▶ Third-party vendor JDBC data source using SQL99 standards

New in V6.1: The new embedded Cloudscape v10.1 is a pure Java database server. The code base, which the open source community calls Derby, is a product of the Apache Software Foundation (ASF) open source relational database project. The new Cloudscape includes Derby without any modification to the underlying source code. Learn more about Derby code at the Apache Derby Web site:

<http://db.apache.org/derby/>

WebSphere Application Server does not currently support Cloudscape v10.1 for production.

6.2.1 What are JDBC providers and data sources?

A data source represents a real-world data source, such as a relational database. When a data source object has been registered with a JNDI naming service, an application can retrieve it from the naming service and use it to make a connection to the data source it represents.

Information about the data source and how to locate it, such as its name, the server on which it resides, its port number, and so on, is stored in the form of properties on the `DataSource` object. This makes an application more portable because it does not need to hardcode a driver name, which often includes the name of a particular vendor. It also makes maintaining the code easier because if, for example, the data source is moved to a different server, all that needs to be done is to update the relevant property in the data source. None of the code using that data source needs to be touched.

Once a data source has been registered with an application server's JNDI name space, application programmers can use it to make a connection to the data source it represents.

The connection will usually be a pooled connection. In other words, once the application closes the connection, the connection is returned to a connection pool, rather than being destroyed.

Data source classes and JDBC drivers are implemented by the data source vendor. By configuring a JDBC provider, we are providing information about the set of classes used to implement the data source and the database driver. We are providing the environment settings for the DataSource object. A driver can be written purely in the Java programming language or in a mixture of the Java programming language and the Java Native Interface (JNI) native methods.

In the next sections, we describe how to create and configure data source objects, as well as how to configure the connection pools used to serve connections from the data source.

6.2.2 WebSphere support for data sources

The programming model for accessing a data source is as follows:

1. An application retrieves a DataSource object from the JNDI naming space.
2. After the DataSource object is obtained, the application code calls getConnection() on the data source to get a Connection object. The connection is obtained from a pool of connections.
3. Once the connection is acquired, the application sends SQL queries or updates to the database.

In addition to the data source support for J2EE 1.3 and J2EE 1.4 applications, support is also provided for J2EE 1.2 data sources. The two types of support differ in how connections are handled. However, from an application point of view, they look the same.

Data source support

The primary data source support is intended for J2EE 1.3 and J2EE 1.4 applications. Connection pooling is provided by two components, a JCA Connection Manager, and a relational resource adapter. See Figure 6-2.

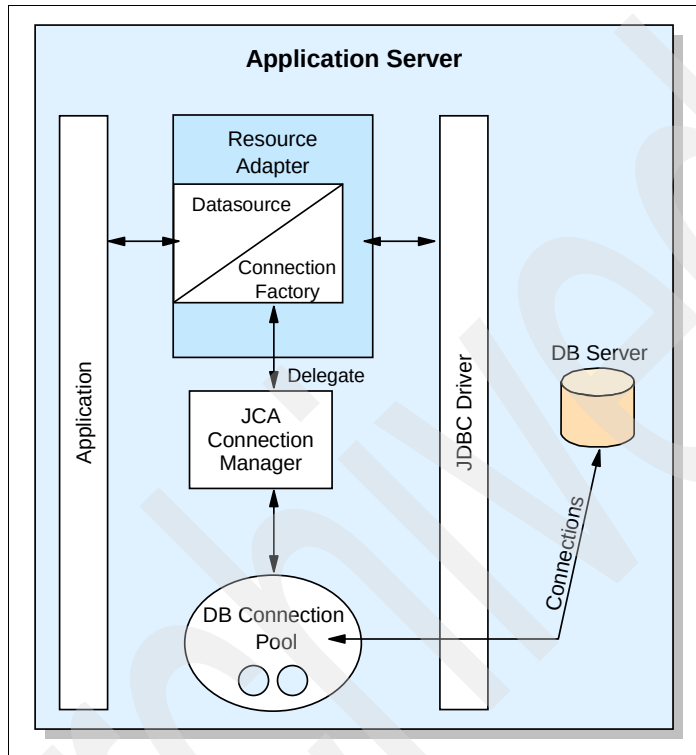


Figure 6-2 Resource adapter in J2EE connector architecture

The JCA Connection Manager provides connection pooling, local transaction, and security support.

The relational resource adapter provides JDBC wrappers and the JCA CCI implementation that allows BMP, JDBC applications, and CMP beans to access the database.

Figure 6-3 on page 309 shows the relational resource adapter model.

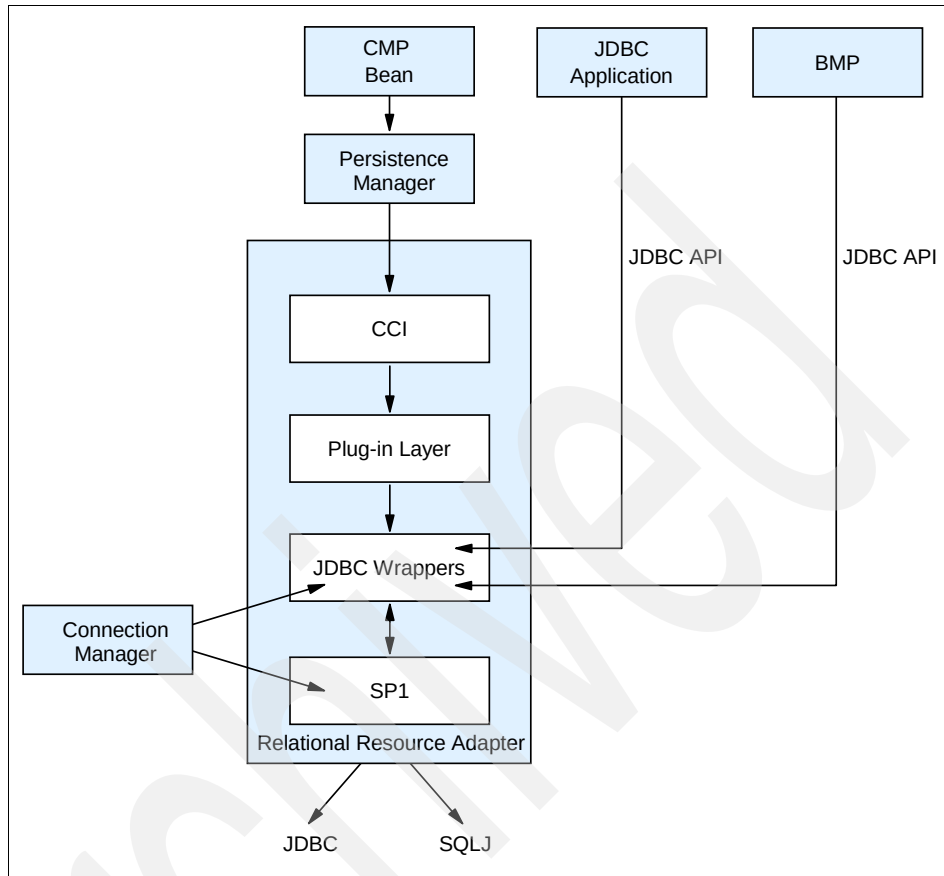


Figure 6-3 Persistence resource adapter model

WebSphere Application Server has a Persistence Resource Adapter that provides relational persistence services to EJB beans as well as providing database access to BMP and JDBC applications. The Persistence Resource Adapter has two components: the Persistence Manager, which supports the EJB CMP persistence model, and the Relational Resource Adapter. The Persistence Resource Adapter code is included in the following Java packages:

- ▶ `com.ibm.ws.rsadapter.cci` contains CCI implementation and JDBC wrappers.
- ▶ `com.ibm.ws.rsadapter.spi` contains SPI implementation.
- ▶ `com.ibm.ws.rsadapter.jdbc` contains all the JDBC wrappers.
- ▶ `com.ibm.websphere.rsadapter` `DataStoreHelper`, `WSCallerHelper` and `DataAccessFunctionSet`.

The Relational Resource Adapter is the Persistence Manager's vehicle to handle data access to and from the back-end store, providing relational persistence services to EJB beans. The implementation is based on the J2EE Connector (JCA) specification and implements the JCA CCI and SPI interfaces.

When an application uses a data source, the data source uses the JCA connector architecture to get to the relational database.

For an EJB, the sequence is as follows:

1. An EJB performs a JNDI lookup of a data source connection factory and issues a `getConnection()` request.
2. The connection factory delegates the request to a connection manager.
3. The connection manager looks for an instance of a connection pool in the application server. If no connection pool is available, then the manager uses the `ManagedConnectionFactory` to create a physical, or nonpooled, connection.

Version 4 data source

WebSphere Application Server V4 provided its own JDBC connection manager to handle connection pooling and JDBC access. This support is included with WebSphere Application Server V6 to provide support for J2EE 1.2 applications. If an application chooses to use a Version 4 data source, the application will have the same connection behavior as in Version 4 of the application server.

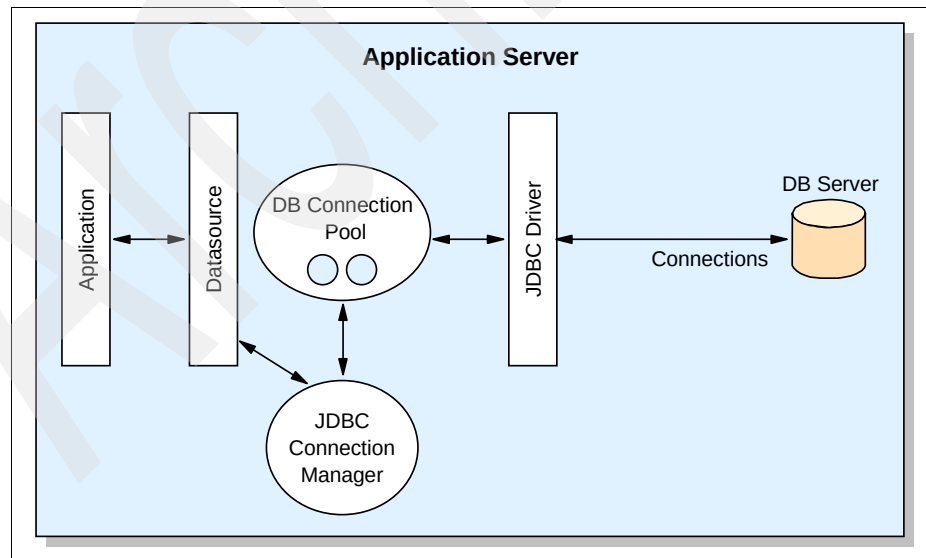


Figure 6-4 Connection pooling in WebSphere Application Server Version 4

Use the Version 4 data source for the following:

- ▶ J2EE 1.2 applications
All EJB beans, JDBC applications, or Version 2.2 servlets must use the Version 4 data source.
- ▶ EJB 1.x modules with 1.1 deployment descriptor
All of these must use the Version 4 data source.

6.2.3 Creating a data source

The following steps are involved in creating a data source:

1. Create a JDBC provider.
The JDBC provider gives the classpath of the data source implementation class and the supporting classes for database connectivity. This is vendor-specific.
2. Create a data source.
The JDBC data source encapsulates the database-specific connection settings.

6.2.4 Creating a JDBC provider

To create a JDBC provider, complete the following steps from the administrative console:

1. Expand **Resources** from the navigation tree.
2. Click **JDBC Providers**.
3. Select the scope. (Although you can select All scopes to view all resources, you must select a specific scope to create a resource.)

Note: JDBC resources are created at a specific scope level. The data source scope level is inherited from the JDBC provider. For example, if we create a JDBC provider at the node level and then create a data source using that JDBC provider, the data source will inherit:

- ▶ The JDBC provider settings, such as classpath, implementation class, and so on
- ▶ The JDBC provider scope level

In this example, if the scope were set to node-level, all application servers running on that node would register the data source in their name space.

The resources.xml file will also get updated at the node and application server level.

The administrative console now shows all the JDBC providers created at that scope level. In Figure 6-5 on page 313, you can see that in this case there is one JDBC provider defined at the server level.

4. Select **New** to create a new JDBC provider.

JDBC providers

JDBC providers

Use this page to edit properties of a JDBC provider. The JDBC provider object encapsulates the specific JDBC driver implementation class for access to the specific vendor database of your environment. Learn more about this task in a [guided activity](#). A guided activity provides a list of task steps and more general information about the topic.

⊕ Scope: Cell=**was61CellDmgrPrf1**, Node=**NodeAIXST02**, Server=**samplemember1**

⊖ Preferences

Maximum rows

☐ Retain filter criteria.

☒ Show built-in resources

Apply

Reset

NewDelete

Select	Name ↕	Scope ↕	Description ↕
☐	Derby JDBC Provider (XA)	Node=NodeAIXST02,Server=samplemember1	Built-in Derby JDBC Provider (XA)

Total 1

Figure 6-5 JDBC providers

Chapter 6. Configuring WebSphere resources 313

5. Use the list boxes to select the type of provider you want to create. See Figure 6-6.

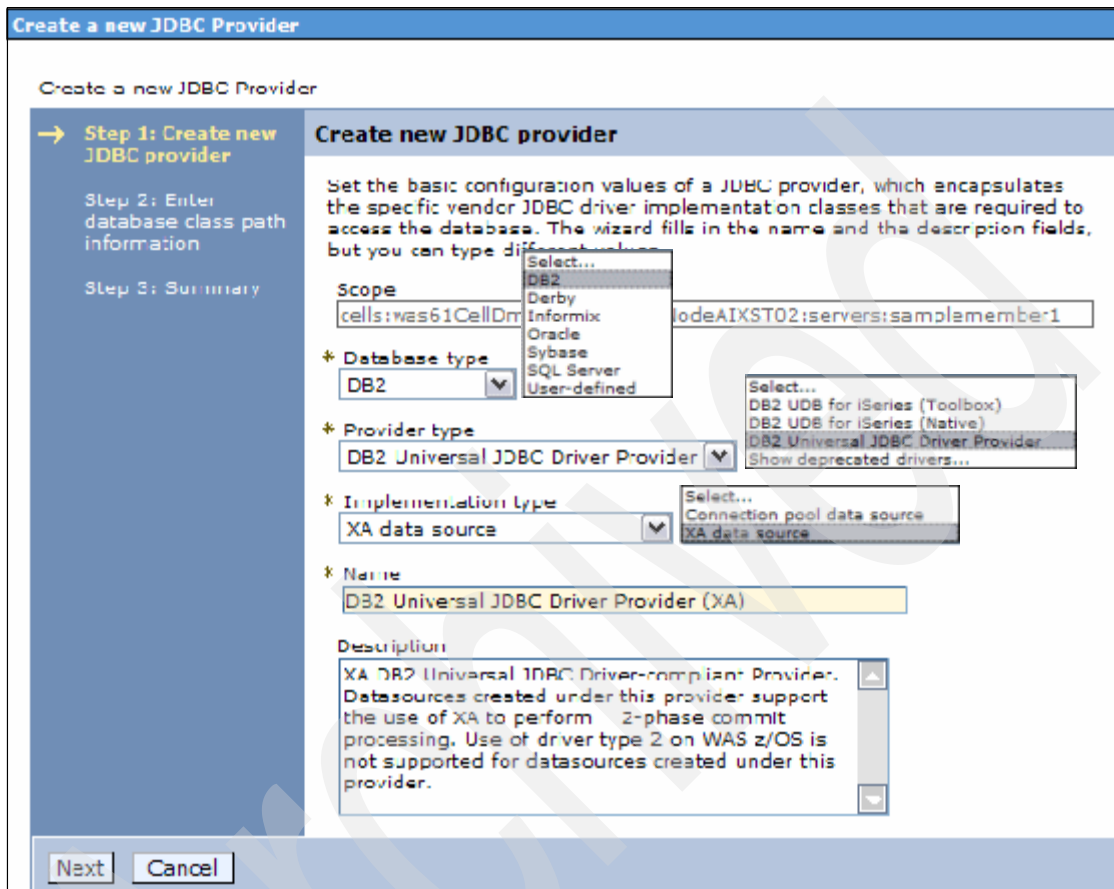


Figure 6-6 Define a new JDBC provider: window 1

- Database type
Select the vendor-specific database type. If the database type you need is not in the list, select **User-defined** and consult the vendor documentation for the specific properties that will be required.
- Provider type
Select from a predefined list of supported provider types, based on the database type you select.
- Implementation type
Select from the implementation types for the provider type you selected.

- Name

Specify a Name for this driver.

6. Click **Next**. The settings page for your JDBC database class path appears. Figure 6-7 on page 315 shows the configuration page for a DB2 Universal JDBC Provider.

Create a new JDBC Provider

Create a new JDBC Provider

Step 1: Create new JDBC provider

→ **Step 2: Enter database class path information**

Step 3: Summary

Enter database class path information

Set the environment variables that represent the JDBC driver class files, which WebSphere(R) Application Server uses to define your JDBC provider. This wizard page displays the file names; you supply only the directory locations of the files. Use complete directory paths when you type the JDBC driver file locations. For example: /home/db2nst1/sqllib/java on Linux(1M). If a value is specified for you, you may click **Next** to accept the value.

Class path:

`${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar`
`${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar`
`${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_csuz.jar`

Directory location for "db2jcc.jar, db2jcc_license_csuz.jar" which is saved as WebSphere variable `${DB2UNIVERSAL_JDBC_DRIVER_PATH}`

Native library path

Directory location which is saved as WebSphere variable `${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}`

Previous **Next** **Cancel**

Figure 6-7 Define a new JDBC provider: window 2

Enter the JDBC provider properties.

- Classpath

This field is a list of paths or JAR file names that together form the location for the resource provider classes. For example, c:\sqllib\java\db2java.zip is the path if the data source connects to DB2. Separate the entries by pressing Enter between each one.

- Library path

This field specifies the values for the global variable DB2UNIVERSAL_JDBC_DRIVER_PATH, which indicates the classpath jar's location.

- Native Library Path

This field is an optional path to any native libraries. Entries are required if the JDBC provider chosen uses non-Java, or native, libraries. The global variable for this is DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH.

Note: The default settings use environment variables in the path names for the classpath and native library path settings. After you complete the process of defining the data source, if you did not during this process, make sure to update the environment variables used to reflect the proper locations of these files on your system. You can set variables by selecting **Environment** → **WebSphere Variables** in the navigation menu.

Refer to 4.1.10, “Using variables” on page 156 for more information about WebSphere environment variables.

7. After verifying the settings, click **Finish**. This enables the links to create data sources under the Additional Properties section.

To create one or more data sources for this provider, proceed to 6.2.5, “Creating JDBC data source” on page 317. If you are not ready to create the data source yet, click **OK** and then save your changes.

Tip: To make a data source available on multiple nodes using different directory structures, complete the following steps using the administrative console:

1. Define the JDBC provider at the cell scope. Use WebSphere environment variables for the classpath and native path.
2. Create the data source that uses this JDBC provider at the cell scope. All files defined at the cell scope are replicated to every node in the cell.
3. For the paths to the driver on each node to be unique, use a variable to specify the driver location and have that variable be defined differently on each node.

For example, `${DRIVER_PATH}` can be used for the classpath in the provider definition. You can then define a variable called `${DRIVER_PATH}` at the cell scope to act as a default driver location. Then you can override that variable on any node by defining `${DRIVER_PATH}` at the node scope. The node-level definition takes precedence over the cell-level definition.

6.2.5 Creating JDBC data source

Data sources are associated with a specific JDBC provider and can be viewed or created from the JDBC provider configuration page. You have two options when creating a data source, depending on the J2EE support of the application. This section discusses creating or modifying data sources for J2EE 1.3 and J2EE 1.4 applications.

For information about using data sources with J2EE 1.2 applications, see the *Data sources (Version 4)* topic in the Information Center.

To create a data source, do the following:

1. Expand **Resources** → **JDBC** in the navigation tree and select **Data sources**.
2. Select the scope. Although you can select **All** to view all resources, you must select a specific scope to create a resource.

3. Click **New** to create a new data source. This will start a wizard (Figure 6-8 on page 318).

Create a data source

→ **Step 1: Enter basic data source information**

Step 2: Select JDBC provider

Step 3: Enter database specific properties for the data source

Step 4: Summary

Enter basic data source information

Set the basic configuration values of a data source for association with your JDBC provider. A data source supplies the physical connections between application server and the database.

Requirement: Use the Data sources (WebSphere(R) Application Server V6 console pages if your applications are based on the Enterprise JavaBeans (TM) (EJB) 1.0 specification or the Java(TM) Servlet 2.2 specification.

Scope
cells:kadw028Cell01:nodes:kadw028Node03

* Data source name
TESTDB

* JNDI name
jdbc/testdb

Component-managed authentication alias and XA recovery authentication alias

Select a component-managed authentication alias. The selected authentication alias will also be set as the XA recovery authentication alias if your JDBC Provider supports XA. If you choose to [create a new J2C authentication alias](#), the wizard will be canceled.

kadw028Node04Cell/samples

Next Cancel

Figure 6-8 Data source general properties

- **Data source name**
This field is a name by which to administer the data source. Use a name that is suggestive of the database name or function.
- **JNDI name**
This field refers to the data source's name as registered in the application server's name space. When installing an application that contains modules with JDBC resource references, the resources defined by the deployment descriptor of the module need to be bound to the JNDI name of the resources. For example, `jdbc/<database_name>`.

- Component-managed authentication alias and Authentication alias for XA recovery

This field specifies a user ID and password to be used by J2C security. The entry references authentication data defined in the J2C authentication data entries. Make new entries by selecting the **J2EE Connector Architecture (J2C) authentication data entries** link on the data source configuration window. See Figure 6-8 on page 318. On the other hand, the Authentication alias for XA recovery is used to specify the authentication alias that should be used during XA recovery processing.

Click **Next**.

4. Now you need to specify database specific properties. These are shown on the right of Figure 6-12 on page 322. Click **Next**.

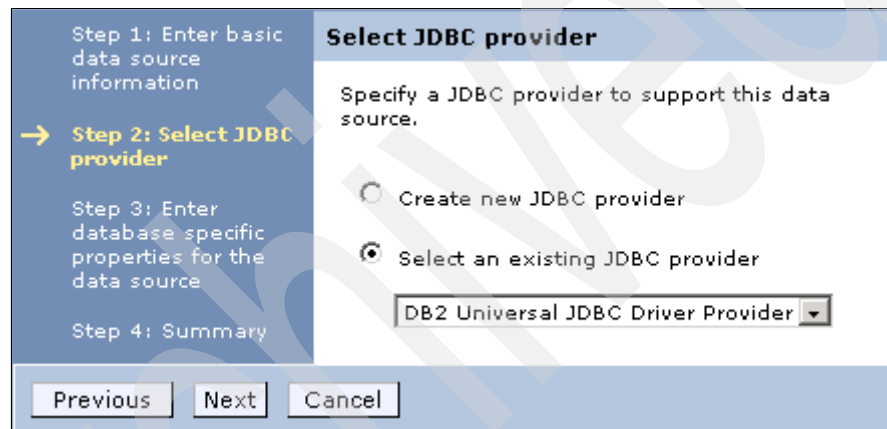


Figure 6-9 Select a JDBC provider

This window allows you to select a JDBC provider or to create a new one. If you create a new JDBC provider, you will be routed through the windows seen earlier in 6.2.4, “Creating a JDBC provider” on page 311. If you select an existing JDBC provider you will continue with the next step here.

In this case, we select an existing JDBC provider and click **Next**.

The entries shown in Figure 6-10 are specific to the JDBC driver and data source type. Figure 6-10 shows the properties for the DB2 Universal data source.

Figure 6-10 Database-specific properties

- ▶ Database Name
The name of the database (or the cataloged alias).
- ▶ Driver type
The type of JDBC Driver (2 or 4) used to access the database.
- ▶ DB2 server name and port.
The DB2 instance's server name and its listening port (by default 50000).
- ▶ Container-managed persistence (CMP)
This field specifies if the data source is to be used for container-managed persistence of EJB beans. Checking this box causes a CMP connection factory corresponding to this data source to be created for the relational resource adapter. The connector factory created has the name `<datasourcename>_CF` and is registered in JNDI under the entry `eis/<jndi_name>_CMP`.

You can see the properties of the just created connection factory by selecting **Resources** → **Resource Adapters** → **Resource Adapters**. Enable the **Show built-in resources** check box (new in V6.1) in the preferences. Select

WebSphere Relational Resource Adapter → CMP Connection Factories.
Be sure to set the scope so it is the same as that for the data source.

Click **Next**.

5. You will see a summary of the options you have chosen. Click **Next** to create the data source.

The new data source will be listed in the table of resources. You can test the new connection by checking the box to the left of the data source and clicking **Test Connection**.

You can view or modify settings for the new data source by clicking the name in the resources list. Figure 6-11 shows a portion of the details page. Other settings not shown include the database details and the component-managed authentication settings.

Test connection

General Properties

* Scope
cells:kadw028Cell01:nodes:kadw028Node03

* Provider
DB2 Universal JDBC Driver Provider

* Name
TESTDB

JNDI name
jdbc/testdb

☒ Use this data source in container managed persistence (CMP)

Description
DB2 Universal Driver Datasource

Category

Additional Properties

- [Connection pool properties](#)
- [WebSphere Application Server data source properties](#)
- [Custom properties](#)

Related Items

- [JAAS - J2C authentication data](#)

Figure 6-11 Data source details page

Adding or updating custom properties

To add or update custom properties, do the following:

1. Open the data source by clicking the name in the resource list.
2. Click **Custom Properties** in the Additional Properties table, to provide or update data source properties that might be required. A list of predefined properties based on the data source type appears.
3. Click **New** to add a custom property, or click a property name to modify it.

Figure 6-12 shows the first few custom properties configured for a data source connecting to a DB2 database.

New Delete				
   				
Select	Name	Value	Description	Required
☐	description		The description of this datasource.	false
☐	traceLevel		The DB2 trace level for logging to the logWriter or trace file. Possible trace levels are: TRACE NONE = 0,TRACE CONNECTION CALLS = 1,TRACE STATEMENT CALLS = 2,TRACE RESULT SET CALLS = 4,TRACE DRIVER CONFIGURATION = 16,TRACE CONNECTS = 32,TRACE DRDA FLOWS = 64,TRACE RESULT SET META DATA = 128,TRACE PARAMETER META DATA = 256,TRACE DIAGNOSTICS = 512,TRACE SQLJ = 1024,TRACE ALL = -1,.	false
☐	traceFile		The trace file to store the trace output. If you specify the trace file, the DB2 Jcc trace will be logged in this trace file. If this property is not specified and the WAS.database trace group is enabled, then both WebSphere trace and DB2 trace will be logged into the WebSphere trace file.	false

Figure 6-12 Data Source custom properties

4. Click **OK** when you finish.

Configure connection pooling properties

The link to connection pooling settings is found in the Additional Properties section of the data source configuration window. See Figure 6-8 on page 318.

General Properties

Scope
cells:kadw028Cell01:nodes:kadw028Node03

Connection timeout
180 seconds

Maximum connections
10 connections

Minimum connections
1 connections

Reap time
180 seconds

Unused timeout
1800 seconds

Aged timeout
0 seconds

Purge policy
EntirePool

Apply OK Reset Cancel

Additional Properties

- [Advanced connection pool properties](#)
- [Connection pool custom properties](#)

Figure 6-13 Data source connection pool properties

► Connection Timeout

Specify the interval, in seconds, after which a connection request times out and a `ConnectionWaitTimeoutException` is thrown. This can occur when the pool is at its maximum (Max Connections) and all of the connections are in use by other applications for the duration of the wait.

For example, if Connection Timeout is set to 300 and the maximum number of connections is reached, the Pool Manager waits for 300 seconds for an available physical connection. If a physical connection is not available within this time, the Pool Manager throws a `ConnectionWaitTimeoutException`.

Tip: If Connection Timeout is set to 0, the pool manager waits as long as necessary until a connection is allocated.

► Max Connections

Specify the maximum number of physical connections that can be created in this pool.

These are the physical connections to the back-end database. Once this number is reached, no new physical connections are created and the requester waits until a physical connection that is currently in use is returned to the pool, or a `ConnectionWaitTimeoutException` is thrown.

For example, if Max Connections is set to 5, and there are five physical connections in use, the Pool Manager waits for the amount of time specified in Connection Timeout for a physical connection to become free. If, after that time, there are still no free connections, the Pool Manager throws a `ConnectionWaitTimeoutException` to the application.

► Min Connections

Specify the minimum number of physical connections to be maintained. Until this number is reached, the pool maintenance thread does not discard any physical connections. However, no attempt is made to bring the number of connections up to this number.

For example, if Min Connections is set to 3, and one physical connection is created, that connection is not discarded by the Unused Timeout thread. By the same token, the thread does not automatically create two additional physical connections to reach the Min Connections setting.

► Reap Time

Specify the interval, in seconds, between runs of the pool maintenance thread.

For example, if Reap Time is set to 60, the pool maintenance thread runs every 60 seconds. The Reap Time interval affects the accuracy of the Unused Timeout and Aged Timeout settings. The smaller the interval you set, the greater the accuracy. When the pool maintenance thread runs, it discards any connections that have been unused for longer than the time value specified in Unused Timeout, until it reaches the number of connections specified in Min Connections. The pool maintenance thread also discards any connections that remain active longer than the time value specified in Aged Timeout.

Tip: If the pool maintenance thread is enabled, set the Reap Time value less than the values of Unused Timeout and Aged Timeout.

The Reap Time interval also affects performance. Smaller intervals mean that the pool maintenance thread runs more often and degrades performance.

► Unused Timeout

Specify the interval in seconds after which an unused or idle connection is discarded.

Tip: Set the Unused Timeout value higher than the Reap Timeout value for optimal performance. Unused physical connections are only discarded if the current number of connections not in use exceeds the Min Connections setting.

For example, if the unused timeout value is set to 120, and the pool maintenance thread is enabled (Reap Time is not 0), any physical connection that remains unused for two minutes is discarded. Note that accuracy of this timeout, as well as performance, is affected by the Reap Time value. See the Reap Time bullet for more information.

► Aged Timeout

Specify the interval in seconds before a physical connection is discarded, regardless of recent usage activity.

Setting Aged Timeout to 0 allows active physical connections to remain in the pool indefinitely. For example, if the Aged Timeout value is set to 1200, and the Reap Time value is not 0, any physical connection that remains in existence for 1200 seconds (20 minutes) is discarded from the pool. Note that accuracy of this timeout, as well as performance, is affected by the Reap Time value. See Reap Time for more information.

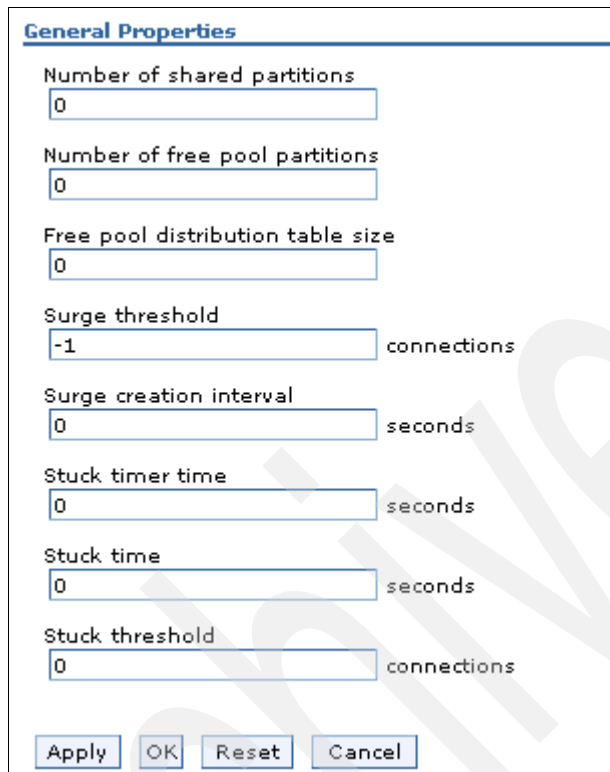
Tip: Set the Aged Timeout value higher than the Reap Timeout value for optimal performance.

► Purge Policy

Specify how to purge connections when a stale connection or fatal connection error is detected.

Valid values are EntirePool and FailingConnectionOnly. If you choose EntirePool, all physical connections in the pool are destroyed when a stale connection is detected. If you choose FailingConnectionOnly, the pool attempts to destroy only the stale connection. The other connections remain in the pool. Final destruction of connections that are in use at the time of the error might be delayed. However, those connections are never returned to the pool.

Selecting the **Advanced connection pool properties** link allows you to modify the properties shown in Figure 6-14 on page 326.



General Properties

Number of shared partitions

Number of free pool partitions

Free pool distribution table size

Surge threshold
 connections

Surge creation interval
 seconds

Stuck timer time
 seconds

Stuck time
 seconds

Stuck threshold
 connections

Figure 6-14 Advanced connection pool properties

These properties require advanced knowledge of how connection pooling works and how your system performs. For information about these settings, see the *Connection pool advanced settings* topic in the Information Center.

WebSphere Application Server data source properties

You can set the properties that apply to the WebSphere Application Server connection, rather than to the database connection, by selecting the **WebSphere Application Server data source properties** link under the Additional Properties section of the data source configuration page. See Figure 6-11 on page 321. Clicking the link gives you the window shown in Figure 6-15 on page 327.

General Properties

Statement cache size
 statements

☐ Enable multithreaded access detection

☐ Enable database reauthentication

☐ Enable JMS one-phase optimization support

☐ Manage cached handles

☒ Log missing transaction context

Pretest connection properties

☐ Pretest existing pooled connections
 Retry interval
 seconds

☐ Pretest new connections
 Number of retries

 Retry interval
 seconds

Pretest SQL string

Figure 6-15 WebSphere data source custom properties

► **Statement Cache Size**

Specify the number of prepared statements that are cached per connection. A prepared statement is a precompiled SQL statement that is stored in a prepared statement object. This object is used to execute the given SQL statement multiple times. The WebSphere Application Server data source optimizes the processing of prepared statements.

In general, the more statements your application has, the larger the cache should be. For example, if the application has five SQL statements, set the statement cache size to 5, so that each connection has five statements.

- Enable multithreaded access detection

If you enable this feature, the application server detects the existence of access by multiple threads.

- Enable database reauthentication

Connection pool searches do not include the user name and password. If you enable this feature, a connection can still be retrieved from the pool, but you must extend the `DataStoreHelper` class to provide implementation of the `doConnectionSetupPerTransaction()` method where the reauthentication takes place.

Connection reauthentication can help improve performance by reducing the overhead of opening and closing connections, particularly for applications that always request connections with different user names and passwords.

- Manage cached handles

When you call the `getConnection()` method to access a database, you get a connection handle returned. The handle is not the physical connection, but a representation of a physical connection. The physical connection is managed by the connection manager. A cached handle is a connection handle that is held across transaction and method boundaries by an application.

This setting specifies whether cached handles should be tracked by the container. This can cause overhead and only should be used in specific situations. For more information about cached handles, see the *Connection Handles* topic in the Information Center.

- Transaction context logging.

The J2EE programming model indicates that connections should always have a transaction context. However, some applications do not have a context associated with them. This option tells the container to log that there is a missing transaction context in the activity log when the connection is obtained.

- Pretest existing pooled connections

If you check this box, the application server tries to connect to this data source before it attempts to send data to or receive data from this source. If you select this property, you can specify how often, in seconds, the application server retries to make a connection if the initial attempt fails. The pretest SQL string is sent to the database to test the connection.

- Pretest new connections

If you check this box, the application server test the initial connection to database. If you select this property, you specify how often, in seconds, the application server retries to make a connection and how many times it tries. The pretest SQL string is sent to the database to test the connection.

6.3 JCA resources

The J2EE Connector architecture (JCA) defines a standard architecture for connecting the J2EE platform to heterogeneous Enterprise Information Systems (EIS), for example, ERP, mainframe transaction processing, database systems, and existing applications not written in the Java programming language. By defining a set of scalable, secure, and transactional mechanisms, the JCA enables the integration of EISs with application servers and enterprise applications.

WebSphere Application Server V6 provides a complete implementation of the JCA 1.5 specification, including the features of the JCA 1.0 Specification:

- ▶ Connection sharing (res-sharing-scope)
- ▶ A get/use/close programming model for connection handles
- ▶ A get/use/cache programming model for connection handles
- ▶ XA, Local, and No Transaction models of resource adapters, including XA recovery
- ▶ Security options A and C, as in the specification
- ▶ Applications with embedded .rar files

The new features for the JCA 1.5 specification are:

- ▶ Deferred enlistment transaction optimization
- ▶ Lazy connection association optimization
- ▶ Inbound communication from an enterprise information system (EIS) to a resource adapter
- ▶ Inbound transactions from an EIS to a resource adapter
- ▶ Work management, enabling a resource adapter to put work on separate threads and pass execution context, such as inbound transactions, to the thread
- ▶ Life cycle management, enabling a resource adapter to be stopped and started

The JCA Resource Adapter is a system-level software driver supplied by EIS vendors or other third-party vendors. It provides the following functionality:

- ▶ Provides connectivity between J2EE components, such as an application server or an application client and an EIS.
- ▶ Plugs into an application server.

- ▶ Collaborates with the application server to provide important services, such as connection pooling, transaction, and security services.

JCA defines the following set of system-level contracts between an application server and EIS:

- A *connection management contract* lets an application server pool connect to an underlying EIS, and lets application components connect to an EIS. This leads to a scalable application environment that can support a large number of clients requiring access to EISs.
- A *transaction management contract* between the transaction manager and an EIS supports transactional access to EIS resource managers. This contract lets an application server use a transaction manager to manage transactions across multiple resource managers. This contract also supports transactions that are managed internally to an EIS resource manager without the necessity of involving an external transaction manager.
- A *security contract* enables a secure access to an EIS. This contract provides support for a secure application environment, reducing security threats to the EIS and protecting valuable information resources managed by the EIS.

The resource adapter implements the EIS-side of these system-level contracts.

- ▶ Implements the Common Client Interface (CCI) for EIS access.

The CCI defines a standard client API through which a J2EE component accesses the EIS. This simplifies writing code to connect to an EIS data store.

The resource adapter provides connectivity between the EIS, the application server, and the enterprise application via the CCI.

- ▶ Implements the standard Service Provider Interface (SPI).

The SPI integrates the transaction, security, and connection management facilities of an application server (JCA Connection Manager) with those of a transactional resource manager

Multiple resource adapters (one resource adapter per type of EIS) are pluggable into an application server. This capability enables application components deployed on the application server to access the underlying EISs. This is shown in Figure 6-16.

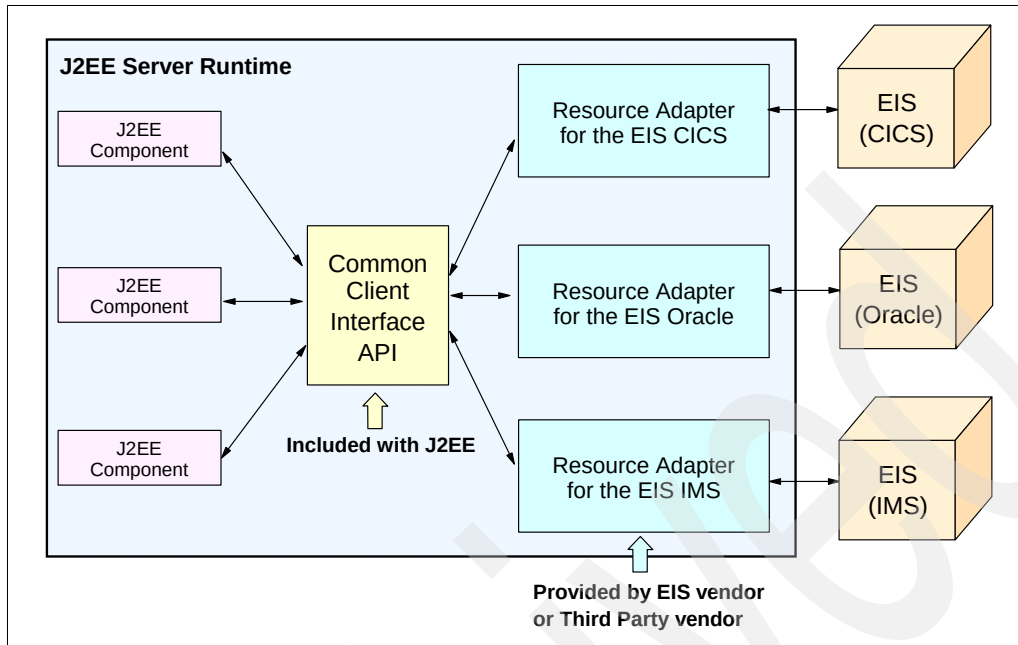


Figure 6-16 Common Client Interface API

The benefits of JCA include:

- ▶ Once an application server implements JCA, any JCA-compliant resource adapter can plug in.
- ▶ Once a resource adapter implements JCA, it can plug in to any JCA-compliant application server.
- ▶ Each EIS requires just one implementation of the resource adapter.
- ▶ The common client interface simplifies application integration with diverse EISs.

6.3.1 WebSphere Application Server JCA support

In WebSphere Application Server, two types of objects are configured for JCA support:

- ▶ Resource adapters
- ▶ Connection factories

The role of the WebSphere administrator is to:

- ▶ Install and define the resource adapter.
- ▶ Define one or more connection factories associated with the resource adapter.

From the application point of view, the application using the resource adapter requests a connection from the connection factory through a JNDI lookup. The connection factory connects the application to the resource adapter.

Resource adapter

- ▶ A WebSphere resource adapter administrative object represents the library that supplies implementation code for connecting applications to a specific EIS, such as CICS® or SAP®. Resource adapters are stored in a Resource Adapter Archive (RAR) file, which is a Java archive (JAR) file used to package a resource adapter for the connector architecture. The file has a standard file extension of .rar.

A RAR file can contain the following:

- ▶ EIS-supplied resource adapter implementation code in the form of JAR files or other executables, such as DLLs
- ▶ Utility classes
- ▶ Static documents, such as HTML files for developer documentation, not used for run time
- ▶ J2C common client interfaces, such as cci.jar
- ▶ A mandatory deployment descriptor (ra.xml)

This deployment descriptor instructs the application server about how to use the resource adapter in an application server environment. The deployment descriptor contains information about the resource adapter, including security and transactional capabilities, and the ManagedConnectionFactory class name.

The RAR file or JCA resource adapter is provided by your EIS vendor.

WebSphere provides two JCA resource adapters:

- ▶ The WebSphere Relational Resource Adapter, used to connect to relational databases using JDBC
- ▶ The SIB JMS Resource Adapter, used to connect to the default messaging provider

Connection factory

The WebSphere connection factory administrative object represents the configuration of a specific connection to the EIS supported by the resource adapter. The connection factory can be thought of as a holder of a list of connection configuration properties.

Application components, such as CMP enterprise beans, have `cmpConnectionFactory` descriptors that refer to a specific connection factory, not to the resource adapter.

6.3.2 Installing and configuring resource adapters

To use a resource adapter, you need to install the resource adapter code and create connection factories that use the adapter. Resource adapter configuration is stored in the `resources.xml` file.

To install a resource adapter (.rar file), do the following:

1. From the administrative console, expand **Resources** from the navigation tree and click **Resource Adapters**.

2. Select a scope, and if you want to see the WebSphere built-in resources, select the **Show built-in resources** preference. A list of existing resources will be shown (Figure 6-15).

Resource adapters

Use this page to manage resource adapters, which provide the fundamental interface for connecting applications to an Enterprise Information System (EIS). The WebSphere(R) Relational Resource Adapter is embedded within the product to provide access to relational databases. To access another type of EIS, use this page to install a standalone resource adapter archive (RAR) file. You can configure multiple resource adapters for each installed RAR file.

⊕ Scope: Cell=kadw028Cell01

⊖ Preferences

Maximum rows
20

☐ Retain filter criteria.

☒ Show built-in resources

Apply Reset

Install RAR New Delete

Select	Name	Scope
☐	SIB JMS Resource Adapter	Cell=kadw028Cell01
☐	WebSphere Relational Resource Adapter	Cell=kadw028Cell01
Total 2		

Figure 6-17 JCA resource adapters

3. Click **Install RAR** to install a new resource adapter.
4. Enter the path to the RAR file supplied by your EIS vendor. It can reside locally, on the same machine as the browser, or on any of the nodes in your cell. See Figure 6-18 on page 335.

Install RAR File

Use this page to install a RAR file in one of two ways. You can either upload a RAR file from the local file system, or specify an existing RAR file on a server. The RAR file must be installed at the node level, and you can select the node below.

Path

☒ Local path:

Specify path
M:\Downloads\CICS Resource adapter\cicseci.rar

☐ Server path:

Specify path

Scope

Node
kadw028Node03

Figure 6-18 RAR file location

5. Select the node where you want to install the RAR file. You have to install the file on each node separately.

6. Click **Next**. The Configuration page for the resource adapter selected is displayed. This is shown in Figure 6-19 on page 336.

General Properties

* Scope
cells:kadw028Cell01:nodes:kadw028Node03

Name
ECIResourceAdapter

Description
CICS J2EE ECI Resource Adapter

Archive path
\${CONNECTOR_INSTALL_ROOT}

Class path

Native path

OK Reset Cancel

Figure 6-19 JCA resource adapter properties

In this example, you do not have to configure any properties. The defaults combined with the information supplied in the RAR file provide all the information needed. However, you have the option of configuring the following:

- Name
Create an administrative name for the resource adapter.
- Description
Create an optional description of the resource adapter, for your administrative records.

- Archive path

This field is the path where the RAR file is installed. If this property is not specified, the archive will be extracted to the absolute path represented by the `${CONNECTOR_INSTALL_ROOT}` variable. The default is `<profile_home>/installedConnectors/<adaptername.rar>`

- Class path

A list of paths or JAR file names that together form the location for the resource adapter classes. The resource adapter codebase itself, the RAR file, is automatically added to the classpath.

- Native path

This is a list of paths that together form the location for the resource adapter native libraries (.dll, and .so files).

7. Click **OK**.

6.3.3 Configuring J2C connection factories

Note: The terms J2C and JCA both refer to J2EE Connector Architecture and they are used here interchangeably.

A J2C connection factory represents a set of connection configuration values. Application components such as EJBs have `<resource-ref>` descriptors that refer to the connection factory, not the resource adapter. The connection factory is just a holder of a list of connection configuration properties. In addition to the arbitrary set of configuration properties defined by the vendor of the resource adapter, there are several standard configuration properties that apply to the connection factory. These standard properties are used by the connection pool manager in the application server run time and are not used by the vendor supplied resource adapter code.

To create a J2C connection factory, do the following:

1. Select **Resources** → **J2C connection factories**. You will see a list of J2C connection factories at the selected scope.
2. Click **New** to create a new connection factory, or select an existing one to modify the connection factory properties.

The J2C Connection Factory Configuration page is shown in Figure 6-20 on page 338.

General Properties

* Scope
cells:kadw028Cell01:nodes:kadw028Node03

Provider
ECIResourceAdapter
Create New Provider

* Name
eciCF

JNDI name
eis/cicseci

Description

* Connection factory interface
javax.resource.cci.ConnectionFactory

Category

Component-managed authentication alias
kadw028CellManager01/CICS

The additional properties will not be available until the general properties for this item are applied or saved.

Additional Properties

- Connection pool properties
- Advanced connection factory properties
- Custom properties

Related Items

Figure 6-20 J2C connection factory properties

The general properties are:

- Name

Type an administrative name for the J2C connection factory.

- JNDI name

This field is the connection factory name to be registered in the application server's name space, including any naming subcontext.

When installing an application that contains modules with J2C resource references, the resources defined by the deployment descriptor of the module need to be bound to the JNDI name of the resource.

As a convention, use the value of the Name property prefixed with eis/, for example,

`eis/<ConnectionFactoryName>`

- Description
This is an optional description of the J2C connection factory, for your administrative records.
- Connection factory interface
This field is the name of the connection factory interfaces supported by the resource adapter.
- Category
Specify a category that you can use to classify or group the connection factory.
- Component-managed authentication alias
This authentication alias is used for component-managed sign-on to the resource.

Deprecated in V6.1: The following security settings are deprecated:

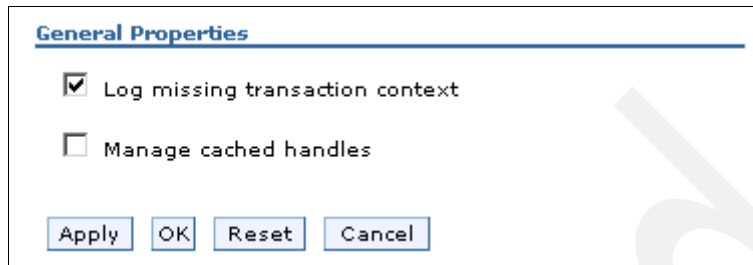
- ▶ Container managed authentication alias
- ▶ Authentication preference
- ▶ Mapping configuration alias

Resource authentication settings should be used instead. For more information, see 6.7, “Resource authentication” on page 361.

3. Click **Apply**. The links under the Additional Properties section for connection pool, advanced connection factory, and custom properties become active.

The connection pool properties are configured the same as for a JDBC data source. For information about these settings, see “Configure connection pooling properties” on page 322.

The advanced connection factory properties are shown in Figure 6-21 on page 340.



General Properties

☒ Log missing transaction context

☐ Manage cached handles

Apply OK Reset Cancel

Figure 6-21 Advanced connection factory properties

- Manage cached handles

When you call the `getConnection()` method to access a database, you get a connection handle returned. The handle is not the physical connection, but a representation of a physical connection. The physical connection is managed by the connection manager. A cached handle is a connection handle held across transaction and method boundaries by an application.

This setting specifies whether cached handles should be tracked by the container. This can cause overhead and only should be used in specific situations. For more information about cached handles, see the *Connection Handles* topic in the Information Center.

- Log missing transaction context

The J2EE programming model indicates that connections should always have a transaction context. However, some applications do not have a context associated with them. This option tells the container to log that there is a missing transaction context in the activity log when the connection is obtained.

6.3.4 Using resource adapters from an application

Example 6-1 shows how you might access the CICS ECI resource adapter from an application. This code snippet assumes you have a resource reference called `eis/ref/ECICICS` that points to a `javax.resource.cci.ConnectionFactory` with the JNDI name `eis/ECICICS`. It is a minimal sample, with no connection factory caching, and so on.


```
private int getRate(String source) throws java.lang.Exception {

    // get JNDI context
    javax.naming.InitialContext ctx = new javax.naming.InitialContext();
    // get local JNDI environment
    javax.naming.Context env =
(javax.naming.Context)ctx.lookup("java:comp/env");
    javax.resource.cci.ConnectionFactory connectionFactory =
        (javax.resource.cci.ConnectionFactory) env.lookup("eis/ref/ECICICS");

    // get a connection to the EIS
    javax.resource.cci.Connection connection =
connectionFactory.getConnection();

    // create an interaction and a CICS ECI specific interaction spec
    javax.resource.cci.Interaction interaction =
connection.createInteraction();
    com.ibm.connector2.cics.ECIInteractionSpec interactionSpec = new
com.ibm.connector2.cics.ECIInteractionSpec();

    // create the comm area record
    source = (source.trim().toUpperCase()+" ").substring(0, 12);
    GenericRecord record = new GenericRecord((source).getBytes("IBM037"));

    // set the CICS program name we want to call
    interactionSpec.setFunctionName("CALCRATE");

    // invoke the CICS program
    interaction.execute(interactionSpec, record, record);

    // close the interaction and the connection
    interaction.close();
    connection.close();

    // get the results from the return comm area record
    byte[] commarea = record.getCommarea();
    int value = Integer.parseInt(new String(commarea,
"IBM037").substring(8,12).trim());
    return value;

}
```

6.4 JavaMail resources

The JavaMail™ APIs provide a platform and protocol-independent framework for building Java-based mail client applications. The JavaMail APIs are generic for sending and receiving mail. They require service providers, known in WebSphere as *protocol providers*, to interact with mail servers that run the protocols.

A JavaMail provider encapsulates a collection of protocol providers. WebSphere Application Server has a Built-in Mail Provider that encompasses three protocol providers: SMTP, IMAP, and POP3. These protocol providers are installed as the default and should be sufficient for most applications.

- ▶ Simple Mail Transfer Protocol (SMTP)

This is a popular transport protocol for sending mail. JavaMail applications can connect to an SMTP server and send mail through it by using this SMTP protocol provider.

- ▶ Post Office Protocol (POP3)

This is the standard protocol for receiving mail.

- ▶ Internet Message Access Protocol (IMAP)

This is an alternative protocol to POP3 for receiving mail.

Note: In this section, the terms *JavaMail provider* and *mail provider* are used interchangeably.

To use other protocols, you must install the appropriate service provider for those protocols.

In addition to service providers, JavaMail requires the Java Activation Framework (JAF) as the underlying framework to deal with complex data types that are not plain text, like Multipurpose Internet Mail Extensions (MIME), Uniform Resource Locator (URL) pages, and file attachments.

The JavaMail APIs, the JAF, the service providers and the protocols are shipped as part of WebSphere Application Server using the following Sun licensed packages:

- ▶ mail.jar

This file contains the JavaMail APIs, and the SMTP, IMAP, and POP3 service providers.

- ▶ activation.jar

This file Contains the JavaBeans Activation Framework.

Figure 6-22 on page 343 illustrates the relationship among the different JavaMail components.

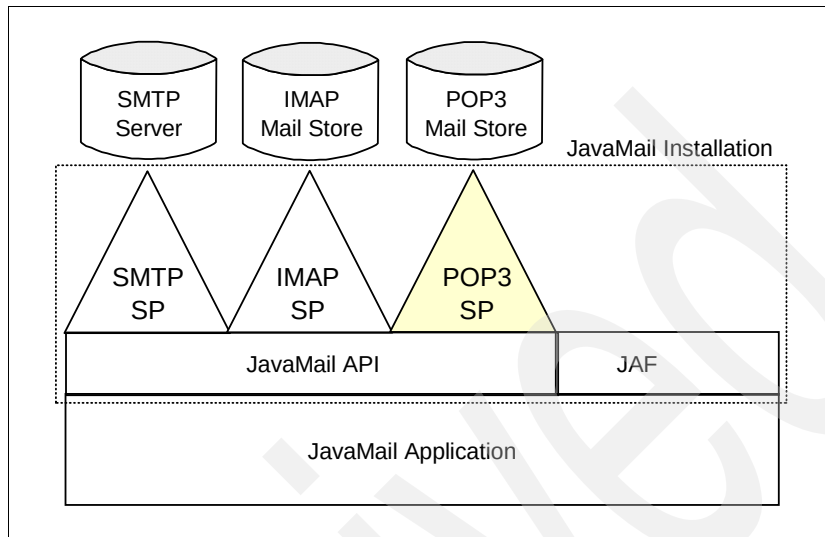


Figure 6-22 JavaMail components

WebSphere Application Server supports JavaMail Version 1.3 and the JavaBeans Activation Framework (JAF) Version 1.0. All Web components of WebSphere, including servlets, JSPs, EJBs, and application clients, support JavaMail.

6.4.1 JavaMail sessions

A JavaMail session object, or session administrative object, is a resource used by the application to obtain connections to a mail server. A mail session object manages the configuration options and user authentication information used to interact with the mail system. JavaMail sessions are configured to use a particular JavaMail provider.

6.4.2 Configuring the mail provider

A mail provider encapsulates a collection of protocol providers. Protocol providers interact with JavaMail APIs and mail servers running those protocols. WebSphere Application Server has a built-in mail provider that encompasses three protocol providers: SMTP, IMAP and POP3. These protocol providers are installed by default and should be sufficient for most applications. However, you can configure a new provider if necessary.

To configure a new mail provider, complete the following steps from the administrative console:

1. Expand **Resources** from the navigation tree and click **Mail Providers**.
2. Select **Scope** and click **Apply**. The scope determines whether JavaMail resources configured to use this provider will be available at the cell, node, or the application server level.

Figure 6-23 shows the mail provider installed with WebSphere. The built-in mail provider is available to all the application servers in the cell.

Mail Providers

Use this page to create a mail provider, an object that encapsulates the protocol providers that your mail application requires. Select the built-in mail provider for access to the three default protocol providers: SMTP, IMAP, and POP3. These protocol providers suffice for most applications.

⊕ Scope: Cell=**kadw028Cell01**

⊕ Preferences

New Delete

⊞ ⊞ ⊞ ⊞

Select	Name ▾	Scope ▾	Description ▾
	Built-in Mail Provider	Cell=kadw028Cell01	The built-in mail provider

Total 1

Figure 6-23 Mail provider page

3. Click **New** to configure a new mail provider.
4. Enter a name and a description, and then click **Apply**. The properties required to configure a new mail provider are shown in Figure 6-24 on page 345.

Mail Providers ?

[Mail Providers](#) > **MyNewMail**

Use this page to create a mail provider, an object that encapsulates the protocol providers that your mail application requires. Select the built-in mail provider for access to the three default protocol providers: SMTP, IMAP, and POP3. These protocol providers suffice for most applications.

Configuration

General Properties	Additional Properties
* Scope cells:was61CellDmgrPrf1	■ Protocol providers ■ Mail sessions ■ Custom properties
* Name MyNewMail	
Description	
Apply OK Reset Cancel	

Figure 6-24 Mail Provider general properties

5. Click **Protocol Providers** under the Additional Properties section.

6. Click **New** to add a protocol provider. See Figure 6-25.

The screenshot shows a web browser window titled "Mail Providers" with a help icon. The breadcrumb navigation is "Mail Providers > MyNewMail > Protocol providers > New". Below this is a descriptive paragraph: "Use this page to set the properties of a protocol provider, which provides the implementation class for a specific protocol to support communication between your JavaMail application and mail servers. After saving your settings, return to the mail provider page to find the link for configuring mail sessions." A "Configuration" tab is active. Under the "General Properties" section, there are four fields: "Scope" (text input with value "cells:was61CellDmgrPrf1"), "Protocol" (text input), "Class name" (text input), and "Class path" (text area with up/down arrows). Below these is a "Type" dropdown menu set to "STORE". At the bottom are four buttons: "Apply", "OK", "Reset", and "Cancel".

Figure 6-25 Protocol provider configuration page

The properties to configure are:

- Protocol

This field specifies the protocol name.

- Classname

This field specifies the implementation class for the specific protocol provider. The class must be available in the classpath.

- Classpath

This field specifies the path to the JAR files that contain the implementation classes for this protocol provider.

– Type

This field specifies the type of protocol provider. Valid options are:

- STORE: This protocol is used for receiving mail.
- TRANSPORT: This protocol is used for sending mail.

For guidance, you can look at the protocol providers provided with the built-in mail provider, as shown in Figure 6-26.

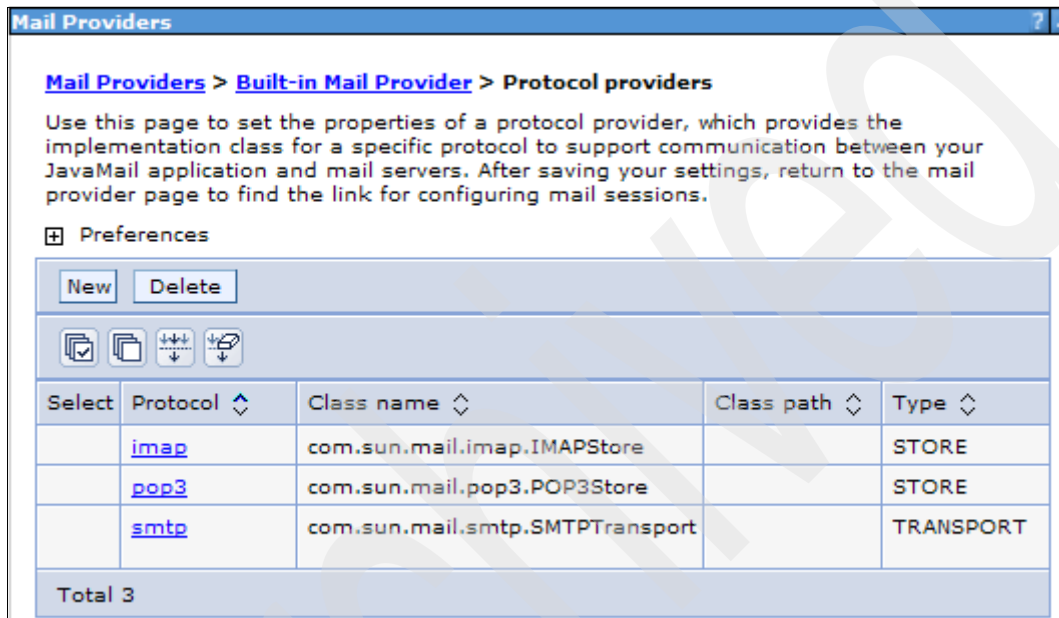


Figure 6-26 Protocol providers

7. Click **OK** and save the configuration.

6.4.3 Configuring JavaMail sessions

To configure JavaMail sessions with a particular mail provider, complete the following steps from the administrative console:

1. Expand **Resources** from the navigation tree.
2. Click **Mail Providers**.
3. Select **Scope** and click **Apply**.
4. Select the mail provider to be used by the JavaMail session.
5. Select **Mail Sessions** in the Additional Properties section. See Figure 6-24 on page 345.

6. Select **New** to create a new mail session object. Figure 6-27 on page 348 shows the configuration page for the PlantsByWebSphere sample application.

Configuration

General Properties

* Scope
cells:was61CellDmgrPrf1

* Provider
Built-in Mail Provider

* Name
[Empty]

* JNDI name
[Empty]

Description
[Text Area]

Category
[Empty]

Mail transport host
[Empty]

Mail transport protocol
smtp

Mail transport user ID
[Empty]

The additional properties will not be available until the general properties for this item are applied or saved.

Additional Properties

Custom properties

Figure 6-27 Configuration page for the mail session

Define the following properties, according to your situation:

- Name

Type an administrative name for the JavaMail session object.

- JNDI name

Use the JavaMail session object name as registered in the application server's name space, including any naming subcontext.

When installing an application that contains modules with JavaMail resource references, the resources defined by the deployment descriptor of the module need to be bound to the real JNDI name of the resources.

As a convention, use the value of the Name property prefixed with mail/, such as mail/<mail_session_name>.

- Mail transport host

This field specifies the server to connect to when sending mail. Use the fully qualified Internet host name of the mail server.

- Mail transport protocol

This field defines the transport protocol to use when sending mail, for example SMTP. Select from the transport protocols defined for the provider.

- Mail transport user ID

This field contains the user ID to provide when connecting to the mail transport host. This setting is not generally used for most mail servers. Leave this field blank unless you use a mail server that requires a user ID and password.

- Mail transport password

Use this field to specify the password to provide when connecting to the mail transport host. Like the user ID, this setting is rarely used by most mail servers. Leave this field blank, unless you use a mail server that requires a user ID and password.

- Enable strict Internet parsing

Check this box to enforce RFC 822 syntax rules for parsing Internet addresses when sending mail.

- Mail from

This value represents the Internet e-mail address that displays as either the From or the Reply-To address. The recipient's reply is sent to this address.

- Mail store host

This field defines the server to which to connect when receiving mail. This setting combines with the mail store user ID and password to represent a valid mail account. For example, if the mail account is itso@itso.ibm.com, then the mail store host is itso.ibm.com.

- Mail store protocol

This field specifies the protocol to be used when receiving mail. It could be IMAP, POP3, or any store protocol for which the user has installed a provider.

- Mail store user ID

This field specifies the user ID to use when connecting to the mail store. This setting combines with the mail store host and password to represent a valid mail account. For example, if the mail account is itso@itso.ibm.com then the user ID is itso.

- Mail store password

This field defines the password to use when connecting to the mail store host. This property combines with the mail store user ID and host to represent a valid mail account.

- Enable debug mode

Use this field to toggle debug mode on and off for this mail session. When true, JavaMail's interaction with mail servers, along with this mail session's properties, will be printed to <stdout>.

7. Click **OK** and save the configuration.

6.4.4 Example code

The code segment shown in Example 6-2 illustrates how an application component sends a message and saves it to the Sent folder.

Example 6-2 JavaMail application code

```
//get JavaMail session

javax.naming.InitialContext ctx = new javax.naming.InitialContext();
javax.mail.Session mail_session = (javax.mail.Session)
ctx.lookup("java:comp/env/mail/MailSession");

//prepare message

MimeMessage msg = new MimeMessage(mail_session);
msg.setRecipients(Message.RecipientType.TO,
InternetAddress.parse("bob@coldmail.net"));
msg.setFrom(new InternetAddress("alice@mail.eedge.com"));
msg.setSubject("Important message from eEdge.com");
msg.setText(msg_text);

//send message

Transport.send(msg);

//save message in "Sent" folder

Store store = mail_session.getStore();
store.connect();
```

```
Folder f = store.getFolder("Sent");  
if (!f.exists()) f.create(Folder.HOLDS_MESSAGES);  
f.appendMessages(new Message[] {msg});
```

6.5 URL providers

A URL provider implements the functionality for a particular URL protocol, such as HTTP, by extending the `java.net.URLStreamHandler` and `java.net.URLConnection` classes. It enables communication between the application and a URL resource that is served by that particular protocol.

A URL provider named Default URL Provider is included in the initial WebSphere configuration. This provider utilizes the URL support provided by the IBM JDK™. Any URL resource with protocols based on the Java 2 Standard Edition 1.3.1, such as HTTP, FTP or File, can use the default URL provider.

You can also plug in your own URL provider for another protocol not supported by the JDK.

6.5.1 Configuring URL providers

URL resource objects are administrative objects used by an application to communicate with an URL. These resource objects are used to read from an URL or to write to an URL. URL resource objects use URL providers for class implementation.

To configure or create a URL provider from the administrative console, do the following:

1. Expand **Resources** from the navigation tree and click **URL Providers**.
2. Select the scope.

3. Click **New** to configure a new URL provider, or select an existing one to edit it.
Figure 6-28 shows the properties for the default URL provider.

The screenshot displays a configuration window for a URL provider. It is divided into two main sections: **General Properties** and **Additional Properties**.

General Properties:

- * Scope:** A text field containing "cells:kadw028Cell01".
- * Name:** A text field containing "Default URL Provider", which is highlighted in yellow.
- * Stream handler class name:** A text field containing "unused".
- * Protocol:** A text field containing "unused".
- Description:** A large text area with up and down arrow buttons on the right side.
- Class path:** A large text area with up and down arrow buttons on the right side.

Additional Properties:

- There are two expandable sections: [URLs](#) and [Custom properties](#), each preceded by a minus sign.

At the bottom of the window, there are four buttons: **Apply**, **OK**, **Reset**, and **Cancel**.

Figure 6-28 URL provider configuration page

Configure the following properties:

- Name

Type an administrative name for the URL provider.

- Class path

Make a list of paths or JAR file names that together form the location for the URL provider classes.

- Stream handler class name

Define the fully qualified name of the Java class that implements the stream handler for the protocol specified by the Protocol property. A stream protocol handler knows how to make a connection for a particular

protocol type, such as HTTP or FTP. It extends the `java.net.URLStreamHandler` class for that particular protocol.

- Protocol

Define the protocol supported by this stream handler, for example, `http` or `ftp`.

4. Click **OK** and save the configuration.

Important: You need to manually install the URL provider (a set of JARs) on each node where the URL provider is going to be used and ensure that it is included in the classpath above.

6.5.2 Configuring URLs

To configure a URL administrative object, do the following from the administrative console:

1. Expand **Resources** from the navigation tree and click **URLs**.
2. Click **New**. See Figure 6-29 on page 353.

The screenshot displays the 'Configuration' tab for defining a new URL. The 'General Properties' section contains the following fields:

- Scope:** cells:was61CellDmgrPrf1
- Provider:** Default URL Provider (with a dropdown arrow and a 'Create New Provider' button)
- Name:** myURL
- JNDI name:** url/myURL
- Specification:** file:///d:/urls/allURLs.txt

On the right side, a note states: 'The additional properties will not be available until the general properties for this item are applied or saved.' Below this note is a section titled 'Additional Properties' with a sub-section for 'Custom properties'.

Figure 6-29 Defining URLs

Use the following properties:

- Name

Define the administrative name for the URL resource object.

- JNDI Name

Type the URL session object name as registered in the application servers name space, including any naming subcontext.

When installing an application that contains modules with URL resource references, the resources defined by the deployment descriptor of the module need to be bound to the real JNDI name of the resources.

As a convention, use the value of the Name property prefixed with url/, such as url/<UrlName>.

- Specification

Type the URL resource to which this URL object is bound.

3. Click **OK** and save the configuration.

6.5.3 URL provider sample

Example 6-3 provides a code sample making use of the URL provider and URL resources. Note that the Web module resource reference, myHttpUrl, is bound to the URL resource JNDI name, url/MotdUrl, during application assembly or deployment.

Example 6-3 HTTP URL provider sample

```
javax.naming.InitialContext ctx = new javax.naming.InitialContext();
javax.naming.Context env =
    (javax.naming.Context) ctx.lookup("java:comp/env");
java.net.URL url = (java.net.URL) env.lookup("myHttpUrl");
java.io.InputStream ins = url.openStream();
int c;
while ((c = ins.read()) != -1) {
    out.write(c);
}
```

In this case, we inserted the Example 6-3 code into a JSP, added the JSP to a Web module, added a URL resource reference to the Web module, and then deployed the Web module. Then we checked that the contents of the file specified in the MotdUrl URL resource, file:///d:/url/motd.txt, were included in the JSP's output.

Similarly, a stock quote custom URL provider could be accessed, as shown in Example 6-4. The Web module resource reference, `myQuoteUrl`, is bound to a URL resource with JNDI name, `url/QuoteUrl`, and URL `quote://IBM`. The custom URL provider will access an online stock quote for IBM.

Example 6-4 Quote URL provider sample

```
javax.naming.InitialContext ctx = new javax.naming.InitialContext();
javax.naming.Context env =
    (javax.naming.Context) ctx.lookup("java:comp/env");
java.net.URL url = (java.net.URL) env.lookup("myQuoteUrl");
out.println("The stock price is "+url.getContent());
```

Note: Each application server's name space is initialized on startup. This means application servers must be restarted to load a modified resource property, such as a URL string.

6.6 Resource environment providers

The `java:comp/env` environment provides a single mechanism by which both JNDI name space objects and local application environment objects can be searched. WebSphere Application Server provides a number of local environment entries by default.

The J2EE 1.4 specification also provides a mechanism for defining custom, non-default, environment entries using `<resource-env-ref>` entries defined in an application's standard deployment descriptors. The specification separates the definition of the resource environment entry from the application by:

- ▶ Requiring the application server to provide a mechanism for defining separate administrative objects that encapsulate a resource environment entry. The administrative objects are accessible through JNDI in the application server's local name space, `java:comp/env`. The specification does not define how an application server should provide this functionality. As a result, the mechanism is generally application-server product-specific.
- ▶ Specifying the administrative object's JNDI lookup name and the expected returned object type in the `<resource-env-ref>`.

Example 6-5 shows a resource environment entry defined in an application's Web module deployment descriptor, web.xml.

Example 6-5 Resource-env-ref in deployment descriptor

```
<web-app>
.....
<resource-env-ref>
  <resource-env-ref-name>myapp/MyLogWriter</resource-env-ref-name>
  <resource-env-ref-type>com.ibm.itso.test.LogWriter</resource-env-ref-type>
</resource-env-ref>
.....
</web-app>
```

Example 6-6 shows how this resource environment entry could be accessed from Java code in the Web module.

Example 6-6 Java code to access resource environment reference

```
import com.ibm.itso.test.*;
.....
InitialContext ctx = new InitialContext();
LogWriter myLog = (LogWriter) ctx.lookup("java:comp/env/myapp/MyLogWriter");
myLog.write(msg);
.....
```

6.6.1 Resource environment references

WebSphere Application Server supports the <resource-env-ref> mechanism by providing resource environment provider administrative objects that are configured using the administration tools. Each <resource-env-ref> requires the creation of the following administered objects in the order shown:

1. Resource environment provider

This provider defines an administrative object that groups together the referenceable, resource environment entry administrative objects and any required custom properties.

The scope you choose determines which resources.xml configuration file is updated to contain the provider's configuration stanza:

```
<resources.env:ResourceEnvironmentProvider
xmi:id="ResourceEnvironmentProvider_1" name="ResProviderName"/>
```

2. Referenceable

This object defines the classname of the factory class that returns object instances implementing a Java interface.

The referenceable's configuration is added to the provider's stanza in the resources.xml file appropriate to the scope, as in Example 6-7.

Example 6-7 Referenceable object

```
<resources.env:ResourceEnvironmentProvider
xmi:id="ResourceEnvironmentProvider_1" name="ResProviderName">
  <referenceables xmi:id="Referenceable_1"
factoryClassname="com.ibm.itso.test.LogWriterFactory"
classname="com.ibm.itso.test.LogWriter"/>
</resources.env:ResourceEnvironmentProvider>
```

3. Resource environment entry

Defines the binding target (JNDI name), factory class, and return object type (via link to the Referenceable) of the resource environment entry.

The referenceable's configuration is added to the provider's stanza in the resources.xml file appropriate to the scope, as in Example 6-8.

Example 6-8 Resource environment entry

```
<resources.env:ResourceEnvironmentProvider
xmi:id="ResourceEnvironmentProvider_1" name="ResProviderName">
  <factories xmi:type="resources.env:ResourceEnvEntry"
xmi:id="ResourceEnvEntry_1" name="MyLogWriter" jndiName="myapp/MyLogWriter"
referenceable="Referenceable_1"/>
  <referenceables xmi:id="Referenceable_1"
factoryClassname="com.ibm.itso.test.LogWriterFactory"
classname="com.ibm.itso.test.LogWriter"/>
</resources.env:ResourceEnvironmentProvider>
```

6.6.2 Configuring the resource environment provider

To create settings for a resource environment provider:

1. Click **Resources** → **Resource Environment Providers** in the navigation tree.
2. Select the scope.
3. Click **New**.

4. Enter a name and description for the new resource environment provider and click **Apply**. See Figure 6-30.

The screenshot shows a web browser window titled 'Resource environment providers'. The breadcrumb navigation is 'Resource environment providers > myLogWriter'. Below this is a descriptive paragraph: 'Use this page to configure a resource environment provider, which encapsulates the referenceables that convert resource environment entry data into resource objects. These objects can then be accessed by applications.' A 'Configuration' tab is selected. The main content area is divided into two sections: 'General Properties' and 'Additional Properties'. In 'General Properties', there are three fields: 'Scope' with the value 'cells:was61CellDmgrPrf1:nodes:NodeAIXST02', 'Name' with the value 'myLogWriter', and a 'Description' text area. In 'Additional Properties', there are three links: 'Referenceables', 'Resource environment entries', and 'Custom properties'. At the bottom of the configuration area are four buttons: 'Apply', 'OK', 'Reset', and 'Cancel'.

Figure 6-30 Creating a resource environment provider

5. Click **Referenceables** in the Additional Properties section.
6. Click **New**. Use this page to set the classname of the factory that will convert information in the name space into a class instance for the type of resource you want. See Figure 6-31.

Resource environment providers

[Resource environment providers](#) > [myLogWriter](#) > [Referenceables](#) > [com.ibm.itso.test.LogWriterFactory](#)

Use this page to configure a new referenceable, which specifies the factory class that converts data in the Java Naming and Directory Interface (JNDI) name space into an object that represents your resource to WebSphere Application Server. After you define the referenceable, return to the provider page to find the link for configuring resource environment entries. The resource environment entry provides the necessary data about your resource in the JNDI name space.

Configuration

General Properties

* Scope
cells:was61CellDmgrPrf1:nodes:NodeAIXST02

* Factory class name
com.ibm.itso.test.LogWriterFactory

* Class name
com.ibm.itso.test.LogWriter

Apply OK Reset Cancel

Figure 6-31 Create a reference

- Factory class name
This field contains a javax.naming.ObjectFactory implementation class name.
 - Class name
This field refers to the Java type that a referenceable provides access to, for binding validation and to create the reference data type string.
7. Click **OK**.
 8. Select the resource environment provider (in the top navigation path) and click **Resource Env Entries** under Additional Properties.

9. Click **New**. See Figure 6-32 on page 360.

The screenshot shows a web-based configuration interface for 'Resource environment providers'. The breadcrumb trail is 'Resource environment providers > myLogWriter > Resource environment entries > myLogWriter'. Below this is a descriptive paragraph: 'Use this page to configure resource environment entries, which are objects that contain information about a resource and represent it in the JNDI name space. Create resource environment entries only after you configure the necessary referenceables.' A 'Configuration' tab is active. The form is divided into two sections: 'General Properties' and 'Additional Properties'. Under 'General Properties', there are fields for 'Scope' (cells:was61CellDmgrPrf1:nodes:NodeAIXST02), 'Provider' (myLogWriter), 'Name' (myLogWriter), and 'JNDI name' (myapp/MyLogWriter). There is also a 'Description' text area and a 'Category' text field. Under 'Additional Properties', there is a 'Custom properties' link and a 'Referenceables' dropdown menu set to 'com.ibm.itso.test.LogWriterFactory'.

Resource environment providers

[Resource environment providers](#) > [myLogWriter](#) > [Resource environment entries](#) > [myLogWriter](#)

Use this page to configure resource environment entries, which are objects that contain information about a resource and represent it in the JNDI name space. Create resource environment entries only after you configure the necessary referenceables.

Configuration

General Properties

* Scope
cells:was61CellDmgrPrf1:nodes:NodeAIXST02

* Provider
myLogWriter

* Name
myLogWriter

* JNDI name
myapp/MyLogWriter

Description

Category

* Referenceables
com.ibm.itso.test.LogWriterFactory

Additional Properties

[Custom properties](#)

Figure 6-32 Creating a resource environment entry

- Name
Type a display name for the resource.
- JNDI name
Type the JNDI name for the resource, including any naming subcontexts.
This name is used as the link between the platform's binding information for resources defined by a module's deployment descriptor and resources bound into JNDI by the platform.

- Referenceable

The referenceable holds the factoryClassname of the factory that will convert information in the name space into a class instance for the type of resource desired, and for the classname of the type to be returned.

10. Click **OK**.

11. Save your configuration.

6.7 Resource authentication

Resources often require you to perform authentication and authorization before an application can access them. You can configure the settings to determine how this is done in a number of ways. This section discusses the configuration settings and how to use them. However, before implementing any security, you should review the information in *WebSphere Application Server V6.1 Security Handbook*, SG24-6316.

The party responsible for the authentication and authorization is determined by the res-auth setting found in the Web and EJB deployment descriptors. There are two possible settings:

- ▶ res-auth=Application: The application, or component, is responsible.
- ▶ res-auth=Container: WebSphere is responsible.

These settings can be configured during application assembly using Rational Application Developer or the Application Server Toolkit in the EJB or Web deployment descriptor. They can also be set or overridden during application installation.

Table 6-1 Authentication settings

Authentication type	Setting at assembly Authorization type	Setting during installation Resource authorization
Application (component) managed: res-auth=Application	Per_Connection_Factory	Per application
WebSphere managed: res-auth=Container	Container	Container

Component-managed authentication

In the case of component-managed authentication, the application component accessing the resource or adapter is responsible for programmatically supplying the credentials. WebSphere can also supply a default component-managed authentication alias if available. After obtaining the connection factory for the

resource from JNDI, the application component creates a connection to the resource using the create method on the connection factory supplying the credentials. If no credentials are supplied when creating a connection and a component-managed authentication alias has been specified on the J2C connection factory, the credentials from the authentication alias will be used. Assuming the credentials are valid, future requests using the same connection will use the same credentials.

The application follows these basic steps:

1. Get the initial JNDI context.
2. Look up the connection factory for the resource adapter.
3. Create a ConnectionSpec object holding credentials.
4. Obtain a connection object from the connection factory by supplying the ConnectionSpec object.

Authentication with WebSphere

Container-managed authentication removes the requirement that the component programmatically supply the credentials for accessing the resource. Instead of calling the getConnection() method with a ConnectionSpec object, getConnection() is called with no arguments. The authentication credentials are then supplied by the Web container, application container, or the EJB container, depending on from where the resource is accessed. WebSphere Application Server V6 supports the JAAS specification, so the credentials can be mapped from any of the configured JAAS authentication login modules, including any custom JAAS authentication login module.

When using container-managed authentication, you have the following options for the authentication method to be used:

- Select **None** if you are using the WebSphere administrative console or **Container Managed Authentication (deprecated)** in the Application Server Toolkit.

This option uses the container-managed authentication settings that are defined for the resource's connection factory. The credentials can come from a JAAS authentication alias when using the DefaultPrincipalMapping Mapping-configuration alias setting, or mapped from another JAAS authentication login module. Any application that can get the resource's connection factory from JNDI will be able to access the EIS. This creates a security exposure where unauthorized applications can gain access to the resource.

Selecting this option and specifying **DefaultPrincipalMapping** and selecting a JAAS authentication alias when defining the resource's connection factory

provides the same functionality as WebSphere Application Server V5. This is no longer the recommended method.

- ▶ Select the **Use default method**.

The Use Default Method setting behaves very similar to container-managed authentication using the DefaultPrincipalMapping option. A JAAS authentication alias is linked to the connection factory and all container-managed authentication requests using the resource reference use the credentials from the alias. The difference is that the linking from the JAAS authentication alias to connection factory is done at the resource reference level within the application. This alleviates a security exposure by limiting the scope of the credentials to the application defining the resource reference. All other applications would need to supply their own credentials when accessing the connection factory directly from JNDI. This is the recommended method for mapping JAAS authentication aliases to connection factories.

- ▶ Select **Use custom login configuration**.

You can also use any WebSphere or user-supplied custom JAAS login configuration.

6.8 More information

These documents and Web sites are also relevant as further information sources:

- ▶ WebSphere Application Server Information Center
<http://www.ibm.com/software/webservers/appserv/infocenter.html>
- ▶ *Java 2 Platform Enterprise Edition Specification, v1.4*
http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf
- ▶ JDBC Technology
<http://java.sun.com/products/jdbc/index.html>
- ▶ Enterprise JavaBeans Technology
<http://java.sun.com/products/ejb/>
- ▶ J2EE Connector Architecture
<http://java.sun.com/j2ee/connector/>
- ▶ JavaMail API Specification
<http://java.sun.com/products/javamail/reference/api/index.html>

Managing Web servers

This chapter describes in detail the system management functionality of the Web server. We cover:

- ▶ Web server support overview
- ▶ Working with Web servers
- ▶ Working with the plug-in configuration file

For information regarding the topology of the Web server installation, refer to *Planning and Designing for WebSphere Application Server V6.1*, SG24-7305.

7.1 Web server support overview

WebSphere Application Server provides Web server plug-ins that work with a Web server to route requests for dynamic content, such as servlets, from the Web server to the proper application server. A Web server plug-in is specific to the type of Web server. It is installed on the Web server machine and configured in the Web server configuration.

A plug-in configuration file generated on the application server and placed on the Web server is used for routing information. In order to manage the generation and propagation of these plug-in configuration files, Web servers are defined to the WebSphere Application Server configuration repository. In some cases, Web server configuration and management features are also available from the WebSphere administrative tools.

The following are the supported Web servers for WebSphere Application Server V6.1:

- ▶ Apache HTTP Server
- ▶ Domino Web Server
- ▶ IBM HTTP Server
- ▶ Microsoft Internet Information Services
- ▶ Sun Java System Web Server (formerly Sun ONE and iPlanet™)

For the latest list of supported Web servers and the versions supported, see the prerequisite document at:

<http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html>

7.1.1 Request routing using the plug-in

The Web server plug-in uses an XML configuration file to determine whether a request is for the Web server or the application server. When a request reaches the Web server, the URL is compared to those managed by the plug-in. If a match is found, the plug-in configuration file contains the information needed to forward that request to the Web container using the Web container inbound transport chain. See Figure 7-1 on page 367.

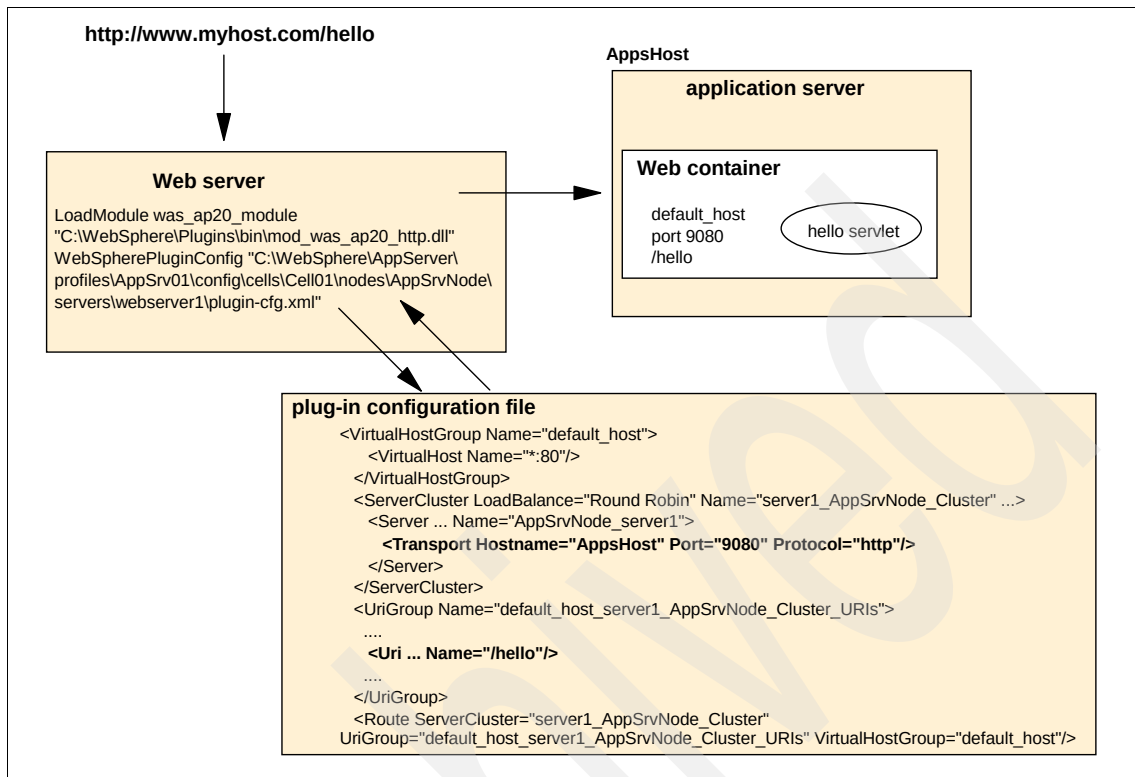


Figure 7-1 Web server plug-in routing

The plug-in configuration file is generated using the WebSphere administrative tools. Each time you make a change to the WebSphere Application Server configuration that would affect how requests are routed from a Web server to the application server, you need to regenerate and propagate the plug-in configuration file to the Web server. You can propagate manually or configure it to be done automatically.

7.1.2 Web server and plug-in management

The setup of your Web server and Web server plug-in environment is defined in a *Web server* definition. The Web server definition includes information about the location of the Web server, its configuration files, and plug-in configuration. During application deployment, Web modules can be mapped to a Web server, ensuring the proper routing information is generated for the plug-in configuration file.

Each Web server is associated with a node, either managed or unmanaged. Web server definitions are located under **Servers** → **Web servers** in the administrative console (see Figure 7-2). The Web server definition is configured as part of the plug-in installation process. Web servers can also be added manually.

Web servers				
Use this page to view a list of the installed Web servers.				
+ Preferences				
Generate Plug-in Propagate Plug-in New Delete Templates... Start				
☐ ☐ ☐ ☐				
Select	Name	Web server Type	Node	Version
☐	ApacheWS1	Apache HTTP Server	UMNode2	Not appli
☐	webserver1	IBM HTTP Server	kadw028Node03	ND 6.1.0
Total 2				

Figure 7-2 Web server definition

Managed Web servers versus unmanaged

When defining Web servers to WebSphere Application Server, it is important to understand the concept of managed versus unmanaged nodes. A supported Web server can be on a managed node or an unmanaged node, depending on the environment on which you are running the Web server.

WebSphere Application Server supports basic administrative functions for all supported Web servers. For example, generation of a plug-in configuration can be performed for all Web servers. If the Web server is defined on a managed node, automatic propagation of the plug-in configuration can be performed using node synchronization. If the Web server is defined on an unmanaged node, automatic propagation of a plug-in configuration is only supported for IBM HTTP Servers.

WebSphere Application Server supports some additional administrative console tasks for IBM HTTP Servers on managed and unmanaged nodes. For example, you can start IBM HTTP Servers, stop them, terminate them, display their log files, and edit their configuration files.

Unmanaged nodes

An *unmanaged node* does not have a node agent to manage its servers. In a stand-alone server environment, you can define one Web server and it, by necessity, resides on an unmanaged node. In a distributed server environment, Web servers defined to an unmanaged node are typically remote Web servers.

If the Web server is defined to an unmanaged node, you can do the following:

1. Check the status of the Web server.
2. Generate a plug-in configuration file for that Web server.

If the Web server is an IBM HTTP Server and the IHS Administration server is installed and properly configured, you can also:

- a. Display the IBM HTTP Server Error log (error.log) and Access log (access.log) files.
- b. Start and stop the server.
- c. Display and edit the IBM HTTP Server configuration file (httpd.conf).
- d. Propagate the plug-in configuration file after it is generated.

You cannot propagate an updated plug-in configuration file to a non-IHS Web server that is defined to an unmanaged node. You must install an updated plug-in configuration file manually to a Web server that is defined to an unmanaged node.

Managed nodes

In a distributed server environment, you can define multiple Web servers. These Web servers can be defined on managed or unmanaged nodes. A *managed node* has a node agent. If the Web server is defined to a managed node, you can do the following:

1. Check the status of the Web server.
2. Generate a plug-in configuration file for that Web server.
3. Propagate the plug-in configuration file after it is generated.

If the Web server is an IBM HTTP Server (IHS) and the IHS Administration server is installed and properly configured, you can also:

- a. Display the IBM HTTP Server Error log (error.log) and Access log (access.log) files.
- b. Start and stop the server.
- c. Display and edit the IBM HTTP Server configuration file (httpd.conf).

How are nodes and servers defined?

During the installation of the plug-in, the Plug-ins installation wizard creates a Web server configuration script named `configure<Web_server_name>`. This configuration script is used to create the Web server definition and, if necessary, the node definition in the configuration of the application server.

If a Web server definition already exists for a stand-alone application server, running the script does not add a new Web server definition. Each stand-alone application server can have only one Web server definition. A distributed server environment, on the other hand, can have multiple Web server definitions. The script creates a new Web server definition unless the Web server name is the same.

The Plug-ins installation wizard stores the script in the `<plug-in_home>/bin` directory on the Web server machine. If the plug-in is installed locally (on the same machine as the application server), the configuration script will be run automatically.

For remote installations, you must copy the script from the Web server machine to the `<was_home>/bin` directory on the application server machine for execution. The script runs against the default profile. If one machine is running under Linux or UNIX and the other machine is running under Windows, use the script created in the `<plug-in_home>/bin/crossPlatformScripts` directory.

Note: Always open a new command window in which to execute the `configure<Web_server_name>` script. There is a potential conflict between a shell environment variable, the `WAS_USER_SCRIPT` variable, and the real default profile. The script always works against the default profile. However, if the `WAS_USER_SCRIPT` environment variable is set, a conflict arises as the script attempts to work on the profile identified by the variable.

If you are federating a stand-alone application server into a cell, any Web server definitions created for a stand-alone application server will be lost when they are federated. If you are creating a distributed server environment this way, wait until after federating your application servers to create Web server definitions.

This chapter will discuss how to administer Web servers and Web server plug-ins using the administration tools. For more information about the installation of Web server plug-ins and how the Web server definitions scripts are generated and executed, see *WebSphere Application Server V6.1: Planning and Design*, SG24-7305.

7.2 Working with Web servers

The introduction of Web server definitions to the WebSphere Application Server administrative tools provides the following administrative features:

- ▶ Define nodes (distributed server environment).
- ▶ Define and modify Web servers.
- ▶ Check the status of a Web server.
- ▶ Start and stop IBM HTTP Servers.
- ▶ Administer IBM HTTP Servers.
- ▶ View or modify the Web server configuration file.
- ▶ Map modules to servers.

Tip: See *Hints and tips for managing IBM HTTP Server using the WebSphere administrative console* in the Information Center for valuable information in troubleshooting problems when managing an IBM HTTP Server.

7.2.1 Defining nodes and Web servers

A managed node is added to the cell as part of the process when you federate an application server profile or custom profile to the cell. An unmanaged node, however, is not created using a profile.

The Web server definition script created by the Plug-ins installation wizard defines an unmanaged node for a Web server and the Web server. However, there might be times when you need to define or update the definitions using the administrative console.

Adding an unmanaged node to the cell

To add an unmanaged node using the administrative console:

1. Select **System Administration** → **Nodes** in the console navigation tree.
2. Click **Add Node**.
3. Select **Unmanaged node**.
4. Click **Next**.
5. Enter the following values in the General Properties page. See Figure 7-3 on page 372:
 - a. Name

Type a logical name for the node. The name must be unique within the cell. A node name usually is identical to the host name for the computer. However, you can make the node name different than the host name.

b. Host name

Enter the host name of the unmanaged node that is added to the configuration.

c. Platform Type

Select the operating system on which the unmanaged node runs. Valid options are:

- Windows
- AIX
- HP-UX
- Solaris
- Linux
- OS/400
- z/OS

Configuration

General Properties

* Name
UMNode2

* Host Name
sys1.ibm.com

* Platform Type
Windows

The additional properties will not be available until the general properties for this item are applied or saved.

Additional Properties

■ Custom Properties

Apply OK Reset Cancel

Figure 7-3 General properties for an unmanaged node

6. Click **OK**. The node is added and the name is displayed in the collection on the Nodes page.

Adding a Web server

Once the node for the Web server has been defined, you can add the Web server definition. To add a Web server definition, do the following:

1. Select **Servers** → **Web servers**.

2. Click **New**. See Figure 7-4.
3. Select the node, enter the server name, and its type.

Create new Web server definition

Use this page to create a new Web server.

→ **Step 1: Select a node for the Web server and select the Web server type**

Step 2: Select a Web server template

Step 3: Enter the properties for the new Web server

Step 4: Confirm new Web server

Select a node for the Web server and select the Web server type

Select a node that corresponds to the Web server you want to add.

Select node
UMNode2

* Server name
webserver3

* Type
IBM HTTP Server

Next Cancel

Figure 7-4 Defining a Web server: Step 1

Click **Next**.

4. Select the template for Web server specification. Initially, this template will be one supplied with WebSphere specific to the Web server type. Once you have defined a Web server, you can make it a template for use the next time. See Figure 7-5 on page 374.

The screenshot shows a dialog box titled "Create new Web server definition". It contains a sidebar with four steps: Step 1 (Select a node for the Web server and select the Web server type), Step 2 (Select a Web server template), Step 3 (Enter the properties for the new Web server), and Step 4 (Confirm new Web server). Step 2 is highlighted with a yellow arrow. The main area is titled "Select a Web server template" and contains the instruction "Select the template that corresponds to the server that you want to create." Below this is a table with columns "Select", "Template Name", "Type", and "Description". The table has one row with a radio button in the "Select" column, "IHS" in the "Template Name" column, "System" in the "Type" column, and "The IHS Web Server Template" in the "Description" column. At the bottom are "Previous", "Next", and "Cancel" buttons.

Create new Web server definition

Use this page to create a new Web server.

Step 1: Select a node for the Web server and select the Web server type

→ Step 2: Select a Web server template

Step 3: Enter the properties for the new Web server

Step 4: Confirm new Web server

Select a Web server template

Select the template that corresponds to the server that you want to create.

Select	Template Name	Type	Description
☒	IHS	System	The IHS Web Server Template

Previous Next Cancel

Figure 7-5 Defining a Web server: Step 2

Click **Next**.

5. Enter the properties for the Web server. See Figure 7-6 on page 375.

Step 1: Select a node for the Web server and select the Web server type

Step 2: Select a Web server template

→ **Step 3: Enter the properties for the new Web server**

Step 4: Confirm new Web server

Enter the properties for the new Web server

Enter the Web server properties.

* Port
80

Web server installation location
C:\Program Files\IBM\HTTPServer

* Service name
IBMHTTPServer6.1

* Plug-in installation location
c:\Program Files\IBM\HTTPServer\Plugin

Application mapping to the Web server
All

Enter the IBM Administration Server properties.

* Port
8008

* Username
Admin

* Password

* Confirm password

☐ Use SSL

Figure 7-6 Defining a Web server: Step 3

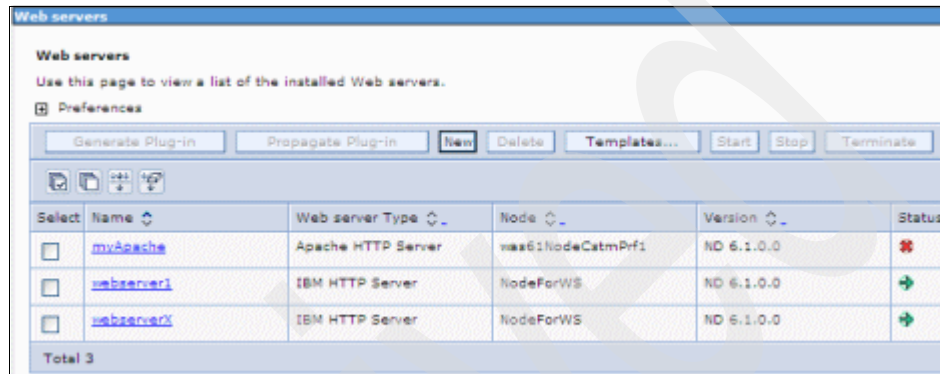
When defining a Web server hosted on a Windows operating system, use the real service name instead of the display name. The service name does not contain spaces. If you do not use the service name, you might have problems starting and stopping the service.

6. Review the options and click **Finish**.

7.2.2 Viewing the status of a Web server

Web server status is reflected in the administrative console. To view Web servers and their status, do the following:

1. Select **Servers** → **Web servers**. If a Web server is started or stopped using a native command, you might need to refresh the view by clicking on the  icon to see the new status. See Figure 7-7.



The screenshot shows the 'Web servers' administrative console. It includes a title bar, a description, a 'Preferences' button, and a toolbar with buttons for 'Generate Plug-in', 'Propagate Plug-in', 'New', 'Delete', 'Template...', 'Start', 'Stop', and 'Terminate'. Below the toolbar is a table with columns: 'Select', 'Name', 'Web server Type', 'Node', 'Version', and 'Status'. The table lists three servers: 'myApache' (Apache HTTP Server, Node: was61NodeCatmPrf1, Version: ND 6.1.0.0, Status: red X), 'webserver1' (IBM HTTP Server, Node: NodeForWS, Version: ND 6.1.0.0, Status: green plus), and 'webserverX' (IBM HTTP Server, Node: NodeForWS, Version: ND 6.1.0.0, Status: green plus). A 'Total 3' summary is at the bottom.

Select	Name	Web server Type	Node	Version	Status
☐	myApache	Apache HTTP Server	was61NodeCatmPrf1	ND 6.1.0.0	
☐	webserver1	IBM HTTP Server	NodeForWS	ND 6.1.0.0	
☐	webserverX	IBM HTTP Server	NodeForWS	ND 6.1.0.0	

Total 3

Figure 7-7 Web server status

WebSphere Application Server reports server status using the Web server host name and port that you have defined. See Figure 7-3 on page 372 and Figure 7-6 on page 375. This is normally port 80. You do not use the remote administration port. If **Use secure protocol** is defined, SSL will be used. See Figure 7-9 on page 379.

7.2.3 Starting and stopping a Web server

A Web server can be started or stopped using one of the following methods.

From the administrative console

You can start or stop the following Web servers from the WebSphere administrative console:

- ▶ All Web servers on a managed node
The node agent will be used to start or stop the Web server.
- ▶ IBM HTTP Server on an unmanaged node
The IBM HTTP Server administration must be up and running on the Web server node.

To start or stop a Web server from the administrative console, do the following:

1. Select **Servers** → **Web servers**. See Figure 7-8 on page 377.
2. Check the box to the left of each Web server you want.
3. Click **Start** or **Stop**.

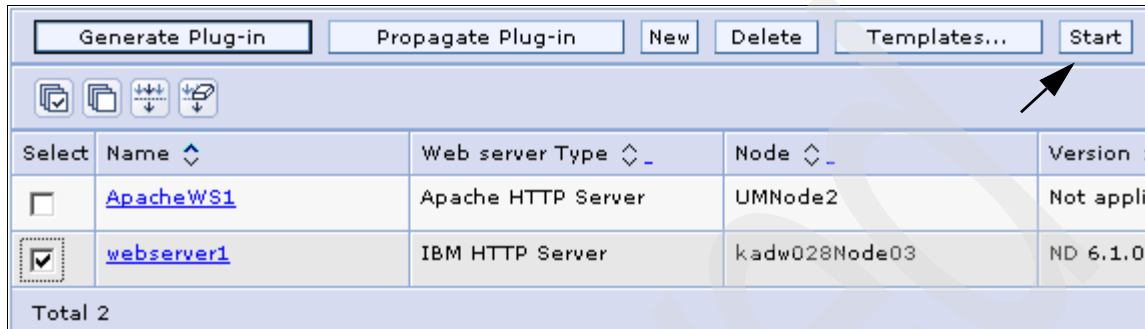


Figure 7-8 Web server definitions

If you have problems starting or stopping an IBM HTTP Server, check the WebSphere console logs (trace) and, if using the IBM HTTP administration server, check the admin_error.log file.

If you have problems starting and stopping IBM HTTP Server on a managed node using the node agent, you can try to start and stop the server by setting up the managed profile and issuing the **startserver <IBM HTTP Server> -nowait -trace** command and check the startServer.log file for the IBM HTTP Server specified.

From a command window

You can also use the native startup or shutdown procedures for the supported Web server. From a command window, change to the directory of your IBM HTTP Server installed image, or to the installed image of a supported Web server.

- ▶ To start or stop the IBM HTTP Server for Linux or UNIX platforms, enter one of the following at a command prompt:
<ihs_install>/bin/apachectl start
<ihs_install>/bin/apachectl stop
- ▶ To start or stop the IBM HTTP Server on Windows platform, select the **IBM HTTP Server 6.1** service from the Services window and invoke the appropriate action.

Note: When the Web server is started or stopped with the native methods, the Web server status on the Web servers page of the administrative console is updated accordingly.

7.2.4 IBM HTTP Server remote administration

You can administer and configure IBM HTTP Server using the WebSphere administrative console. On a managed node, administration is performed using the node agent. This is true of all Web server types. However, unlike other Web servers, administration is possible for an IBM HTTP Server installed on an unmanaged node. In this case, administration is done through the IBM HTTP administration server. This server must be configured and running. Administration is limited to generation and propagation of the plug-in configuration file.

Remote administration setup (unmanaged nodes)

In order for the administrative console to access the IBM HTTP administration server, you must define a valid user ID and password to access the IBM HTTP Server administration server. The user ID and password are stored in the Web server's IHS administration server properties.

You can update your IHS administration server properties in the Web server definition through the Remote Web server management properties page of the administrative console. To set or change these properties, do the following:

1. Click **Servers** → **Web servers**.
2. Select the Web server.
3. Click **Remote Web server management** in the Additional Properties section.
4. Enter the remote Web server management information, as in Figure 7-9 on page 379.

Configuration

Remote Web server management

* Port
8008

* Username
ihsadmin

* Password

Apply OK Reset Cancel

Figure 7-9 IHS remote management properties

- a. Enter the port number for the IHS administration server. The default is 8008.
 - b. Enter a user ID and password that are defined to the IBM HTTP administration server. The IBM HTTP administration server User ID and password are not verified until you attempt to connect.
5. Click **OK** and save the configuration.

Setting the user ID and password in the IBM HTTP administration server:

The IBM HTTP administration server is set, by default, to look at the following file to get the user ID and passwords to use for authentication:

```
<ihs_install>/conf/admin.passwd
```

To initialize this file with a user ID, use the **htpasswd** command. The following example initializes the file with the user ID **webadmin**:

```
C:\IBM HTTP Server\bin>htpasswd "C:\IBM HTTP Server\conf\admin.passwd"
webadmin
```

Automatically using MD5 format.

New password: *****

Re-type new password: *****

Adding password for user webadmin

When you are managing an IBM HTTP Server using the WebSphere administrative console, you must ensure the following conditions are met:

- ▶ Verify that the IBM HTTP Server administration server is running.
- ▶ Verify that the Web server host name and port defined to WebSphere match the IBM HTTP Server administration host name and port.
- ▶ Verify that the firewall is not preventing you from accessing the IBM HTTP Server administration server from the WebSphere administrative console.
- ▶ Verify that the user ID and password specified in the WebSphere administrative console under Remote Web server management is an authorized combination for IBM HTTP Server administration (conf/admin.passwd file).
- ▶ If you are trying to connect securely, verify that you have exported the IBM HTTP Server administration server keydb personal certificate into the WebSphere key database as a signer certificate. This key database will be specified by the com.ibm.ssl.trustStore in the sas.client.props file in which profile your console is running. This is mainly for self-signed certificates.
- ▶ Verify that the IBM HTTP Server admin_error.log file and the WebSphere Application Server logs (trace.log) do not contain any errors.

Hints and tips

The following list describes hints and tips on starting, stopping, and obtaining the status of the IBM HTTP Server using the WebSphere administrative console.

Viewing or modifying the Web server configuration file

The Plug-ins installation wizard automatically configures the Web server configuration file with the information necessary to use the plug-in. For example, among the updates made at the bottom of the httpd.conf file are the lines shown in Example 7-1.

Example 7-1 Plug-in configuration location defined in httpd.conf

```
LoadModule was_ap20_module "C:\opt\WebSphere\Plugins\bin\mod_was_ap20_http.dll"  
WebSpherePluginConfig  
"C:\opt\WebSphere\Plugins\config\webserver1\plugin-cfg.xml"
```

Note that the location the Web server expects to find the plug-in configuration file in is specified in these lines. When you generate the Web server plug-in configuration for this Web server, you will need to propagate or copy the generated file to this location.

The Web server configuration file is a text file and can be modified or viewed manually with a text editor. You can also view or modify this file using the WebSphere Application Server administrative console.

To view or modify the contents of the Web server configuration file in your Web browser:

1. Click **Servers** → **Web servers**.
2. Select the Web server.
3. Click **Configuration File** in the Additional Properties section. See Figure 7-10 on page 381.

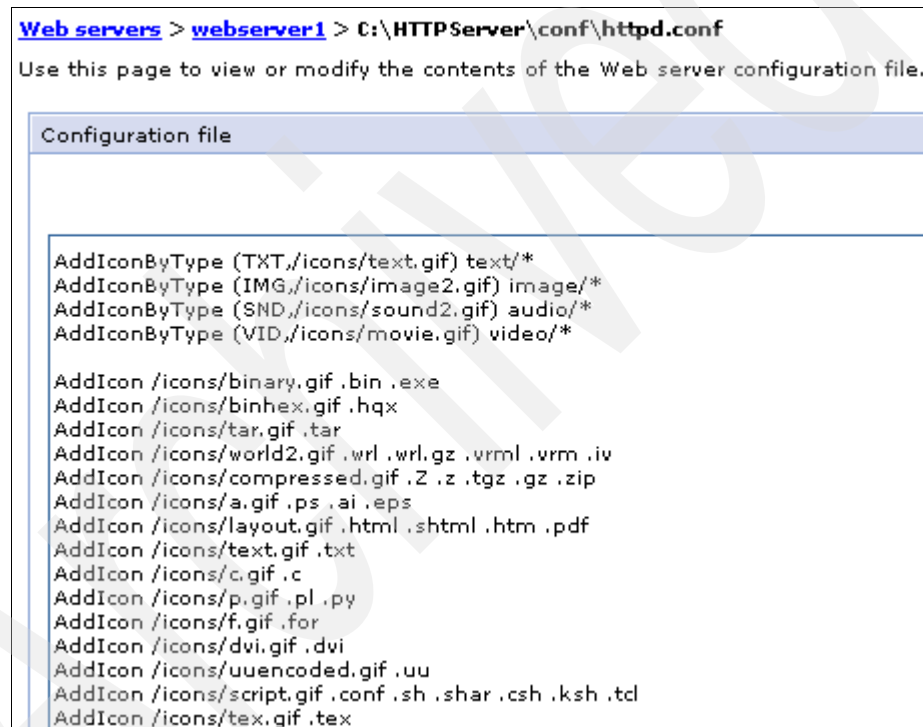


Figure 7-10 IBM HTTP Server configuration file `httpd.conf`

4. Type your changes directly in the window and click **OK**. Save the changes.

Note: If you made changes to the configuration file, you need to restart your Web server for the changes to take effect.

Viewing Web server logs

With remote administration, you can also view the IBM HTTP Server access log and error log. To view the logs, do the following:

1. Click **Servers** → **Web servers**.
2. Select the Web server.
3. Click **Log file** in the Additional Properties section.
4. Select the Runtime tab. See Figure 7-11 on page 382.

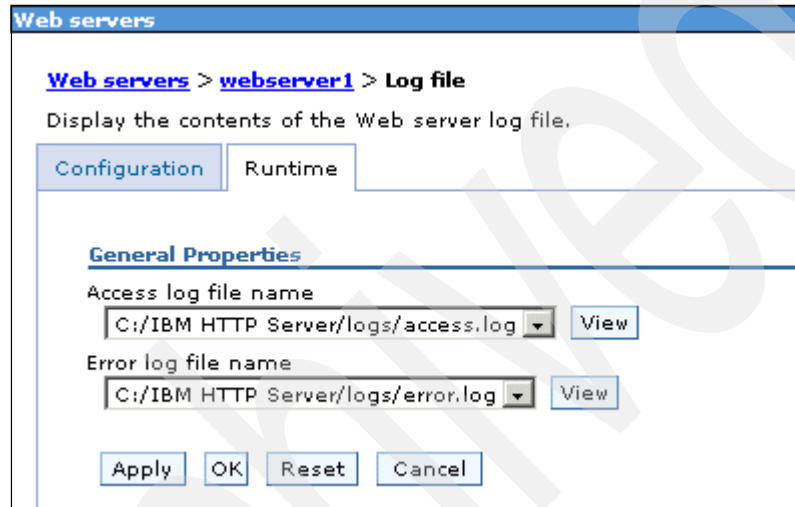


Figure 7-11 Web server Runtime page for logs

5. Click **View** beside the log you want to view. See Figure 7-12.

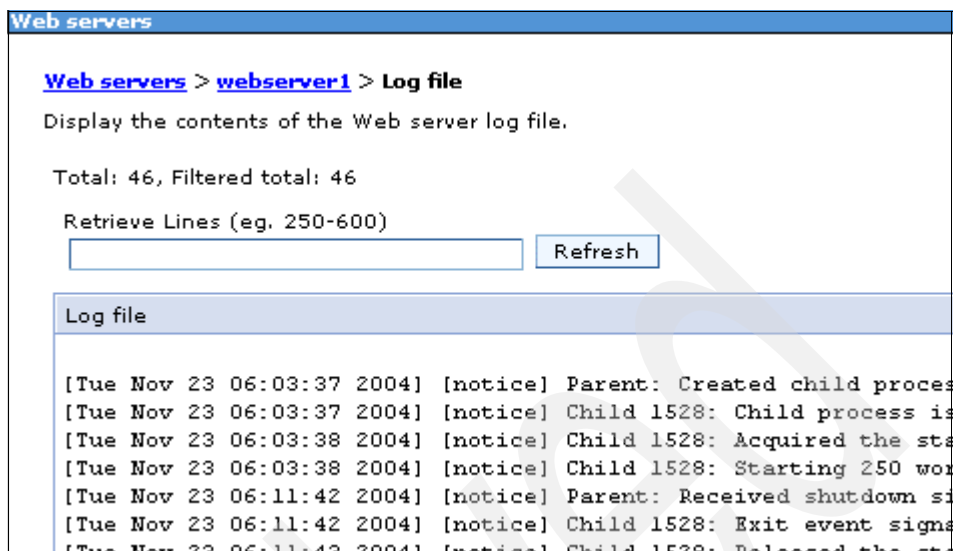


Figure 7-12 Viewing the error log

7.2.5 Mapping modules to servers

Each module of an application is mapped to one or more target servers. The target server can be an application server, cluster of application servers, or Web server. Modules can be installed on the same application server or dispersed among several application servers. Web servers specified as targets will have routing information for the application generated in the plug-in configuration file for the Web server.

This mapping takes place during application deployment. Once an application is deployed, you can view or change these mappings. To check or change the mappings, do the following:

1. Select **Applications** → **Enterprise Applications**.
2. Click the application for which you want to review the mapping.
3. Click **Manage Modules**.

4. Examine the list of mappings. See Figure 7-13.

Clusters and Servers:

WebSphere:cell=ITSOCell,node=wan,server=webserver1
WebSphere:cell=ITSOCell,node=NodeA,server=server1

Apply

Select	Module	URI	Server
☐	Increment Enterprise Java Bean	Increment.jar,META-INF/ejb-jar.xml	WebSphere:cell=ITSOCell,node=wan,server=webserver1 WebSphere:cell=ITSOCell,node=NodeA,server=server1
☐	Default Web Application	DefaultWebApplication.war,WEB-INF/web.xml	WebSphere:cell=ITSOCell,node=wan,server=webserver1 WebSphere:cell=ITSOCell,node=NodeA,server=server1

Figure 7-13 Map modules to selected servers

5. To change a mapping, do the following:
 - a. Select each module that you want mapped to the same targets by placing a check mark in the box to the left of the module.
 - b. From the Clusters and Servers list, select one or more targets. Use the Ctrl key to select multiple targets. For example, to have a Web server serve your application, use the Ctrl key to select an application server and the Web server together.
6. Click **Apply**.
7. Repeat step 5 on page 384 until each module maps to the desired targets.
8. Click **OK** and save your changes.
9. Regenerate and propagate the plug-in configuration, if it is not automatic.

Once you have defined at least one Web server, you must specify a Web server as a deployment target whenever you deploy a Web application. If the Web server plug-in configuration service is enabled, a Web server plug-in's configuration file is automatically regenerated whenever a new application is associated with that Web server.

7.3 Working with the plug-in configuration file

The plug-in configuration file (plugin-cfg.xml) contains routing information for all applications mapped to the Web server. This file is read by a binary plug-in module loaded in the Web server. An example of a binary plug-in module is the mod_ibm_app_server_http.dll file for IBM HTTP Server on the Windows platform.

The binary plug-in module does not change. However, the plug-in configuration file for the binary module needs to be regenerated and propagated to the Web server whenever a change is made to the configuration of applications mapped to the Web server. The binary module reads the XML file to adjust settings and to locate deployed applications for the Web server.

Example 7-2 shows an excerpt from a generated plug-in configuration file.

Example 7-2 An excerpt from the plugin-cfg.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?><!--HTTP server plugin config file
for the webserver ITS0Cell.wan.webserver1 generated on 2004.10.29 at 03:32:12
PM BST-->
<Config ASDisableNagle="false" AcceptAllContent="false"
AppServerPortPreference="HostHeader" ChunkedResponse="false"
IISDisableNagle="false" IISPluginPriority="High" IgnoreDNSFailures="false"
RefreshInterval="60" ResponseChunkSize="64" VHostMatchingCompat="false">
  <Log LogLevel="Error"
Name="c:\opt\WebSphere\Plugins\logs\webserver1\http_plugin.log"/>
  <Property Name="ESIEnable" Value="true"/>
  <Property Name="ESIMaxCacheSize" Value="1024"/>
  <Property Name="ESIInvalidationMonitor" Value="false"/>

  <VirtualHostGroup Name="default_host">
    <VirtualHost Name="*:9080"/>
    <VirtualHost Name="*:80"/>
    <VirtualHost Name="*:9443"/>
  </VirtualHostGroup>

  <ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin"
Name="server1_NodeA_Cluster" PostSizeLimit="-1" RemoveSpecialHeaders="true"
RetryInterval="60">
    <Server ConnectTimeout="0" ExtendedHandshake="false" MaxConnections="-1"
Name="NodeA_server1" WaitForContinue="false">
      <Transport Hostname="wan" Port="9080" Protocol="http"/>
      <Transport Hostname="wan" Port="9443" Protocol="https">
        <Property Name="keyring"
Value="c:\opt\WebSphere\Plugins\etc\plugin-key.kdb"/>
        <Property Name="stashfile"
Value="c:\opt\WebSphere\Plugins\etc\plugin-key.sth"/>
      </Transport>
```

```

        </Server>
    </ServerCluster>

    <UriGroup Name="default_host_server1_NodeA_Cluster_URIs">
        <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionId"
Name="/snoop/*"/>
        <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionId"
Name="/hello"/>

    </UriGroup>
    <Route ServerCluster="server1_NodeA_Cluster"
UriGroup="default_host_server1_NodeA_Cluster_URIs"
VirtualHostGroup="default_host"/>
</Config>

```

The specific values for the UriGroup Name and AffinityCookie attributes depend on how you have assembled your application. When you assemble your application:

- ▶ If you specify **File Serving Enabled**, then only a wildcard URI is generated, regardless of any explicit servlet mappings.
- ▶ If you specify **Serve servlets by class name**, then a URI of the form URI name = `<web_app_uri>/servlet/` is generated.

Both these options apply for both the UriGroup Name and AffinityCookie attributes.

When the plug-in configuration file is generated, it does not include `admin_host` in the list of virtual hosts. See *Allowing Web servers to access the administrative console* in the Information Center for information about how to add it to the list.

7.3.1 Regenerating the plug-in configuration file

The plug-in configuration file needs to be regenerated and propagated to the Web servers when there are changes to your WebSphere configuration that affect how requests are routed from the Web server to the application server. These changes include:

- ▶ Installing an application
- ▶ Creating or changing a virtual host
- ▶ Creating a new server
- ▶ Modifying HTTP transport settings
- ▶ Creating or altering a cluster

The plug-in file can be regenerated manually using the administration tools. You can also set up the plug-in properties of the Web server to enable automatic

generation of the file whenever a relevant configuration change is made. See “Enabling automated plug-in regeneration” on page 391.

To regenerate the plug-in configuration manually, you can either use the administrative console, or you can issue the `GetPluginCfg` command.

Generating the plug-in with administrative console

To generate or regenerate the plug-in configuration file, do the following:

1. Select **Servers** → **Web servers**.
2. Click the box to the left of your Web server.
3. Click **Generate Plug-in**.
4. Verify that the generation was successful by looking at the messages. A success message will be accompanied with the location of the generated plug-in configuration file:

```
<profile_home>/config/cells/<cell_name>/nodes/<web_server_node>/servers  
/<web_server>/plugin-cfg.xml
```

See Figure 7-14.

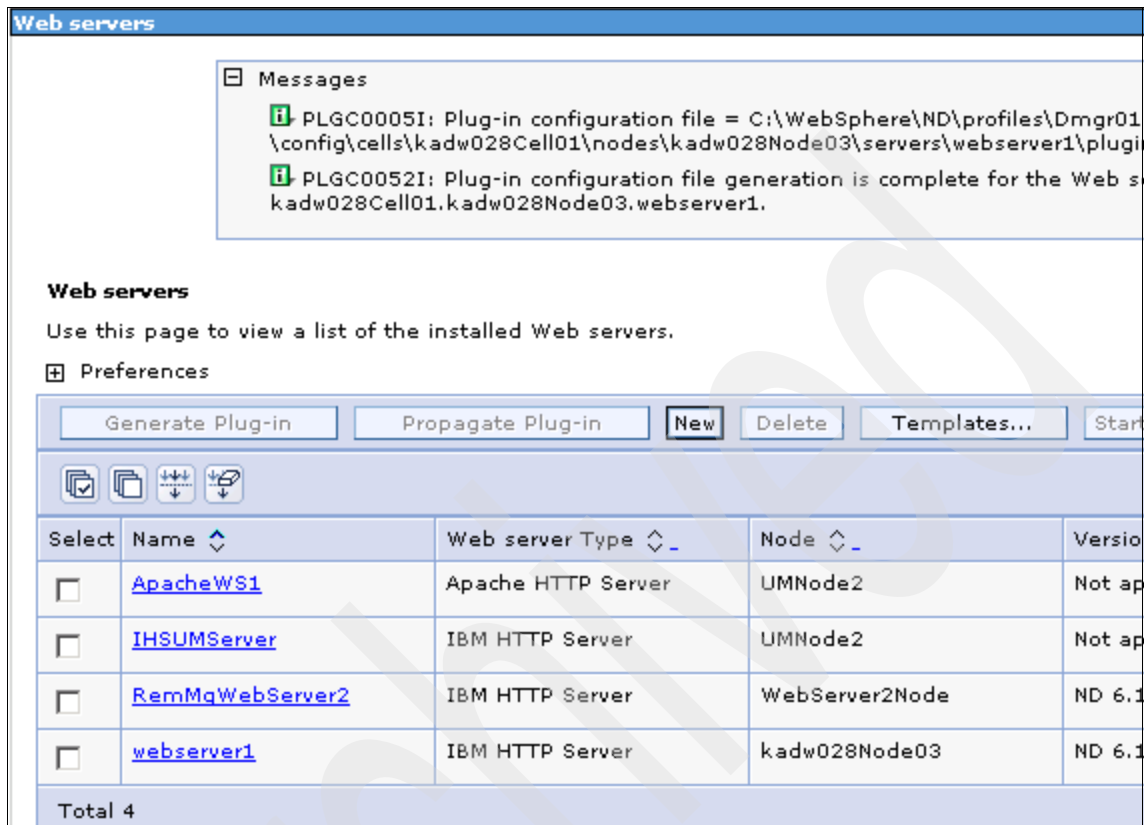


Figure 7-14 Web server definitions

5. You can view the plug-in configuration file by selecting the **View** button next to the Plug-in configuration file name on the Plug-in properties page of your Web server definition. See Figure 7-15. You can also open it with a text editor.

The screenshot shows a web management interface with two tabs: 'Runtime' and 'Configuration'. The 'Configuration' tab is active. Below the tabs is a section titled 'Plug-in properties'. It contains several configuration options:

- An unchecked checkbox labeled 'Ignore DNS failures during Web server startup'.
- A section for 'Refresh configuration interval' with a value of '60' in a text box and the unit 'seconds'.
- A section titled 'Repository copy of Web server plug-in files:' containing:
 - A field for 'Plug-in configuration file name' with the value 'plugin-cfg.xml'. A 'View' button is to its right, which is circled in red.
 - Two checked checkboxes: 'Automatically generate the plug-in configuration file' and 'Automatically propagate plug-in configuration file'.
 - A field for 'Plug-in key store file name' with the value 'plugin-key.kdb'.
 - Two buttons: 'Manage keys and certificates' and 'Copy to Web server key store directory'.
- A section titled 'Web server copy of Web server plug-in files:' which is currently empty.

Figure 7-15 Plug-in properties

To use the new plugin-cfg.xml, file you must propagate it to the Web server system. See 7.3.2, “Propagating the plug-in configuration file” on page 392.

Regenerating the plug-in with the GenPluginCfg command

The GenPluginCfg command is used to regenerate the plug-in configuration file. Depending on the operating platform, the command is:

- ▶ Linux and UNIX: **GenPluginCfg.sh**
- ▶ Windows: **GenPluginCfg.bat**

You can use the -profileName option to define the profile of the Application Server process in a multi-profile installation. The -profileName option is not required for running in a single profile environment. The default for this option is the default profile. For a distributed server environment, the default profile is the deployment manager profile.

Syntax

The **GenPluginCfg** command reads the contents of the configuration repository on the local node to generate the Web server plug-in configuration file.

The syntax of the **GenPluginCfg** command is as follows:

```
:GenPluginCfg.bat(sh) [options]
```

All options are optional. The options are listed in Table 7-1.

Table 7-1 Options for GenPluginCfg

Option	Description
-config.root <config root>	Specify the directory path of the particular configuration repository to be scanned. The default is the value of CONFIG_ROOT defined in the SetupCmdLine.bat(sh) script.
-profileName <profile>	Use this profile to run the command against. If the command is run from <was_home>/bin and -profileName is not specified, the default profile is used. If it is run from <profile_home>/bin, that profile is used.
-cell.name <cell name>	Restrict generation to only the named cell in the configuration repository. The default is the value of WAS_CELL defined in the SetupCmdLine.bat(sh) script.
-node.name <node name>	Restrict generation to only the named node in the particular cell of the configuration repository. The default is the value of WAS_NODE defined in the SetupCmdLine.bat(sh) script.
-webserver.name <webserver1>	Required for creating plug-in configuration file for a given Web server.
-propagate yes/no	This option applies only when the option webserver.name is specified. The default is no.
-cluster.name <cluster_name,cluster_name> ALL	Generate an optional list of clusters. Ignored when the option webserver.name is specified.
-server.name <server_name, server_name>	Generate an optional list of servers. It is required for single server plug-in generation. It is ignored when the option webserver.name is specified.

Option	Description
-output.file.name <filename>	Define the path to the generated plug-in configuration file. The default is <configroot_dir>/plugin-cfg.xml file. It is ignored when the option webserver.name is specified.
-destination.root <root>	Specify the installation root of the machine the configuration is used on. It is ignored when the option webserver.name is specified.
-destination.operating.system windows/unix	Specify the operating system of the machine the configuration is used on. It is ignored when the option webserver.name is specified.
-debug <yes no>	Enable or disable output of debugging messages. The default is no, that is, debug is disabled.
-help or -?	Print command syntax.

Examples

To generate a plug-in configuration for all of the clusters in a cell, type the following:

```
GenPluginCfg -cell.name NetworkDeploymentCell
```

To generate a plug-in configuration for a single server, type:

```
GenPluginCfg -cell.name BaseApplicationServerCell -node.name appServerNode
-server.name appServerName
```

To generate a plug-in configuration file for a Web server, type:

```
GenPluginCfg -cell.name BaseApplicationServerCell -node.name webserverNode
-webserver.name webserverName
```

When this command is issued without the option -webserver.name webserverName, the plug-in configuration file is generated based on topology.

Enabling automated plug-in regeneration

The Web server plug-in configuration service by default regenerates the plugin-cfg.xml file automatically. You can view or change the configuration settings for the Web server plug-in configuration service.

See Example 7-14 on page 388. To view or change the plug-in generation property, do the following:

1. Select **Servers** → **Web servers**.
2. Click your Web server.

3. Select **Plug-in properties** in the Additional Properties section.
4. View or change the **Automatically generate the plug-in configuration file** option.

When selected, the Web server plug-in configuration service automatically generates the plug-in configuration file whenever the Web server environment changes. For example, the plug-in configuration file is regenerated whenever one of the following activities occurs:

- A new application is deployed on an associated application server.
- The Web server definition is saved.
- An application is removed from an associated application server.
- A new virtual host is defined.

Whenever a virtual host definition is updated, the plug-in configuration file is automatically regenerated for all of the Web servers.

7.3.2 Propagating the plug-in configuration file

After a plug-in configuration file is regenerated, it needs to be propagated to the Web server.

The configuration service can automatically propagate the plugin-cfg.xml file to a Web server machine if it is configured on a managed node, and to an IBM HTTP Server if it is configured on an unmanaged node. For other scenarios, you must manually copy the file to the Web server machines.

You can manually propagate the file by copying it from the application server machine to the Web server machine, or you can do it from the administrative console.

From a command window

To copy the file from one machine to another, do the following:

1. Copy the file:

```
<profile_home>/config/cells/<cell_name>/nodes/<web_server_node>/servers  
/<web_server>/plugin-cfg.xml
```

2. Place the copy in this directory on the remote Web server machine:

```
<plugins_home>/config/<web_server>
```

From the administrative console

To propagate the plug-in configuration manually from the administrative console, do the following:

1. Select **Servers** → **Web servers**.
2. Click the box to the left of your Web server.
3. Click **Propagate plug-in**. See Example 7-14 on page 388.
4. Verify that the propagation was successful by looking at the messages.

If you are in doubt, check whether the plug-in configuration file has been propagated to the Web server plug-in location by viewing it.

Activating the new plug-in configuration

The Web server binary plug-in module checks for a new configuration file every 60 seconds. You can wait for the plug-in to find the changes, or you can restart the Web sever to invoke the changes immediately.

Tip: If you encounter problems restarting your Web server, check the `http_plugin.log` file in `<plug-ins_home>/config/<web_server>` for information about what portion of the `plugin-cfg.xml` file contains an error. The log file states the line number on which the error occurred along with other details that might help you diagnose why the Web server did not start.

Enable automated plug-in propagation

The Web server plug-in configuration service by default propagates the `plugin-cfg.xml` file automatically. To view or change the plug-in propagation property, do the following steps. See Example 7-14 on page 388 for further information.

1. Select **Servers** → **Web servers**.
2. Click your Web server.
3. Select **Plug-in properties** in the Additional Properties sub section.
4. View or change the **Automatically propagate plug-in configuration file** option.

7.3.3 Modifying the plug-in request routing options

You can specify the load balancing option that the plug-in uses when sending requests to the various application servers associated with that Web server.

To view or modify the Request routing, do the following:

1. Select **Servers** → **Web Servers**.

2. Click your Web server.
3. Select **Plug-in properties** in the Additional Properties section.
4. Select **Request Routing** in the Additional Properties section. See Figure 7-16.

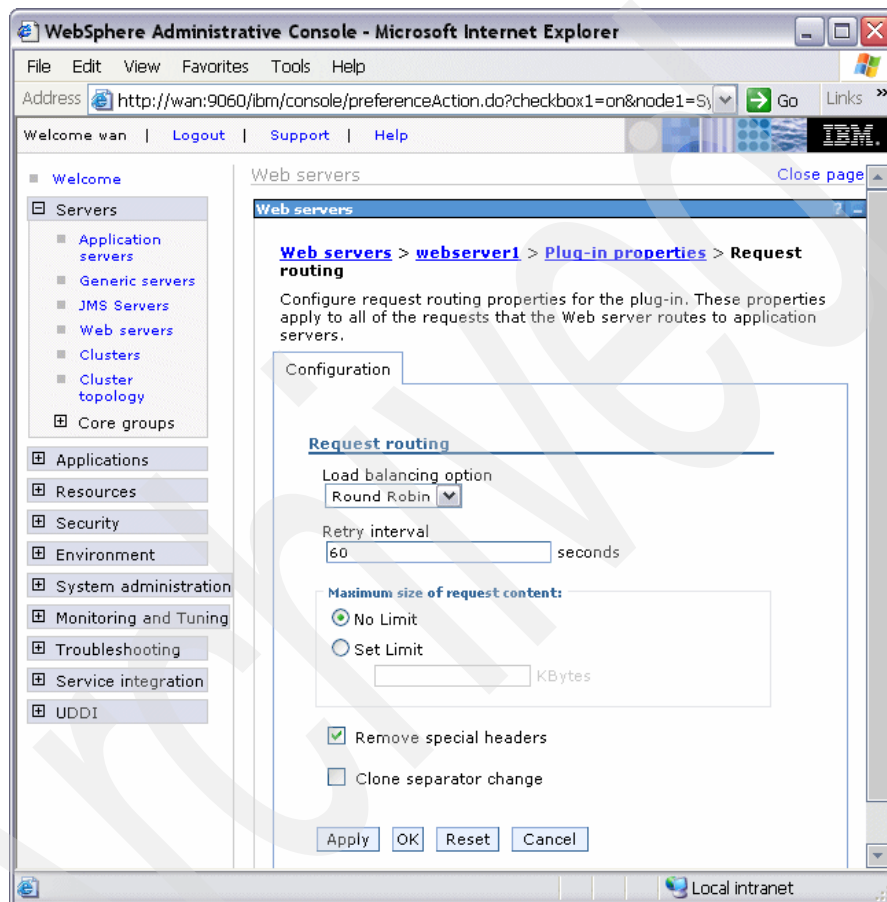


Figure 7-16 Request routing properties

a. Load balancing option

This field corresponds to the LoadBalanceWeight element in the plugin-cfg.xml file. The load balancing options are covered in detail in *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392. The following items are short overviews.

i. Round robin (default)

When using this algorithm, the plug-in selects, at random, a cluster member from which to start. The first successful browser request will be routed to this cluster member and then its weight is decremented by one. New browser requests are then sent round robin to the other application servers and, subsequently, the weight for each application server is decremented by one. The spreading of the load is equal between application servers until one application server reaches a weight of zero. From then on, only application servers without a weight higher than zero will receive routed requests. The only exception to this pattern is when a cluster member is added or restarted.

ii. Random

Requests are passed to cluster members randomly. Weights are not taken into account as in the round robin algorithm. The only time the application servers are not chosen randomly is when there are requests with associated sessions. When the random setting is used, cluster member selection does not take into account where the last request was handled. This means that a new request could be handled by the same cluster member as the last request.

b. Retry interval

The length of time, in seconds, that should elapse from the time an application server is marked down to the time that the plug-in retries a connection.

This field corresponds to the ServerWaitforContinue element in the plugin-cfg.xml file. The default is 60 seconds.

c. Maximum size of request content

Limits the size of request content. If limited, this field also specifies the maximum number of bytes of request content allowed in order for the plug-in to attempt to send the request to an application server.

This field corresponds to the PostSizeLimit element in the plugin-cfg.xml file. When a limit is set, the plug-in fails any request that is received that is greater than the specified limit.

You can set a limit in kilobytes or no limit. The default is set to no limit for the post size.

d. Remove special headers

When enabled, the plug-in will remove any headers from incoming requests before adding the headers the plug-in is supposed to add before forwarding the request to an application server.

This field corresponds to the `RemoveSpecialHeaders` element in the `plugin-cfg.xml` file. The plug-in adds special headers to the request before it is forwarded to the application server. These headers store information about the request that will need to be used by the application. Not removing the headers from incoming requests introduces a potential security exposure.

The default is to remove special headers.

e. Clone separator change

When enabled, the plug-in expects the plus character (+) as the clone separator.

This field corresponds to the `ServerCloneID` element in the `plugin-cfg.xml` file. Some pervasive devices cannot handle the colon character (:) used to separate clone IDs in conjunction with session affinity. If this field is checked, you must also change the configurations of the associated application servers so that the application servers separate clone IDs with the plus character as well.



Part 2

Messaging with WebSphere

This part of this IBM Redbook introduces you to the new service integration technology included with WebSphere Application Server V6. It gives you the basic knowledge you need to configure a run time environment for messaging applications.

This part includes the following chapters:

- ▶ Chapter 8, “Asynchronous messaging” on page 399
- ▶ Chapter 9, “Default messaging provider” on page 539

Asynchronous messaging

In this chapter, we describe the concepts behind the asynchronous messaging functionality provided as part of WebSphere Application Server. We discuss:

- ▶ Messaging concepts
- ▶ Java Message Service
- ▶ Messaging and the J2EE Connector Architecture
- ▶ Message-driven beans
- ▶ Managing WebSphere JMS providers
- ▶ Configuring WebSphere JMS administered objects
- ▶ Connecting to a service integration bus

8.1 Messaging concepts

The term *messaging*, in the generic sense, is usually used to describe the exchange of information between two interested parties. In the context of computer science, messaging can be used to loosely describe a broad range of mechanisms used to communicate data. For example, e-mail and instant messaging are two communication mechanisms that could be described using the term messaging. In both cases, information is exchanged between two parties, but the technology used to achieve the exchange is different.

8.1.1 Loose coupling

These two technologies can also be used to describe one of the main benefits of messaging, that is, *loose coupling*. We discuss two aspects of coupling in the context of messaging applications: process coupling and application coupling.

Process coupling

In the case of Instant Messaging, both parties involved in the exchange of messages need to be available at the point in time when the message is sent. Therefore, from a process point of view, the sending and receiving applications can be said to have *tight coupling*.

In contrast, a user can send an e-mail to a recipient regardless of whether the recipient is currently online. In this case, the sender connects to an intermediary that is able to store the message until the recipient requests it. The sender and receiver processes in this situation can be described as *loosely coupled*. The intermediary in this situation is usually a mail server of some variety, but it can be generically referred to as a *messaging provider*.

Application coupling

As well as enabling loose coupling at the process level, messaging can also enable loose coupling at the application level. In this context, loose coupling means that the sending application is not dependent on any interface exposed by the receiving application. Both applications need only worry about the interface that the messaging provider exposes to enable them to connect and exchange data. With most messaging providers today, these interfaces are reasonably stable and, in some cases, based on open standards. This allows messaging applications to focus on the format of the data that is being exchanged, rather than the interface used to exchange the data. For this reason, messaging applications can be described as *datacentric*.

Contrast this with applications that make use of Enterprise JavaBeans (EJB). EJB client applications need to know about the interface exposed by the EJB. If

this interface changes, then the EJB client application needs to be recompiled to prevent run time errors. For this reason, EJBs and their clients can be described as tightly coupled. Also, due to the dependence on the interface exposed by the EJB, they can also be described as *interface centric* applications.

8.1.2 Messaging types

The terms tight and loose coupling are not commonly used when describing messaging applications. It is more common to refer to the type of messaging that a given application uses. The messaging type describes the style of interaction between the sender and receiver.

The two messaging types are:

- Synchronous messaging

Synchronous messaging involves tightly coupled processes, where the sending and receiving applications communicate directly and both must be available in order for the message exchange to occur.

- Asynchronous messaging

Asynchronous messaging involves loosely coupled processes, where the sending and receiving applications communicate through a messaging provider. The sending application is able to pass the data to the messaging provider and then continue with its processing. The receiving application is able to connect to the messaging provider, possibly at some later point in time, to retrieve the data.

8.1.3 Destinations

With synchronous messaging, because there is no intermediary involved in the exchange of messages, the sending application must know how to connect to the receiving application. Once connected, there is no ambiguity to the intended destination of a message because messages can only be exchanged between the connected parties. This is shown in Figure 8-1.

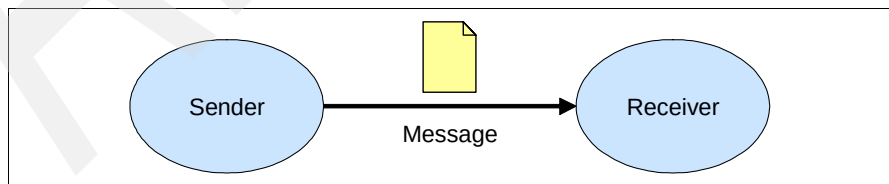


Figure 8-1 Direct communication using synchronous messaging

With asynchronous messaging, however, we need to introduce the concept of a destination. The need for a destination becomes apparent when we consider the fact that a single messaging provider can act as an intermediary for many applications. In this situation, the sending and receiving applications must agree on a single destination used to exchange messages. This destination must be specified when sending a message to the messaging provider, or receiving a message from the messaging provider. This is shown in Figure 8-2.

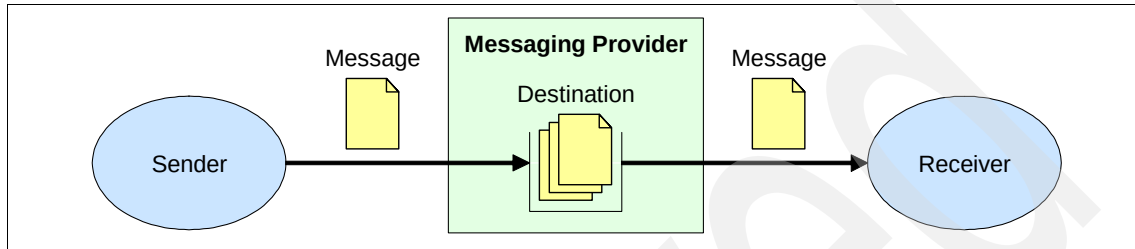


Figure 8-2 Indirect communication via a destination using asynchronous messaging

A sending application might need to exchange different messages with several receiving applications. In this situation, it would be normal for the sending application to use a different destination for each receiving application with which it wants to communicate. This is shown in Figure 8-3.

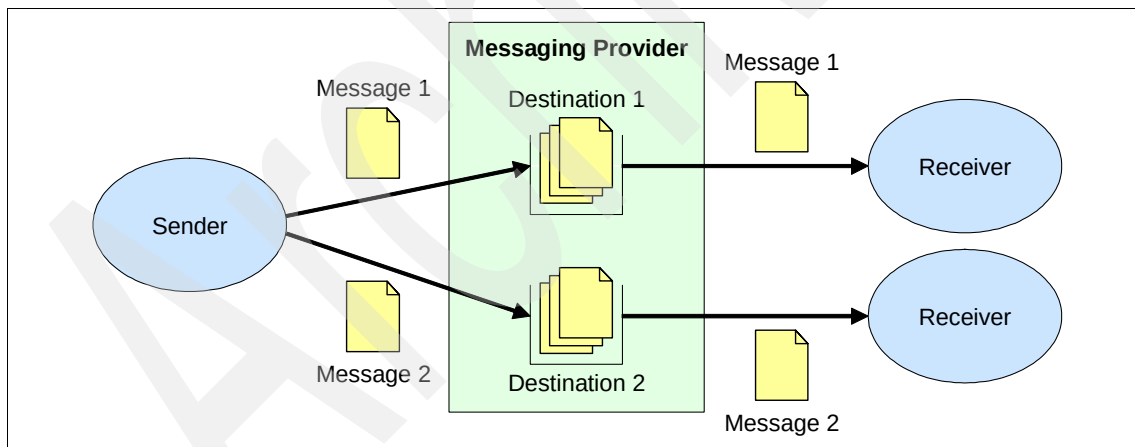


Figure 8-3 Communicating with multiple receivers using asynchronous messaging

8.1.4 Messaging models

As messaging technologies have evolved, two types of asynchronous messaging models have emerged, *Point-to-Point* and *Publish/Subscribe*. These models describe how the messaging provider distributes messages to the target

destination, that is, they describe the cardinalities for the sender-receiver relationship. It is possible for an application to make use of both messaging models. The Point-to-Point and Publish/Subscribe messaging models are described in the following sections.

Point-to-Point

In the Point-to-Point messaging model, the sending application must specify the target destination for the message. In order to receive the message, the receiving application must specify the same destination when it communicates with the messaging provider. This means that there is a one-to-one mapping between the sender and receiver of a message. This is the same situation as depicted in Figure 8-2 on page 402. In the Point-to-Point messaging model, the destination is usually referred to as a *queue*.

Publish/Subscribe

In the Publish/Subscribe messaging model, the sending application publishes messages to a destination. Multiple receiving applications can subscribe to this destination in order to receive a copy of any messages that are published.

When a message arrives at a destination, the messaging provider distributes a copy of the message to all of the receiving applications who have subscribed to the destination. This means that there is potentially a one-to-many relationship between the sender and receiver of a message. However, there might also be no receiving applications subscribed to a destination when a message arrives.

Note that this is not the same situation as depicted in Figure 8-3 on page 402. Figure 8-3 shows a sending application communicating with several receiving applications using the Point-to-Point messaging model with each. Figure 8-4 shows the Publish/Subscribe messaging model.

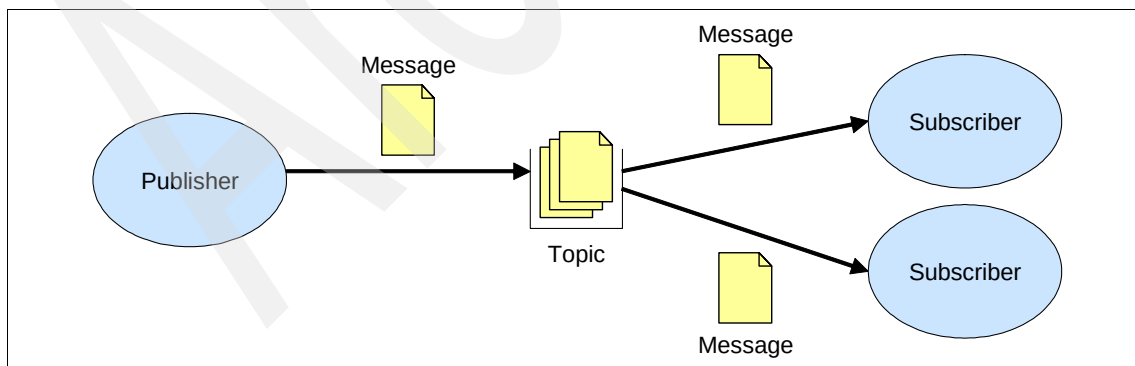


Figure 8-4 Publish/Subscribe messaging model

8.1.5 Messaging patterns

Several patterns also exist that describe the way in which messaging applications connect to, and use, messaging providers. These patterns describe whether a messaging application interacts with the messaging provider as a *message producer*, *message consumer*, or both. When a messaging application acts as both message producer and message consumer, the messaging pattern is referred to as *request-reply*. These messaging patterns are discussed in more detail in the following sections.

Message producers

In the message producer pattern, the sending application simply connects to the messaging provider, sends a message, and then disconnects from the messaging provider. Because the sending application is not interested in what happens to the message once the messaging provider has accepted it, this pattern is sometimes referred to as *fire and forget*, although it is also commonly referred to as *datagram*. The message producer pattern is shown in Figure 8-5.

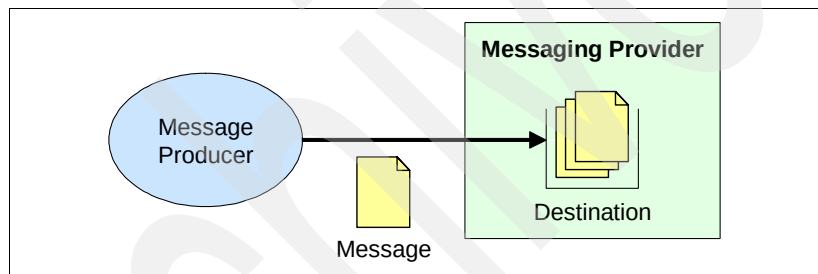


Figure 8-5 Message producer pattern

Message consumers

Message consumers operate in one of two modes:

- Pull mode

In pull mode, the receiving application connects to the messaging provider and explicitly receives a message from the target destination. Obviously, there is no guarantee that a message will be available on the destination at a given point in time, so the receiving application might need to retry at some later stage in order to retrieve a message. For this reason, the receiving application is said to *poll* the destination.

- Push mode

In push mode, it is the messaging provider who initiates the communication with the receiving application when a message arrives at a destination. The

receiving application must register an interest in messages that arrive at the target destination with the messaging provider.

The message consumer pattern is shown in Figure 8-6 on page 405.

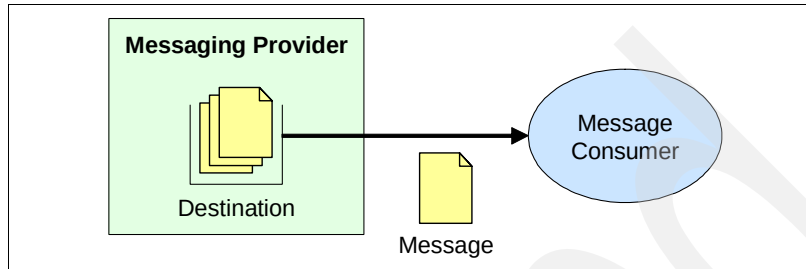


Figure 8-6 Message consumer pattern

Request-reply

The request-reply pattern involves the sending and receiving applications acting as both message producers and message consumers. The sending application initiates the process by sending a message to a destination within the messaging provider and then waiting for a reply. The receiving application receives the message from the messaging provider, performs any required processing, and then sends the reply to the messaging provider. The sending application then receives this response from the messaging provider.

In this situation, the sending and receiving applications are tightly coupled processes, even though they are communicating using asynchronous messaging. For this reason, this pattern is often referred to as *pseudo-synchronous* messaging.

The request-reply pattern is shown in Figure 8-7.

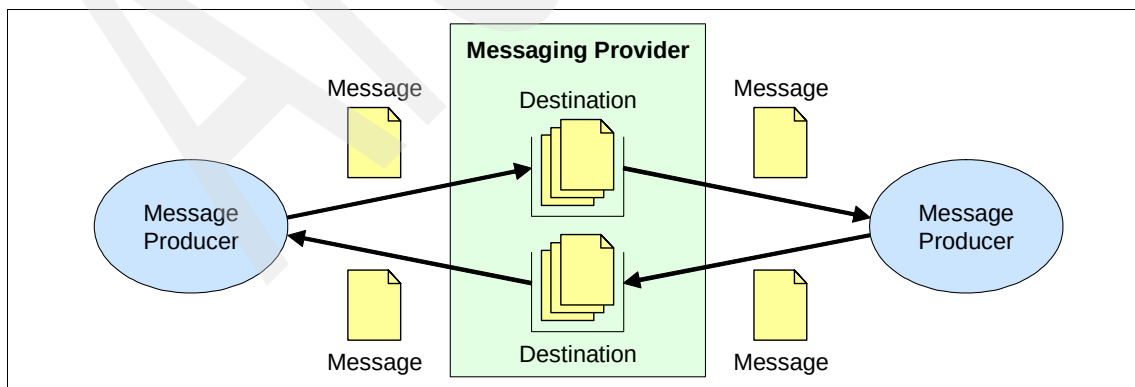


Figure 8-7 Request-reply pattern

8.2 Java Message Service

The Java Message Service (JMS) API is the standard Java API for accessing enterprise messaging systems from Java programs. In other words, it is a standard API that sending and receiving applications written in Java can use to access the messaging provider to create, send, receive, and read messages. We discuss some of the important features of the JMS specification in this section, such as:

- ▶ JMS API history
- ▶ JMS providers
- ▶ JMS domains
- ▶ JMS administered objects
- ▶ JMS and JNDI
- ▶ JMS connections
- ▶ JMS sessions
- ▶ JMS messages
- ▶ JMS message producers
- ▶ JMS consumers
- ▶ JMS exception handling
- ▶ Application Server Facilities
- ▶ JMS and J2EE

For a complete discussion of JMS, refer to the Java Message Service Version 1.1 specification. A link for this specification is contained in 8.8, “References and resources” on page 536.

Note: This section introduces the features of the JMS API, as described in the JMS Version 1.1 specification. The J2EE Version 1.4 specification places certain restrictions on the use of the JMS API within the various J2EE containers. These restrictions are discussed in 8.2.13, “JMS and J2EE” on page 422.

8.2.1 JMS API history

IBM, among others, was involved actively with Sun Microsystems™ in the specification process that led to the original JMS API being published in 1999. Several versions of the API have subsequently been released. The latest is Version 1.1, which includes many changes that resulted from a review of the API by the Java community.

It is important to note that the JMS API defines a vendor-independent programming interface. It does not define how the messaging provider should be implemented or which communication protocol should be used by clients to communicate with the messaging provider. Different vendors can produce

different JMS implementations. They should all be able to run the same JMS applications, but the implementations from different vendors will not necessarily be able to communicate directly with each other.

8.2.2 JMS providers

JMS providers are simply messaging providers that provide a JMS API implementation. However, this does not mean that the underlying messaging provider will be written using the Java programming language. It simply means that the JMS provider written by a specific vendor will be able to communicate with the corresponding messaging provider. As an example, the WebSphere MQ JMS provider knows how to communicate with WebSphere MQ.

8.2.3 JMS domains

The JMS API introduces the concept of *JMS domains*, and defines the point-to-point and publish/subscribe domains. These JMS domains simply represent, in the Java environment, the messaging models described in 8.1.4, “Messaging models” on page 402.

The JMS API also defines a set of domain-specific interfaces that enable client applications to send and receive messages in a given domain. However, Version 1.1 of the JMS specification introduces a set of domain independent interfaces, referred to as the *common interfaces*, in support of a unified messaging model. The domain-specific interfaces have been retained in Version 1.1 of the JMS specification for backwards compatibility.

The preferred approach for implementing JMS client applications is to use the common interfaces. For this reason, the JMS code examples in this chapter all make use of the common interfaces.

Durable subscriptions in the Publish/Subscribe domain

The JMS API also recognizes the need in the Publish/Subscribe domain for topic subscriptions to persist beyond the lifetime of the Java objects that represent them. The JMS API introduces the concept of *durable subscriptions* to address this requirement.

A topic subscriber is said to be *active* when the Java objects that represent them exist. That is, they are active when the JMS client application that they are defined within is executing. When the JMS client application is not executing, a topic subscriber is said to be *inactive*.

A non-durable subscription only lasts as long as the topic subscriber is active. A topic subscriber only receives messages that are published on a topic as long as it is active. When the topic subscriber is inactive, it is no longer subscribed to the topic and, therefore, will not receive any messages published to the topic.

A durable subscription, on the other hand, continues to exist even when the topic subscriber is inactive. If there is no active topic subscriber for a durable subscription, the JMS provider stores any publication messages until they expire. The next time that a topic subscriber for a durable subscription becomes active, the JMS provider delivers any messages that it is storing for the durable subscription. A topic subscriber specifies a unique identity when it creates the durable subscription. Subsequent topic subscribers that specify the same unique identity resume the subscription in the state it was left in by the previous subscriber.

8.2.4 JMS administered objects

Administered objects encapsulate JMS provider-specific configuration information. They are created by an administrator and are later used at run time by JMS clients.

The JMS specification states that the benefits of administered objects are:

- ▶ They hide provider specific configuration details from JMS clients.
- ▶ They abstract JMS administrative information into Java objects that are easily organized and administered from a common management console.

The JMS specification defines two types of administered objects: JMS connection factories and JMS destinations. These are discussed in the following sections.

JMS connection factories

A connection factory encapsulates the configuration information that is required to connect to a specific JMS provider. A JMS client uses a connection factory to create a connection to that JMS provider. ConnectionFactory objects support concurrent use, that is, they can be accessed at the same time by multiple threads within a JMS client application.

The connection factory interfaces defined within the JMS specification are shown in Table 8-1.

Table 8-1 JMS connection factory interfaces

Common interface	Domain-specific interfaces	
	Point-to-Point	Publish/Subscribe
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory

JMS destinations

A destination encapsulates addressing information for a specific JMS provider. A JMS client uses a destination object to address a message to a specific destination on the underlying JMS provider. Destination objects support concurrent use, that is, they can be accessed at the same time by multiple threads within a JMS client application.

The destination interfaces defined within the JMS specification are shown in Table 8-2.

Table 8-2 JMS destination interfaces

Common interface	Domain-specific interfaces	
	Point-to-Point	Publish/Subscribe
Destination	Queue	Topic

8.2.5 JMS and JNDI

At run time, JMS clients need a mechanism by which to obtain references to the configured JMS administered objects. The JMS specification establishes the convention that these references are obtained by looking them up in a name space using the Java Naming and Directory Interface™ (JNDI) API.

The JMS specification does not define a naming policy that indicates where messaging resources should be placed in a name space. However, if the JMS client is a J2EE application, then the J2EE specification does recommend that messaging-related resources be placed in a JMS sub-context.

Administrators require additional tools in order to create and bind the JMS administered objects into the JNDI name space. The JMS specification places the onus of providing these tools on the JMS provider. The tools that are provided for this purpose by WebSphere Application Server are discussed in 8.5, “Managing WebSphere JMS providers” on page 451 and 8.6, “Configuring WebSphere JMS administered objects” on page 461.

J2EE references and JMS

An additional consideration in this discussion is that the JMS client application needs to know where the JMS administered object was placed within the JNDI name space in order to be able to locate it at run time. This requirement creates a dependency between the JMS client code and the run time topology. If the JMS administered object is moved within the JNDI name space, the JMS client application needs to be modified. This is obviously unacceptable.

The J2EE specification provides various naming mechanisms you can use to decouple the JMS client code from the real JNDI names to which the JMS administered objects are bound. For a JMS connection factory, use a Resource Manager Connection Factory Reference. For a JMS destination, use a Resource Environment Reference. These references are defined within the deployment descriptor for a J2EE component. Refer to Chapter 5, “Naming,” of Version 1.4 of the *J2EE Specification* for more information about the definition of these references.

Defining either of these references within a J2EE component results in a JNDI entry being created in the local JNDI name space for that component at run time. You can access this local JNDI name space by the JMS client by performing JNDI lookups with names that begin with `java:comp/env`.

These references are mapped by the administrator to the real JMS-administered objects in the global JNDI name space when the application is deployed to the target operational environment. At run time, when the JMS client performs a lookup in its local JNDI name space, it is redirected to the JMS administered object in the global name space.

Consequently, if a JMS administered object is moved within the JNDI name space, only the mapping for the resource reference needs to be modified. The code for the JMS client application would remain unchanged.

Retrieving administered objects from JNDI

The code required to obtain references to a `ConnectionFactory` and `Destination` object is shown in Example 8-1.

Example 8-1 Using JNDI to retrieve JMS administered objects

```
import javax.jms.*;
import javax.naming.*

// Create the JNDI initial context
InitialContext initCtx = new InitialContext();

// Get the connection factory
ConnectionFactory connFactory
```

```

    = (ConnectionFactory)initCtx.lookup("java:comp/env/jms/myCF");

// Get the destination used to send a message
Destination destination
    = (Destination)initCtx.lookup("java:comp/env/jms/myQueue");

```

8.2.6 JMS Connections

A JMS Connection object represents the connection that a JMS client has to its JMS provider. The JMS specification states that a Connection encapsulates an open connection with a JMS provider and that it typically represents an open TCP/IP socket between a client and a JMS provider. However, this is dependent on the JMS providers implementation.

It is important to note that the creation of a Connection object normally results in resources being allocated within the JMS provider itself, that is, resources are allocated outside of the process running the JMS client. For this reason, care must be taken to close a Connection when it is no longer required within the JMS client application. Invoking the close method on a Connection object results in the close method being called on all of the objects created from it.

The creation of the Connection object is also the point at which the JMS client authenticates itself with the JMS provider. If no credentials are specified, then the identity of the user under which the JMS client is running is used.

Connection objects support concurrent use.

ConnectionFactory objects are used to create instances of Connection objects. The connection interfaces defined within the JMS specification are shown in Table 8-3.

Table 8-3 JMS connection interfaces

Common interface	Domain-specific interfaces	
	Point-to-Point	Publish/Subscribe
Connection	QueueConnection	TopicConnection

The code required to create a `Connection` object is shown in Example 8-2.

Example 8-2 Creating JMS Connections

```
// User credentials
String userID = "jmsClient";
String password = "password";

// Create the connection, specifying no credentials
Connection conn1 = connFactory.createConnection();

// Create connection, specifying credentials
Connection conn2 = connFactory.createConnection(userID, password);
```

8.2.7 JMS sessions

A JMS session is used to create message producers and message consumers for a single JMS provider. It is created from a `Connection` object.

It is also used to define the scope of local transactions. It can group multiple send and receive interactions with the JMS provider into a single unit of work. However, the unit of work only spans the interactions performed by message producers or consumers created from this `Session` object. A transacted session can complete a transaction using the `commit` or `rollback` methods of the `Session` object. Once the current transaction has been completed, a new transaction is automatically started.

`Session` objects do not support concurrent use. They cannot be accessed at the same time by multiple threads within a JMS client application. If a JMS client requires one thread to produce messages while another thread consumes them, the JMS specification recommends that the JMS client uses separate `Sessions` for each thread.

The session interfaces defined within the JMS specification are shown in Table 8-4.

Table 8-4 JMS session interfaces

Common interface	Domain-specific interfaces	
	Point-to-Point	Publish/Subscribe
Session	QueueSession	TopicSession

The code required to create a `Session` object is shown in Example 8-3.


```
// Create a non-transacted session  
Session session = conn1.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

8.2.8 JMS messages

The JMS session acts as factory for JMS messages. The JMS specification defines a logical format for the messages that can be sent to, and received from, JMS providers. Recall that the JMS specification only defines interfaces and not any implementation specifics, so the physical representation of a JMS message is provider-specific.

The elements that make up a JMS message are:

- Headers

All messages support the same set of header fields. Header fields contain values that are used by both clients and providers to identify and route messages.

- Properties

Each message contains a built-in facility to support application-defined property values. Properties provide an efficient mechanism to filter application-defined messages.

- Body

The JMS specification defines several types of message body.

The logical format of a JMS message is shown in Figure 8-8.

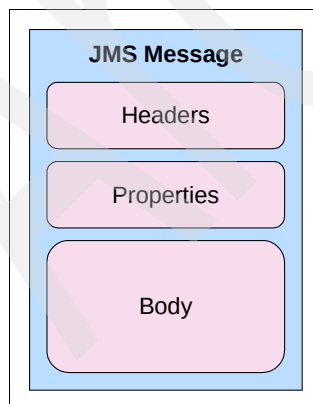


Figure 8-8 Logical format of a JMS message

The JMS specification defines five message interface children. These child interfaces enable various types of data to be placed into the body of the message. The JMS message interfaces are described in Table 8-5.

Table 8-5 JMS message interface types

Message type	Message body
BytesMessage	A stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format.
MapMessage	A set of name-value pairs, where names are strings and values are Java primitive types. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined.
ObjectMessage	A message that contains a serializable Java object.
StreamMessage	A stream of Java primitive values. It is filled and read sequentially.
TextMessage	A message containing a java.lang.String.

Message selectors

A JMS message selector allows a JMS client to filter the messages on a destination so that it only receives the messages that it is interested in. It must be a string whose syntax is based on a subset of the SQL92 conditional expression syntax. However, the message selector expression might only reference JMS message headers and properties, not values that might be part of the message body. An example of a message selector is shown in Example 8-4.

Example 8-4 Sample message selector

```
JMSType='car' AND color='blue' AND weight>2500
```

If a message consumer specifies a message selector when receiving a message from a destination, only messages whose headers and properties match the selector are delivered. If the destination in question is a JMS queue, the message remains on the queue. If the destination in question is a topic, the message is never delivered to the subscriber (from the subscribers perspective, the message does not exist).

For a full description of message selectors and their syntax, please refer to the JMS specification. A link for this specification is contained in 8.8, “References and resources” on page 536.

8.2.9 JMS message producers

The JMS session also acts as a factory for JMS message producers. A JMS message producer is used to send messages to a specific destination on the JMS provider. A JMS message producer does not support concurrent use.

The target destination is specified when creating the message producer. However, it is possible to pass a value of null when creating the message producer. When using a message producer created in this manner, the target destination must be specified on every invocation of the send method.

The message producer can also be used to specify certain properties of messages that it sends, such as delivery mode, priority, and time-to-live.

The message producer interfaces defined within the JMS specification are shown in Table 8-6.

Table 8-6 JMS MessageProducer interfaces

Common interface	Domain-specific interfaces	
	Point-to-Point	Publish/Subscribe
MessageProducer	QueueSender	TopicPublisher

The code required to create and send a message is shown in Example 8-5.

Example 8-5 Creating and sending a JMS message

```
// Create the message producer
MessageProducer msgProducer = session.createProducer(destination);

// Create the message
TextMessage txtMsg = session.createTextMessage("Hello World");

// Send the message
msgProducer.send(txtMsg);
```

8.2.10 JMS message consumers

The JMS session also acts as a factory for JMS message consumers. A JMS client uses a message consumer to receive messages from a destination on the JMS provider. A JMS message consumer does not support concurrent use.

The message consumer interfaces defined within the JMS specification are shown in Table 8-7.

Table 8-7 JMS MessageConsumer Interfaces

Common interface	Domain-specific interfaces	
	Point-to-Point	Publish/Subscribe
MessageConsumer	QueueReceiver	TopicSubscriber

Recall from the discussion in “Message consumers” on page 404 that message consumers can operate in pull mode or push mode. The JMS specification defines message consumers for both of these modes. The message consumers for these are modes are discussed in the following sections.

Pull mode

A JMS client operates in pull mode simply by invoking one of the receive methods on the MessageConsumer object. The MessageConsumer interface exposes a variety of receive methods that allow a client to poll the destination or wait for the next message to arrive.

The code required to receive a message using pull mode is shown in Example 8-6.

Example 8-6 Receiving a JMS message using pull mode

```
// Create the message consumer
MessageConsumer msgConsumer = session.createConsumer(destination);

// Start the connection
conn1.start();

// Attempt to receive a message
Message msg = msgConsumer.receiveNowait();

// Make sure that we have a text message
if (msg instanceof TextMessage)
{
    // Cast the message to the correct type
    TextMessage txtMsg = (TextMessage)msg;

    // Print the contents of the message
    System.out.println(txtMsg.getText());
}
```

Note: The start method must be invoked on the Connection object prior to attempting to receive a message. A connection does not need to be started in order to send messages, only to receive them. This enables the application to complete all of the required configuration steps before attempting to receive a message.

Push mode

In order to implement a solution that uses push mode, the JMS client must register an object that implements the `javax.jms.MessageListener` interface with the `MessageConsumer`. With a message listener instance registered, the JMS provider delivers messages as they arrive by invoking the listener's `onMessage` method.

The `javax.jms.MessageListener` interface is shown in Example 8-7 on page 417.

Example 8-7 The `javax.jms.MessageListener` interface

```
package javax.jms;

public interface MessageListener
{
    public void onMessage(Message message);
}
```

A simple class that implements the `javax.jms.MessageListener` interface is shown in Example 8-8.

Example 8-8 Simple MessageListener implementation

```
package com.ibm.itso.jms;

import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;

public class SimpleListener implements MessageListener
{
    public void onMessage(Message msg)
    {
        // Make sure that we have a text message
        if (msg instanceof TextMessage)
        {
            // Cast the message to the correct type
            TextMessage txtMsg = (TextMessage)msg;

            try
            {
                // Print the contents of the message
                System.out.println(txtMsg.getText());
            }
            catch (JMSException e)
            {
                e.printStackTrace();
            }
        }
    }
}
```

An instance of the message listener can now be registered with the JMS message consumer by the JMS client application. Once the listener is registered, the connection needs to be started in order for messages to be delivered to the message listener. The code required to register a message listener with a JMS message consumer is shown in Example 8-9.

Example 8-9 Receiving a JMS message using push mode

```
import com.ibm.itso.jms.SimpleListener;

// Create the message consumer
MessageConsumer msgConsumer = session.createConsumer(destination);

// Create an instance of the message listener
SimpleListener listener = new SimpleListener();

// Register the message listener with the consumer
msgConsumer.setMessageListener(listener);

// Start the connection
conn1.start();
```

Note: In the JMS Point-to-Point domain, messages remain on a destination until they are either received by a message consumer, or they expire. In the JMS Publish/Subscribe domain, messages remain on a destination until they have been delivered to all of the registered subscribers for the destination or they expire. In order for a message to be retained when a subscribing application is not available, the subscribing application must create a durable subscription. Please refer to “Durable subscriptions in the Publish/Subscribe domain” on page 407 for more information.

8.2.11 JMS exception handling

Any run time errors in a JMS application results in a thrown `javax.jms.JMSEException`. The `JMSEException` class is the root class of all JMS API exceptions.

A `JMSEException` contains the following information:

- ▶ A provider-specific string describing the error
- ▶ A provider-specific string error code
- ▶ A reference to another exception

The `JMSEException` is usually caused by another exception being thrown in the underlying JMS provider. The `JMSEException` class allows JMS client applications to access the initial exception using the `getLinkedException` method. The linked exception can then be used to determine the root cause of the problem in the JMS provider.

The implementation of `JMSEException` does not include the embedded exception in the output of its `toString` method. Therefore, it is necessary to check explicitly for an embedded exception and print it out, as shown in Example 8-10.

Example 8-10 Handling a `javax.jms.JMSEException`

```
try
{
    // Code which may throw a JMSEException
}
catch (JMSEException exception)
{
    System.err.println("Exception caught: " + exception);
    Exception linkedException = exception.getLinkedException();
    if (linkedException != null)
    {
        System.err.println("Linked exception: " + linkedException);
    }
}
```

However, when using a message listener to receive messages asynchronously, the application code cannot catch exceptions raised by failures to receive messages. This is because the application code does not make explicit calls to the receive methods on the message consumer.

The JMS API provides the `javax.jms.ExceptionListener` interface to solve this problem. An exception listener allows a client to be notified of a problem asynchronously. The JMS client must register an object that implements this interface with the connection using the `setExceptionListener` method. With an exception listener instance registered, the JMS provider invokes its `onException` method to notify it that a problem has occurred.

The `javax.jms.ExceptionListener` interface is shown in Example 8-11.

Example 8-11 The `javax.jms.ExceptionListener` interface

```
package javax.jms;

public interface ExceptionListener
{
    public void onException(JMSEException exception);
}
```

A simple class that implements the `javax.jms.ExceptionListener` interface is shown in Example 8-12.


```
package com.ibm.itso.jms;

import javax.jms.ExceptionListener;
import javax.jms.JMSException;

public class SimpleExceptionListener implements ExceptionListener
{
    public void onException(JMSException exception)
    {
        System.err.println("Exception caught: " + exception);
        Exception linkedException = exception.getLinkedException();
        if (linkedException != null)
        {
            System.err.println("Linked exception: " + linkedException);
        }
    }
}
```

8.2.12 Application Server Facilities

The JMS specification defines a number of optional facilities that are intended to be implemented by JMS providers and application server vendors. These facilities extend the functionality of JMS when the JMS client is executing within the context of a J2EE container. The Application Server Facilities are concerned with two main areas of functionality, concurrent message processing and distributed transactions, and these are briefly described in the following sections.

Concurrent message consumers

Recall that Session and MessageConsumer objects do not support being accessed from multiple threads concurrently. Such a restriction would be a huge obstacle to implementing JMS applications within an application server environment, where performance and resource usage are key concerns. The Application Server Facilities define a mechanism that allows an application server to create MessageConsumers that can concurrently process multiple incoming messages.

Distributed transactions

The JMS specification states that it does require a JMS provider to support distributed transactions. However, it also states that if a provider supplies this support, it should be done in the JTA XAResource API. The Application Server Facilities define the interfaces that an application server should implement in order to correctly provide support for distributed transactions.

8.2.13 JMS and J2EE

The JMS API was first included in Version 1.2 of the J2EE specification. This specification required that the JMS API definitions be included in a J2EE product, but that the platform was not required to include an implementation of the JMS `ConnectionFactory` and `Destination` objects.

Subsequent versions of the J2EE specification have placed further requirements on application server vendors. WebSphere Application Server V6 is fully compliant with Version 1.4 of the J2EE specification. See 6.6, “Java Message Service (JMS) 1.1 Requirements”, of the *J2EE Specification V1.4* for information related to these requirements. The *J2EE Specification V1.4* can be downloaded from the following Web site:

<http://java.sun.com/j2ee/index.jsp>

WebSphere Application Server V6 also provides full support for the Application Server Facilities described in 8.2.12, “Application Server Facilities” on page 421.

8.3 Messaging in the J2EE Connector Architecture

Prior to J2EE Version 1.3, there was no architecture that specified the interface between an application server and providers implementing an Enterprise Information System (EIS). Consequently, application server and EIS vendors used vendor-specific architectures to provide EIS integration. This meant that, for each application server that an EIS vendor wanted to support, it needed to provide a specific resource adapter, and, for every resource adapter that an application server vendor wanted to support, it needed to extend the application server.

J2EE Version 1.3 required application servers to support Version 1.0 of the J2EE Connector Architecture (JCA). The J2EE Connector Architecture defines a standard for connecting a compliant application server to an EIS. It defines a standard set of system-level contracts between the J2EE application server and a resource adapter.

As a result, application servers only need to be extended once to add support for all J2EE Connector Architecture compliant resource adapters. Conversely, EIS vendors only need to implement one J2EE Connector Architecture compliant resource adapter, which can then be installed on any compliant application server.

The system contracts defined by Version 1.0 of the J2EE Connector Architecture are described by the specification as follows:

- ▶ Connection management

Connection management enables an application server to pool connections to the underlying EIS and enables application components to connect to an EIS. This leads to a scalable application environment that can support a large number of clients requiring access to an EIS.

- ▶ Transaction management

Transaction management enables an application server to use a transaction manager to manage transactions across multiple resource managers. This contract also supports transactions that are managed internal to an EIS resource manager without the necessity of involving an external transaction manager.

- ▶ Security management

Security management provides support for a secure application environment that reduces security threats to the EIS and protects valuable information resources managed by the EIS.

While Version 1.0 of the J2EE Connector Architecture addressed the main requirements of both application server and EIS vendors, it left some issues unresolved. As a result, Version 1.5 of the specification was produced and it is this version that application servers are now required to support by Version 1.4 of the J2EE specification.

The additional system contracts defined by Version 1.5 of the J2EE Connector Architecture are described by the specification as follows:

- ▶ Life cycle management

Life cycle management enables an application server to manage the life cycle of a resource adapter. This contract provides a mechanism for the application server to bootstrap a resource adapter instance during its deployment or application server startup, and to notify the resource adapter instance during its undeployment or during an orderly shutdown of the application server.

- ▶ Work management

Work management enables a resource adapter to do work (monitor network endpoints, call application components, and so on) by submitting work instances to an application server for execution. The application server dispatches threads to execute submitted work instances. This allows a resource adapter to avoid creating or managing threads directly, and allows an application server to efficiently pool threads and have more control over its run time environment. The resource adapter can control the transaction context with which work instances are executed.

► Transaction inflow management

Transaction inflow management enables a resource adapter to propagate an imported transaction to an application server. This contract also allows a resource adapter to transmit transaction completion and crash recovery calls initiated by an EIS, and ensures that the Atomicity, Consistency, Isolation and Durability (ACID) properties of the imported transaction are preserved.

► Message inflow management

Message inflow management enables a resource adapter to asynchronously deliver messages to message endpoints residing in the application server independent of the specific messaging style, messaging semantics, and messaging infrastructure used to deliver messages. This contract also serves as the standard message provider pluggability contract that allows a wide range of message providers (Java Message Service (JMS), Java API for XML Messaging (JAXM), and so on) to be plugged into any J2EE compatible application server with a resource adapter.

Note: For a full description of all of the system contracts listed above, please refer to the J2EE Connector Architecture Version 1.5 specification. A link for this specification is included in 8.8, “References and resources” on page 536.

In the context of asynchronous messaging, we are interested in the connection management and message inflow system contracts. These system contracts provide for both inbound and outbound communication from a messaging client, to a messaging provider. This is shown in Figure 8-9 on page 424.

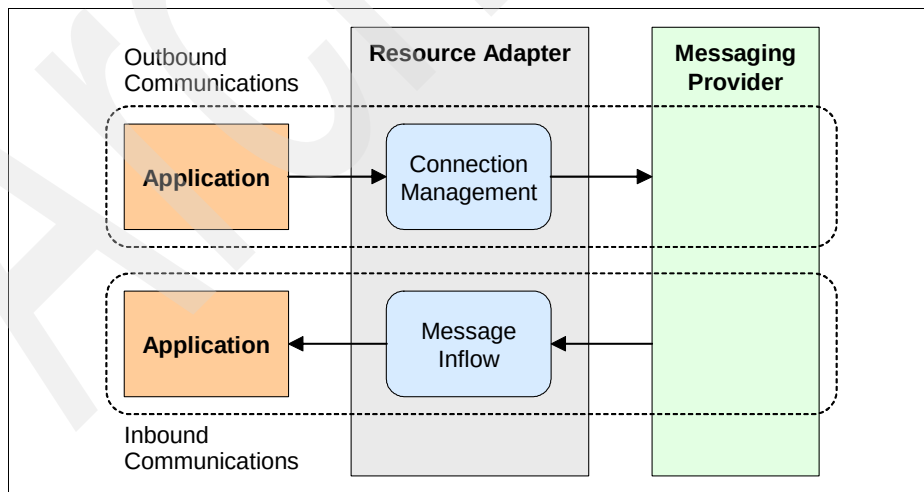


Figure 8-9 Inbound and outbound communication using a resource adapter

Because the connection management system contract was introduced in Version 1.0 of the J2EE Connector Architecture, we will not discuss it further here. Refer to the J2EE Connector Architecture Version 1.5 specification for more information regarding the connection management system contract.

The sections that follow discuss the following aspects of the message inflow system contract:

- ▶ Message endpoints
- ▶ Resource adapters
- ▶ JMS ActivationSpec JavaBean
- ▶ Administered objects

8.3.1 Message endpoints

The message inflow system contract makes use of the message-driven bean (MDB) programming model to asynchronously deliver messages from an EIS into a running application server. A message endpoint is simply a message-driven bean application that is running inside a J2EE application server. It asynchronously consumes messages from a message provider.

An application server compliant with J2EE Version 1.4 is required to support Version 2.1 of the Enterprise JavaBeans specification. This version of the EJB specification has defined additional elements for the message-driven bean deployment descriptor to support the message inflow system contract of the J2EE Connector Architecture. These deployment descriptor elements are discussed in more detail in 8.4.6, “Message-driven bean activation configuration properties” on page 443.

8.3.2 MessageEndpointFactory

The J2EE Connector Architecture requires application server vendors to provide a `MessageEndpointFactory` implementation. A `MessageEndpointFactory` is used by the resource adapter to obtain references to message endpoint instances in order to process messages. In other words, the resource adapter uses the `MessageEndpointFactory` to obtain references to message-driven beans. Multiple message endpoint instances can be created for a single message endpoint, enabling messages to be processed concurrently.

8.3.3 Resource adapters

A resource adapter is the component that maps the proprietary API exposed by the EIS to the API defined by the JCA or some other architecture, JDBC or JMS, for example. Resource adapters are also commonly referred to as *connectors*.

The resource adapter itself runs in the same process as the application server and is responsible for delivering messages to the message endpoints hosted by the application server.

Resource adapter packaging

A resource adapter typically is provided by the messaging provider or a third party and comes packaged in a Resource Adapter Archive (RAR) file. This RAR must be packaged using the Java Archive (JAR) file format and can contain:

- ▶ Any utility classes
- ▶ Native libraries required for any platform dependencies
- ▶ Documentation
- ▶ A deployment descriptor
- ▶ Java classes that implement the J2EE Connector Architecture contracts and any other functionality of the adapter

The only element of the RAR file that is required is the deployment descriptor. This must called ra.xml and must be placed in the META-INF subdirectory of the RAR file.

The resource adapter is installed normally on the application server so that it is available to several J2EE applications at run time. However, it is possible to package the resource adapter within the message endpoint application.

WebSphere Application Server provides a pre-configured resource adapter for the default messaging JMS provider. The RAR file for this resource adapter is called sib.api.jmsra.rar and is located in the \lib\ subdirectory of the WebSphere installation directory.

Resource adapter deployment descriptor

The resource adapter deployment descriptor contains several pieces of information that are used by the application server and the resource adapter at run time, such as:

- ▶ Supported message listener types

The resource adapter lists the types of message listener that it supports. The J2EE Connector Architecture Version 1.5 and the EJB Version 2.1 specifications do not restrict message listeners to using the JMS API.

- ▶ ActivationSpec JavaBean

For each message listener type supported for the resource adapter, the deployment descriptor must also specify the Java class name of the ActivationSpec JavaBean. An ActivationSpec JavaBean instance encapsulates the configuration information needed to set up asynchronous

message delivery to a message endpoint. Section 8.3.4, “JMS ActivationSpec JavaBean” on page 428 discusses the ActivationSpec JavaBean for JMS providers in more detail.

► Required configuration properties

Each ActivationSpec can also specify a list of required properties. These required properties can be used to validate the configuration of an ActivationSpec JavaBean instance. Example 8-13 shows the messagelistener entry in the deployment descriptor for the default messaging JMS provider. Notice that it supports the JMS message listener (`javax.jms.MessageListener`) and that the ActivationSpec JavaBean has three required properties: `destination`, `destinationType`, and `busName`.

Example 8-13 J2EE Connector Architecture message listener definition

```
<inbound-resourceadapter>
  <messageadapter>
    <messagelistener>
      <messagelistener-type>
        javax.jms.MessageListener
      </messagelistener-type>
      <activationspec>
        <activationspec-class>
          com.ibm.ws.sib.api.jmsra.impl.JmsJcaActivationSpecImpl
        </activationspec-class>
        <required-config-property>
          <config-property-name>destination</config-property-name>
        </required-config-property>
        <required-config-property>
          <config-property-name>destinationType</config-property-name>
        </required-config-property>
        <required-config-property>
          <config-property-name>busName</config-property-name>
        </required-config-property>
      </activationspec>
    </messagelistener>
  </messageadapter>
</inbound-resourceadapter>
```

► Administered objects

The resource adapter deployment descriptor can also specify a set of administered objects. For each administered object listed, the deployment descriptor must provide the Java class name of the administered object and the interface that it implements.

These administered objects are similar in nature to JMS administered objects, discussed in 8.2.4, “JMS administered objects” on page 408. In fact, for the

default messaging JMS provider within WebSphere Application Server, the J2EE Connector Architecture administered objects that it defines implement the relevant JMS administered object interfaces. This is shown in Example 8-14.

Example 8-14 J2EE Connector Architecture administered object definition

```
<adminobject>
  <adminobject-interface>
    javax.jms.Queue
  </adminobject-interface>
  <adminobject-class>
    com.ibm.ws.sib.api.jms.impl.JmsQueueImpl
  </adminobject-class>
  <config-property>
    <config-property-name>QueueName</config-property-name>
    <config-property-type>java.lang.String</config-property-type>
  </config-property>

  ... additional properties removed ...

  <config-property>
    <config-property-name>BusName</config-property-name>
    <config-property-type>java.lang.String</config-property-type>
  </config-property>
</adminobject>
```

8.3.4 JMS ActivationSpec JavaBean

An ActivationSpec JavaBean instance encapsulates the configuration information needed to set up asynchronous message delivery to a message endpoint. The J2EE Connector Architecture recommends that JMS providers include the following properties in their implementation of an ActivationSpec JavaBean:

- destination

Recall that a JMS destination encapsulates addressing information for the JMS provider. A JMS client explicitly specifies a destination when sending a message to, or receiving a message from, the JMS provider. A message endpoint needs to specify which destination the resource adapter should monitor for incoming messages. The resource adapter is then responsible for notifying the message endpoint when a message arrives at the specified destination.

The J2EE Connector Architecture does not define the format for the destination property, but it does acknowledge that it is not always practical for a value to be specified in the deployment descriptor for a message endpoint

application. However, a value for the destination property is required when deploying the message endpoint application. For this reason, the J2EE Connector Architecture recommends that a JMS resource adapter defines the destination property as a required property on the ActivationSpec JavaBean. The resource adapter for the default messaging JMS provider within WebSphere Application Server does just this, as shown in Example 8-13 on page 427.

The J2EE Connector Architecture also recommends that, if the destination object specified implements the `javax.jms.Destination` interface, the JMS resource adapter should provide an administered object that implements this same interface. Once again, the resource adapter for the default messaging JMS provider within WebSphere Application Server does just this, as shown in Example 8-14 on page 428.

- **destinationType**

The `destinationType` property simply indicates whether the destination specified is a JMS queue or JMS topic. The valid values for this property are, therefore, `javax.jms.Queue` or `javax.jms.Topic`. The J2EE Connector Architecture recommends that a JMS resource adapter defines the `destinationType` property as a required property on the ActivationSpec JavaBean. The resource adapter for the default messaging JMS provider within WebSphere Application Server does just this, as shown in Example 8-13 on page 427.

- **messageSelector**

The JMS ActivationSpec JavaBean can optionally define a `messageSelector` property. JMS message selectors are discussed in “Message selectors” on page 414.

- **acknowledgeMode**

The JMS ActivationSpec JavaBean can optionally define an `acknowledgeMode` property. This property indicates to the EJB container how a message received by a message endpoint (MDB) should be acknowledged. Valid values for this property are `Auto-acknowledge` or `Dups-ok-acknowledge`. If no value is specified, `Auto-acknowledge` is assumed.

For a full description of message acknowledgement, please see both the JMS Version 1.1 and the EJB Version 2.1 specifications. Links for these specifications are contained in 8.8, “References and resources” on page 536.

- **subscriptionDurability**

The JMS ActivationSpec JavaBean can optionally define a `subscriptionDurability` property. This property is only relevant if the message endpoint (MDB) is receiving messages from a JMS topic. The `destinationType` property specifies a value of `javax.jms.Topic`.

As discussed in “Durable subscriptions in the Publish/Subscribe domain” on page 407, in the JMS Publish/Subscribe domain, in order for a message to be retained on a destination when a subscribing application is not available, the subscribing application must create a durable subscription. With message-driven beans, it is the EJB container that is responsible for creating subscriptions when the specified destination is a JMS topic. This property indicates to the EJB container whether it must create a durable subscription to the JMS topic.

The valid values for the `subscriptionDurability` property are either `Durable` or `NonDurable`. If no value is specified, `NonDurable` is assumed.

- ▶ `clientId`

The JMS `ActivationSpec` JavaBean can optionally define a `clientId` property. This property is only relevant if the message endpoint (MDB) defines a durable subscription to a JMS topic (the `destinationType` property specifies a value of `javax.jms.Topic` and the `subscriptionDurability` property specifies a value of `Durable`).

The JMS provider uses the `clientId` for durable subscriptions to uniquely identify a message consumer. If a message endpoint defines a durable subscription, then a value for the `clientId` property must be specified. A suitable value for the `clientId` property would normally be specified when deploying the message endpoint application.

- ▶ `subscriptionName`

The JMS `ActivationSpec` JavaBean can optionally define a `subscriptionName` property. This property is only relevant if the message endpoint (MDB) defines a durable subscription to a JMS topic. The `destinationType` property specifies a value of `javax.jms.Topic` and the `subscriptionDurability` property specifies a value of `Durable`.

The JMS provider uses the `subscriptionName` in combination with the `clientId` to uniquely identify a message consumer. If a message endpoint defines a durable subscription, then a value for the `subscriptionName` property must be specified. A suitable value for the `subscriptionName` property would normally be specified when deploying the message endpoint application.

8.3.5 Message endpoint deployment

Before any messages can be delivered to a message endpoint, the message endpoint must be associated with a destination. This task is performed during application installation. Therefore, the responsibility of associating a message-driven bean with a destination lies with the application deployer.

The application deployer creates an instance of the `ActivationSpec` JavaBean for the relevant resource adapter and associates it with the message endpoint

during installation. In this way, an ActivationSpec JavaBean, through its destination property, associates a message endpoint with a destination on the message provider. This relationship is shown in Figure 8-10 on page 431.

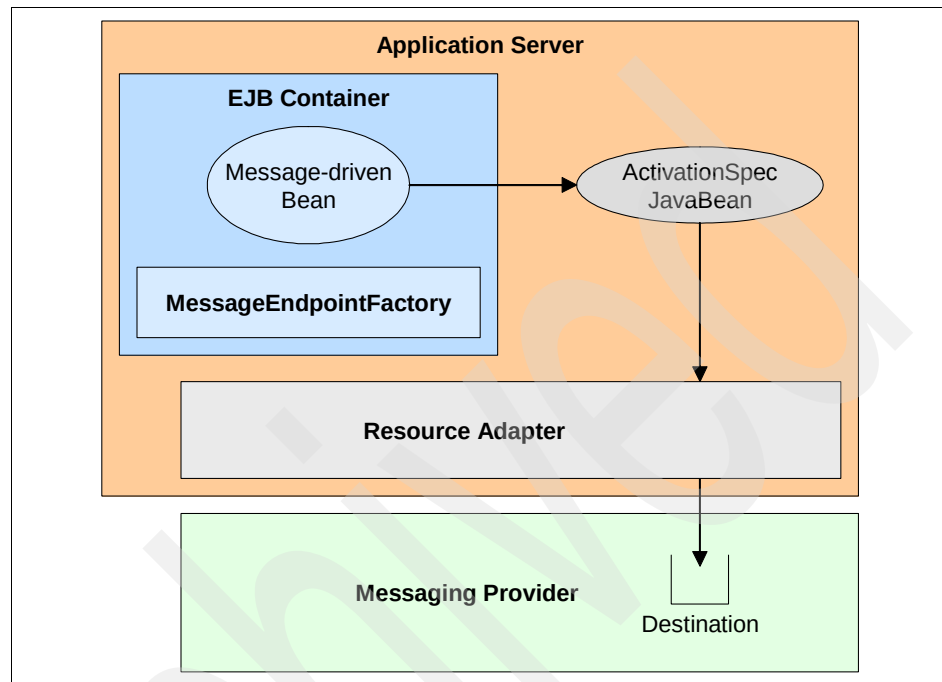


Figure 8-10 Associating an MDB with a destination using a ActivationSpec JavaBean

8.3.6 Message endpoint activation

A message endpoint is activated by the application server when the message endpoint application is started. During message endpoint activation, the application server passes the ActivationSpec JavaBean, and a reference to the MessageEndpointFactory, to the resource adapter by invoking its endpointActivation method.

The resource adapter uses the information in the ActivationSpec JavaBean to interact with messaging provider and set up message delivery to the message endpoint. For a JMS message-driven bean, this might involve configuring a message selector or a durable subscription against the destination. Once the endpointActivation method returns, the message endpoint is ready to receive messages. This process is shown in Figure 8-11 on page 432.

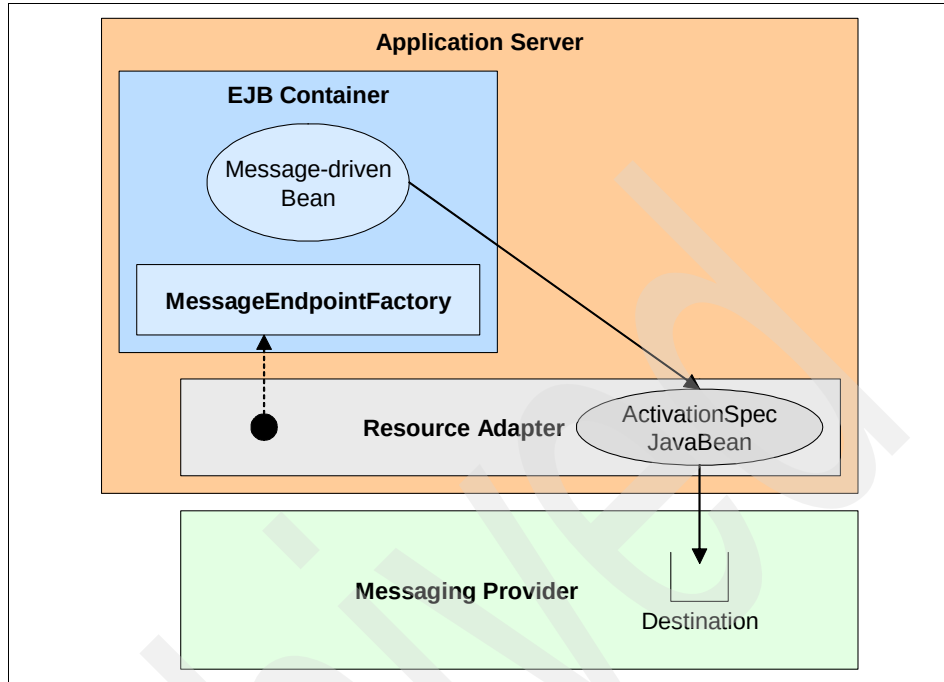


Figure 8-11 Activating a message endpoint

8.3.7 Message delivery

The following steps describe the sequence of events that occur when a message arrives at a destination:

1. The resource adapter detects the arrival of a message at the destination.
2. The resource adapter invokes the `createEndpoint` method on the `MessageEndpointFactory`.
3. The `MessageEndpointFactory` obtains a reference to a message endpoint. This might be an unused message endpoint obtained from a pool or, if no message endpoints are available, it can create a new message endpoint.
4. The `MessageEndpointFactory` returns a proxy to this message endpoint instance to the resource adapter.
5. The resource adapter uses the message endpoint proxy to deliver the message to the message endpoint.

This process is shown in Figure 8-12 on page 433.

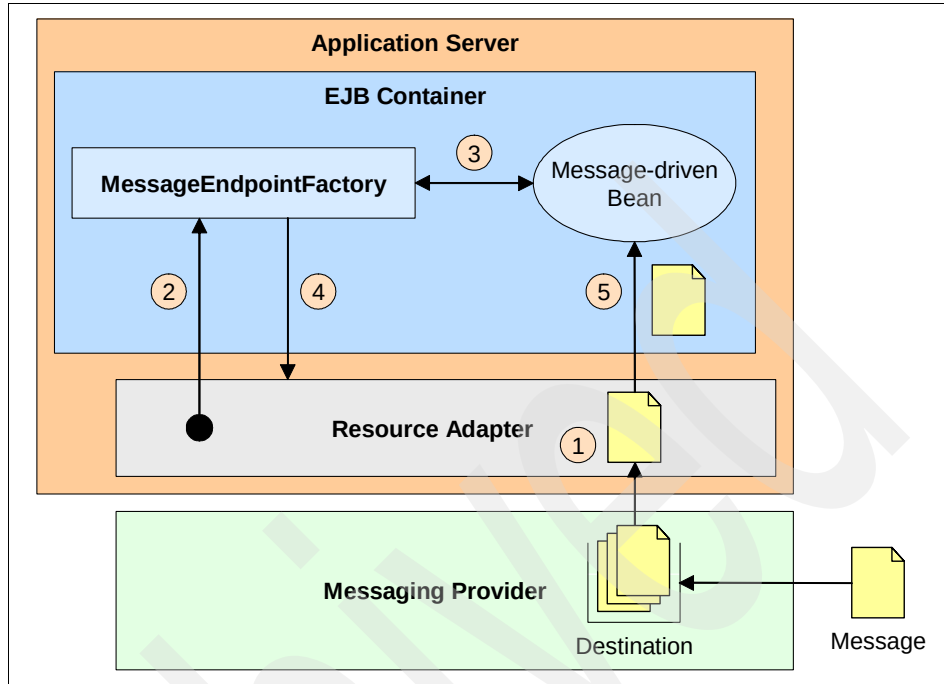


Figure 8-12 Delivering a message to a message endpoint

8.3.8 Administered objects

The resource adapter deployment descriptor defines the list of administered objects implemented by the resource adapter. However, it does not define any administered object instances. This must still be performed as an administrative task within the WebSphere administrative console. Because the default messaging JMS provider is specific to the JMS programming model, the WebSphere administrative console provides a set of JMS administration windows for this resource adapter. Section 8.6, “Configuring WebSphere JMS administered objects” on page 461 details the steps required to configure administered objects for the default messaging JMS provider.

8.4 Message-driven beans

The Enterprise JavaBeans specification (EJB) Version 2.0 introduced a new type of EJB called the message-driven bean (MDB). Message-driven beans are asynchronous message consumers that run within the context of an application servers EJB container. This enables the EJB container to provide additional services to the message-driven bean during the processing of a message, such as transactions, security, concurrency, and message acknowledgement.

The EJB container is also responsible for managing the lifetime of the message-driven beans and for invoking message-driven beans when a message arrives for which a given message-driven bean is the consumer.

Message-driven bean instances should not maintain any conversational state on behalf of a client. This enables the EJB container to maintain a pool of message-driven bean instances and to select any instance from this pool to process an incoming message. However, this does not prevent a message-driven bean from maintaining a state that is not specific to a client, for example, data source references or references to another EJB.

WebSphere Application Server V6 is fully compliant with Version 1.4 of the J2EE specification, which requires application servers to support Version 2.1 of the EJB specification.

8.4.1 Message-driven bean types

Version 2.0 of the EJB specification defined a single type of message-driven bean that enabled the asynchronous delivery of messages via the Java Message Service.

However, the integration of multiple JMS providers into application servers has proven difficult. For various reasons, many application server vendors have only provided support for one JMS provider within their product. Also, the fact that message-driven beans within the EJB 2.0 specification only support the JMS programming model was considered too restrictive. Several other messaging providers exist that require similar functionality to message-driven beans within the EJB container, such as the Java API for XML Messaging (JAXM).

Because of this, Version 2.1 of the EJB specification expanded the definition of message-driven beans to provide support for messaging providers other than JMS providers. It does this by allowing a message-driven bean to implement an interface other than the `javax.jms.MessageListener` interface. The type of message listener interface that a message-driven bean implements determines its type. Therefore, a message-driven bean that implements the

`javax.jms.MessageListener` interface is referred to as a *JMS message-driven bean*.

8.4.2 Client view of a message-driven bean

Unlike session and entity beans, message-driven beans do not expose home or component interfaces. A client is not able to locate instances of a message-driven bean and invoke methods on it directly.

The only manner in which a client can interact with a message-driven bean is to send a message to the destination or endpoint for which the message-driven bean is the listener. The EJB container is responsible for invoking an instance of the message-driven bean as a result of the arrival of a message. From the client's perspective, the existence of the message-driven bean is completely transparent. This is shown in Figure 8-13, where the client is only able to see the messaging provider and the target destination.

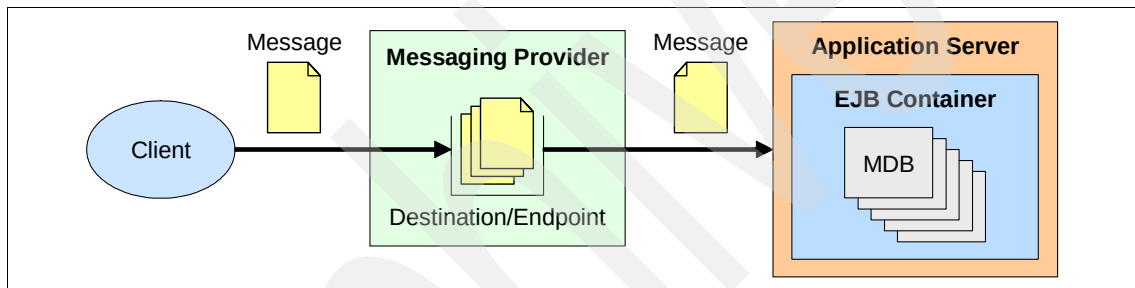


Figure 8-13 Client view of a message-driven bean

8.4.3 Message-driven bean implementation

A bean provider developing a message-driven bean must provide a message-driven bean implementation class. This class must implement, directly or indirectly, the `javax.ejb.MessageDrivenBean` interface and a message listener interface. It must also provide an `ejbCreate` method implementation. These aspects of message-driven implementation are discussed in the next sections.

MessageDrivenBean interface

The `javax.ejb.MessageDrivenBean` interface defines a number of callback methods that allow the EJB container to manage the life cycle of each message-driven bean instance. Because message-driven beans expose no home or component interfaces, the `javax.ejb.MessageDrivenBean` interface defines fewer callback methods than the corresponding `javax.ejb.SessionBean` and `java.ejb.EntityBean` interfaces. The definition of the `javax.ejb.MessageDrivenBean` interface is shown in Example 8-15.

Example 8-15 The `javax.ejb.MessageDrivenBean` interface

```
public interface MessageDrivenBean extends javax.ejb.EnterpriseBean
{
    public void setMessageDrivenContext(MessageDrivenContext ctx);
    public void ejbRemove();
}
```

The purpose of each of the callback methods is described below:

► **setMessageDrivenContext**

This method is invoked by the EJB container to associate a context with an instance of a message-driven bean. The message-driven bean instance stores a reference to the context as part of its state.

► **ejbRemove**

This method is invoked by the EJB container to notify the message-driven bean instance that it is in the process of being removed. This gives the message-driven bean the opportunity to release any resources that it might be holding.

Message listener interface

As discussed in 8.4.1, “Message-driven bean types” on page 434, Version 2.1 of the EJB specification no longer requires a message-driven bean to implement the `javax.jms.MessageListener` interface. The specification simply states that a message-driven bean is required to implement the appropriate message listener interface for the messaging type that the message-driven bean supports.

The specification also allows the message listener interface to define more than one message listener method and for these methods to specify return types. If a messaging provider has defined an interface that contains more than one message listener method, it is the responsibility of the resource adapter to determine which of these methods to invoke upon the receipt of a message.

The message listener interface for JMS message-driven beans is the `javax.jms.MessageListener` interface, as shown in Example 8-7 on page 417.

As an example of other types of message listener interface that might be used by messaging providers, again, consider a theoretical JAXM messaging provider. A JAXM messaging provider might decide to use the `javax.xml.messaging.ReqRespListener` interface as its message listener interface. This interface is shown in Example 8-16.

Example 8-16 The `javax.xml.messaging.ReqRespListener` interface

```
package javax.xml.messaging;

import javax.xml.soap.SOAPMessage;

public interface ReqRespListener
{
    public SOAPMessage onMessage(SOAPMessage message);
}
```

Notice that this interface is similar to the `javax.jms.MessageListener` interface in that it defines an `onMessage` method. However, any method name can be used when defining methods within the message listener interface.

Also, notice that the `onMessage` method specifies a return type of `SOAPMessage`. The `SOAPMessage` can be considered to be a reply message. However, because it is the EJB container that invokes the `onMessage` method, the `SOAPMessage` is returned to the EJB container. The EJB specification states that, if the message listener interface supports the request-reply pattern in this manner, it is the responsibility of the EJB container to deliver the reply message to the resource adapter.

The `ejbCreate` method

One other requirement on the implementation class for a message-driven bean is that it implements the `ejbCreate` method. Once again, this implementation can be defined within the message-driven bean class itself, or within any of its superclasses. The EJB container invokes the `ejbCreate` as the last step in creating a new instance of a message-driven bean. This gives the message-driven bean the opportunity to allocate any resources that it requires.

8.4.4 Message-driven bean life cycle

The EJB container is responsible for hosting and managing message-driven bean instances. It controls the life cycle of the message-driven bean and uses the callback methods within the bean implementation class to notify the instance when important state transitions are about to occur.

The life cycle of a message-driven bean is shown in Figure 8-14.

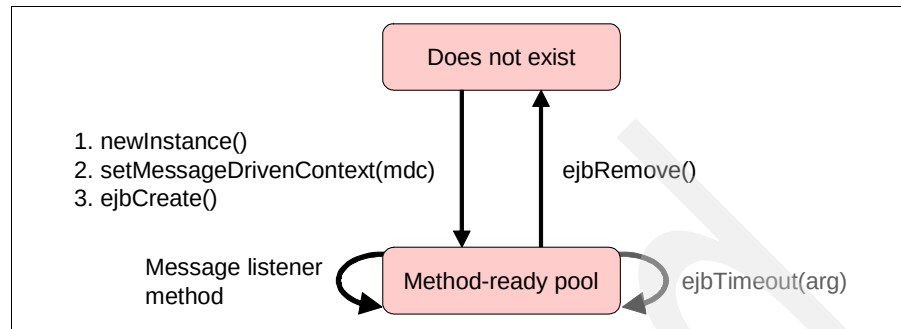


Figure 8-14 Message-driven bean life cycle

The relevant state transitions for a message-driven bean are:

► Message-driven bean creation

Message-driven bean instances are created in three steps by the EJB container:

- The EJB container invokes the `Class.newInstance()` method on the bean implementation class.
- The EJB container provides the new instance with its `MessageDrivenContext` reference by invoking the `setMessageDrivenContext` method.
- The EJB container gives the new message-driven bean instance the opportunity to perform one-time initialization by invoking the `ejbCreate` method. The message-driven bean is able to allocate any resources that it requires here.

► Message listener method invocation

Once in the method-ready pool, a message-driven bean instance is available to process any message that is sent to its associated destination or endpoint. When a message arrives at this destination, the EJB container receives the message and allocates a message-driven bean instance from the method-ready pool to process the message. When processing is complete, the message-driven bean instance is returned to the method-ready pool.

Note: The EJB container performs a number of other operations during the processing of a message, such as ensuring that the processing takes place within the specified transactional context and performing any required security checks. These steps have been omitted for clarity.

► Message-driven bean removal

The EJB container decides at any time that it needs to release resources. To do this, it can reduce the number of message-driven bean instances in the method-ready pool. As part of the removal process, it invokes the `ejbRemove` method on the instance being removed to give the message-driven bean the opportunity to release any resources that it might be holding.

8.4.5 Message-driven beans and transactions

A bean provider can specify whether a message-driven bean will demarcate its own transactions programmatically or whether it will rely on the EJB container to demarcate transactions on its behalf. The bean provider does this by specifying either `bean` or `container` as the value for the `transaction-type` field for the message-driven bean in the EJB module deployment descriptor.

Regardless of whether transaction demarcation is bean-managed or container-managed, a message-driven bean can only access the transactional context within which it is running by using the relevant methods of the `MessageDrivenContext` interface.

MessageDrivenContext interface

The `javax.ejb.MessageDrivenContext` interface extends the `javax.ejb.EJBContext` interface. However, unlike the `SessionContext` and `EntityContext` interfaces, the `MessageDrivenContext` interface does not define any additional methods. The parent `EJBContext` interface is shown in Example 8-17.

Example 8-17 The `javax.ejb.EJBContext` interface

```
package javax.ejb;

import java.security.Identity;
import java.security.Principal;
import java.util.Properties;
import javax.transaction.UserTransaction;

public interface EJBContext
{
    // EJB Home methods
    public abstract EJBHome getEJBHome();
    public abstract EJBLocalHome getEJBLocalHome();

    // Security methods
    public abstract Principal getCallerPrincipal();
    public abstract boolean isCallerInRole(String s);

    // Transaction methods
```

```

public abstract UserTransaction getUserTransaction()
    throws IllegalStateException;
public abstract void setRollbackOnly() throws IllegalStateException;
public abstract boolean getRollbackOnly() throws IllegalStateException;

// Timer service methods
public abstract TimerService getTimerService()
    throws IllegalStateException;

// Deprecated Methods
public abstract Properties getEnvironment();
public abstract Identity getCallerIdentity();
public abstract boolean isCallerInRole(Identity identity);
}

```

Note: When using a message-driven bean instance, only invoke the transaction and timer service methods exposed by the `MessageDrivenContext` interface.

Attempting to invoke the EJB home methods results in a `java.lang.IllegalStateException` being thrown because message-driven beans do not define `EJBHome` or `EJBLocalHome` objects.

Attempting to invoke the `getCallerPrincipal` method is allowed by Version 2.1 of the EJB specification. However, with a message-driven bean, the caller is the EJB container, which does not have a client security context. In this situation the `getCallerPrincipal` method returns a representation of the unauthenticated identity. Invoking the `isCallerInRole` method is still not allowed by the EJB specification and will result in a `java.lang.IllegalStateException` being thrown.

Container-managed transactions

A message-driven bean with a transaction-type of `Container` is said to make use of *container-managed transactions*. When a message-driven bean is using container-managed transactions, the EJB container uses the transaction attribute of the message listener method to determine the actions that it needs to take when a message arrives at the relevant destination.

The transaction attributes that can be specified for message listener method are:

► NotSupported

The EJB container does not create a transaction prior to receiving the message from the destination and invoking the message listener method on the message-driven bean. Consequently, if the message-driven bean

accesses other resource managers or enterprise beans, it does so with an unspecified transaction context.

Also, depending on the capabilities of the underlying JMS provider, if an error occurs during the processing of the message, it might not be placed back on the destination for redelivery.

► Required

The EJB container creates a transaction prior to receiving the message from the destination and invoking the message listener method on the message-driven bean.

If the message-driven bean accesses a resource manager within the message listener method, then this access takes place within the context of this transaction. Similarly, if the message-driven bean invokes other EJBs within the message listener method, the EJB container passes the transaction context with the invocation.

When the message listener method completes, the EJB container attempts to commit the transaction. For a JMS message-driven bean, a rollback of the transaction has the effect of placing the message back on the destination for redelivery.

When a message listener method specifies a transaction attribute of Required, it can only use the `getRollbackOnly` and `setRollbackOnly` methods of the `MessageDrivenContext` object. The code required to mark a transaction for rollback within a message listener method is shown in Example 8-18.

Example 8-18 Using the `setRollbackOnly` method

```
public class SampleMDBBean implements MessageDrivenBean, MessageListener
{
    private MessageDrivenContext msgDrivenCtx;

    // Lifecycle methods removed for clarity

    public void onMessage(Message msg)
    {
        try
        {
            // Process the message

            // Try to access a relational database
        }
        catch (SQLException e)
        {
            // An error occurred, rollback the transaction
            msgDrivenCtx.setRollbackOnly();
        }
    }
}
```

```
}  
}
```

Bean-managed transactions

A message-driven bean with a transaction-type of Bean is said to make use of bean-managed transactions. When a message-driven bean is using bean-managed transactions, the EJB container does **not** create a transaction prior to receiving the message from the destination and invoking the message listener method on the message-driven bean. Consequently, for a JMS message-driven bean, the message might not be placed back on the destination for redelivery if an error occurs during the processing of the message. The message listener method is responsible for creating any transactions that it requires when processing a message.

A message-driven bean using bean-managed transactions can only use the `getUserTransaction` method of the `MessageDrivenContext` object. It is then able to use the `javax.transaction.UserTransaction` interface to begin, commit, and roll back transactions. The code required to use the `UserTransaction` interface within a message listener method is shown in Example 8-19.

Example 8-19 Using the `javax.transaction.UserTransaction` interface

```
public class SampleMDBBean implements MessageDrivenBean, MessageListener  
{  
    private MessageDrivenContext msgDrivenCtx;  
  
    // Lifecycle methods removed for clarity  
  
    public void onMessage(Message msg)  
    {  
        // Get the UserTransaction object reference  
        UserTransaction userTx = msgDrivenCtx.getUserTransaction();  
  
        try  
        {  
            // Begin the transaction  
            userTx.begin();  
  
            // Process the message  
  
            // Try to access a relational database  
  
            // Attempt to commit the transaction  
            userTx.commit();  
        }  
        catch (Exception e)  
        {  
            // Handle exception  
        }  
    }  
}
```

```

        try
        {
            // An error occurred, rollback the transaction
            userTx.rollback();
        }
        catch (SystemException e2)
        {
            e2.printStackTrace();
        }
    }
}

```

Note: Because of the complex nature of distributed transactions, it is recommended that bean providers make use of container-managed transactions.

8.4.6 Message-driven bean activation configuration properties

The way in which message-driven beans specify deployment options within the EJB deployment descriptor has changed significantly for EJB Version 2.1. This reflects the changes made to the J2EE Connector Architecture specification to enable a resource adapter to asynchronously deliver messages to a message-driven bean, independent of the specific messaging style, messaging semantics, and messaging infrastructure. Consequently, Version 2.1 of the EJB specification introduced a more generic mechanism to specify the messaging semantics of a message-driven bean, known as *activation configuration* properties.

The EJB specification defines the following activation configuration properties for a JMS message-driven bean:

- ▶ destinationType
- ▶ messageSelector
- ▶ acknowledgeMode
- ▶ subscriptionDurability

Notice that the names of these activation configuration properties match the names of the equivalent JMS ActivationSpec JavaBean properties described in 8.3.4, “JMS ActivationSpec JavaBean” on page 428. The description of each of the properties is also the same.

This is intentional on the part of the J2EE Connector Architecture and the EJB specifications. The intention is that this will allow the automatic merging of the activation configuration element values with the corresponding entries in the JMS ActivationSpec JavaBean, while configuring the JMS ActivationSpec JavaBean

during endpoint deployment. This is exactly what happens when WebSphere starts an application that contains a message-driven bean.

Note: If a message-driven bean and the JMS activation specification with which it is associated both specify a value for a given property, the value contained in the EJB deployment descriptor for the message-driven bean will be used.

Example 8-20 on page 444 shows the relevant entry for the BankListener message-driven bean that is packaged as part of the WebSphereBank sample in WebSphere Application Server. The elements of the deployment descriptor that are specific to messaging are shown in bold. Table 8-8 shows activation configuration properties that are defined within the deployment descriptor.

Table 8-8 Activation configuration properties for the BankListener message-driven bean

Property name	Property value
destinationType	javax.jms.Queue
acknowledgeMode	Auto-acknowledge
messageSelector	JMSType = 'transfer'

Example 8-20 BankListener message-driven bean deployment descriptor

```
<message-driven id="MessageDriven_1037986117955">
  <ejb-name>BankListener</ejb-name>
  <ejb-class>com.ibm.websphere.samples.bank.ejb.BankListenerBean</ejb-class>
  <message-listener>javax.jms.MessageListener</message-listener>
  <transaction-type>Container</transaction-type>
  <message-destination-type>javax.jms.Queue</message-destination-type>
  <message-destination-link>BankJSQueue</message-destination-link>
  <activation-config>
    <activation-config-property>
      <activation-config-property-name>
        destinationType
      </activation-config-property-name>
      <activation-config-property-value>
        javax.jms.Queue
      </activation-config-property-value>
    </activation-config-property>
    <activation-config-property>
      <activation-config-property-name>
        acknowledgeMode
      </activation-config-property-name>
      <activation-config-property-value>
        Auto-acknowledge
      </activation-config-property-value>
    </activation-config-property>
  </activation-config>
</message-driven>
```



```

        </activation-config-property-value>
    </activation-config-property>
    <activation-config-property>
        <activation-config-property-name>
            messageSelector
        </activation-config-property-name>
        <activation-config-property-value>
            JMSType = 'transfer'
        </activation-config-property-value>
    </activation-config-property>
</activation-config>
<ejb-local-ref id="EJBLocalRef_1037986243867">
    <description></description>
    <ejb-ref-name>ejb/Transfer</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <local-home>
        com.ibm.websphere.samples.bank.ejb.TransferLocalHome
    </local-home>
    <local>com.ibm.websphere.samples.bank.ejb.TransferLocal</local>
    <ejb-link>Transfer</ejb-link>
</ejb-local-ref>
</message-driven>

```

8.4.7 Associating a message-driven bean with a destination

Before any messages can be delivered to a message-driven bean, the message-driven bean must be associated with a destination. As discussed in 8.3.5, “Message endpoint deployment” on page 430, the responsibility of associating a message-driven bean with a destination lies with the application deployer.

Within WebSphere Application Server, there are two mechanisms that can be used to associate these objects: JMS activation specifications and listener ports. This is due to the fact that the service integration bus is accessed using a J2EE Connector Architecture resource adapter, while WebSphere MQ is accessed using a standard JMS API implementation.

If the message-driven bean that is being deployed needs to be associated with a destination defined on a service integration bus, use a JMS activation specification. If the message-driven bean that is being deployed needs to be associated with a destination defined on WebSphere MQ, use a listener port. JMS activation specifications and listener ports are discussed in the sections that follow.

JMS activation specification

An ActivationSpec JavaBean, through its destination property, associates a message endpoint with a destination. Within WebSphere Application Server, an instance of the ActivationSpec JavaBean for the default messaging JMS provider is configured by creating a JMS activation specification using the WebSphere administrative console. These JMS activation specifications are normally created prior to installing the message-driven bean application and are stored in the JNDI name space by WebSphere.

At installation time, the deployer specifies which JMS activation specification to associate with a particular message-driven bean, using its JNDI name. The destination property within the JMS activation specification, specifies the JNDI name of the target JMS destination. This relationship is shown Figure 8-15.

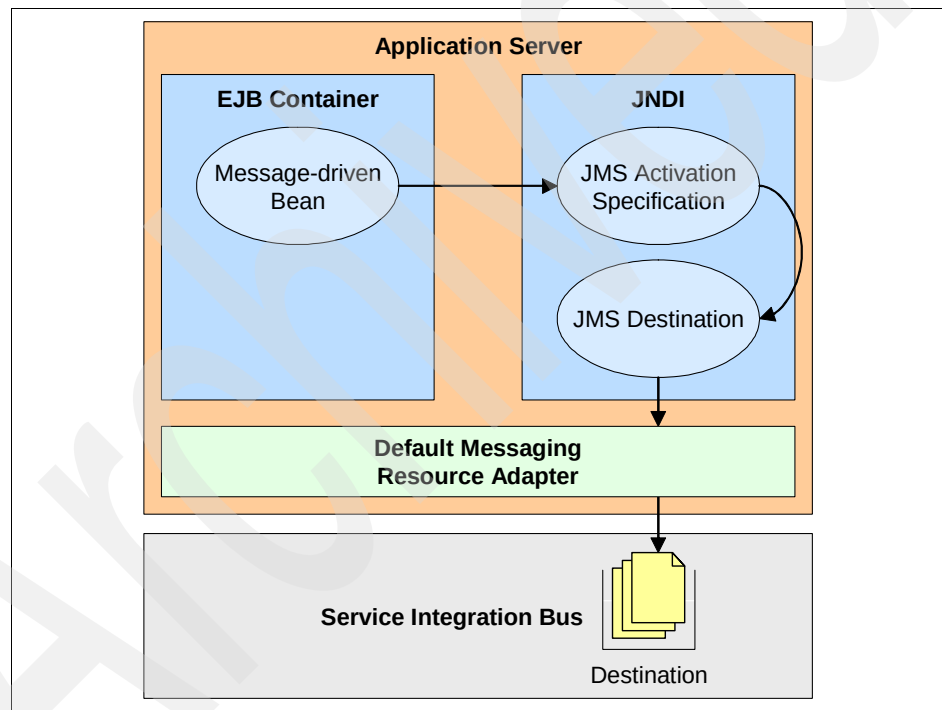


Figure 8-15 Associating an MDB with a destination using a JMS activation specification

The steps required to create a JMS activation specification for the default messaging JMS provider are described in “JMS activation specification configuration” on page 488.

Listener ports

Prior to Version 1.5 of the J2EE Connector Architecture, there was no standard way to associate a message-driven bean with a destination. To solve this problem, WebSphere Application Server V5 introduced the concept of a listener port. A listener port is used to simplify the administration of the association between a connection factory, destination, and deployed message-driven bean, as shown in Figure 8-16 on page 447. WebSphere Application Server V6 continues to use listener ports for those JMS providers that are not accessed using a resource adapter.

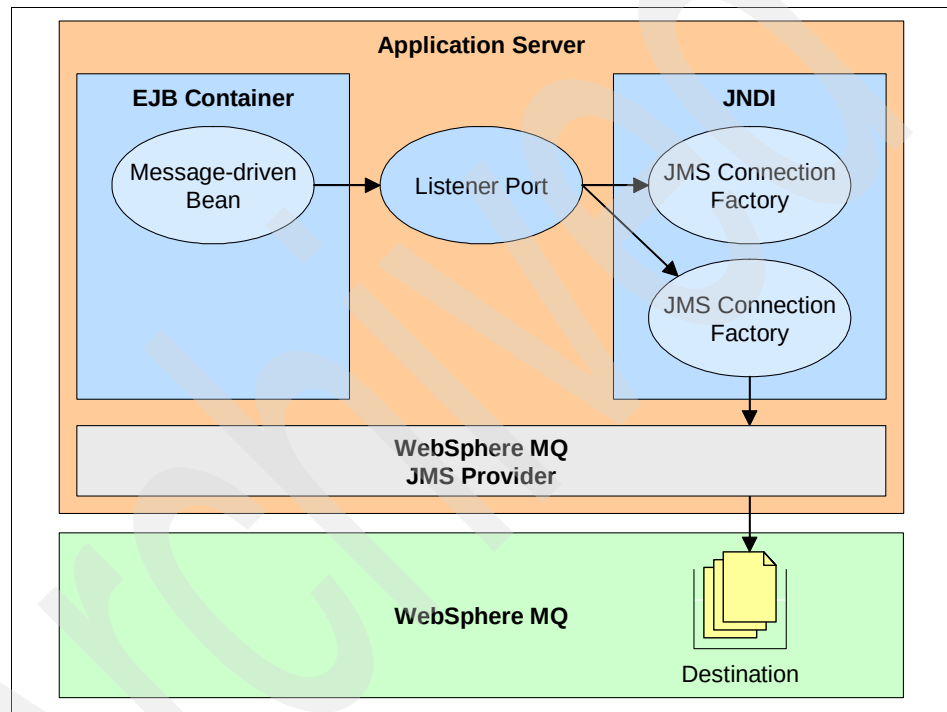


Figure 8-16 Associating an MDB with a destination using a listener port

The steps required to create a listener are described in 8.6.4, “Configuring listener ports” on page 511.

8.4.8 Message-driven bean best practices

As with all programming models, certain best practices have emerged for using the message-driven bean programming model. These best practices are discussed below:

- Delegate business logic to another handler.

Traditionally, the role of a stateless session bean is to provide a facade for business logic. Message-driven beans should delegate the business logic concerned with processing the contents of a message to a stateless session bean. Message-driven beans can then focus on what they were designed to do, which is processing messages. This is shown in Figure 8-17.

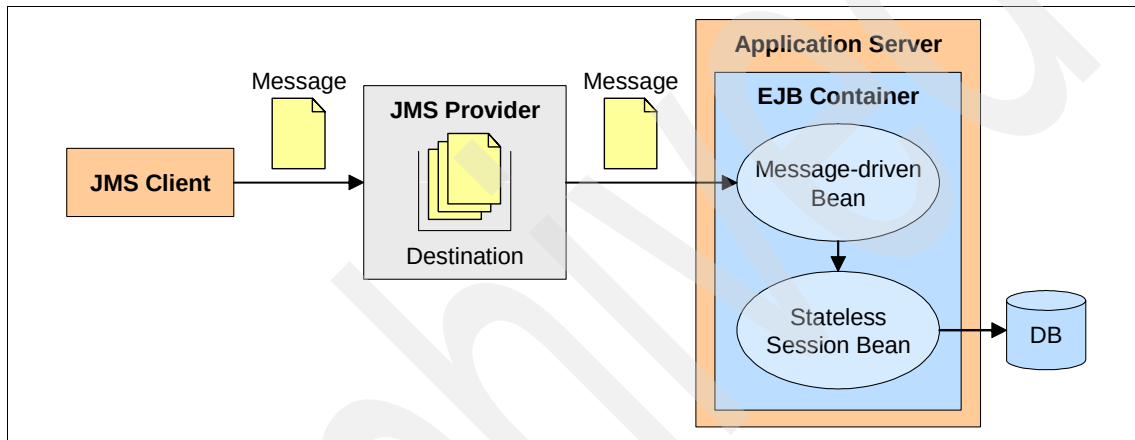


Figure 8-17 Delegating business logic to a stateless session bean

An additional benefit of this approach is that the business logic within the stateless session bean can be reused by other EJB clients. This is shown in Figure 8-18.

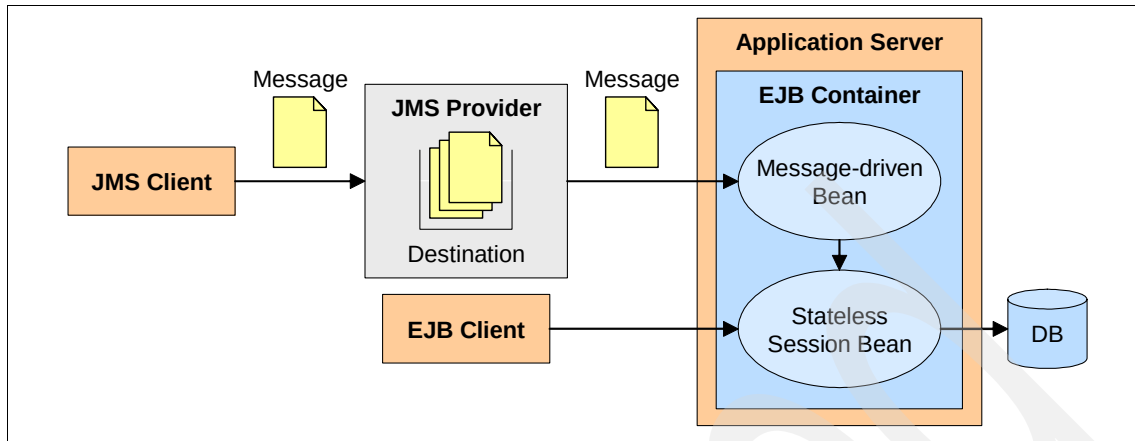


Figure 8-18 Business logic reuse

- Do not maintain a client-specific state within an MDB.

As discussed earlier, message-driven bean instances should not maintain any conversational state on behalf of a client. This enables the EJB container to maintain a pool of message-driven bean instances and to select any instance from this pool to process an incoming message. However, this does not prevent a message-driven bean from maintaining a state that is not specific to a client, for example, datasource references or references to another EJB.

- Avoid large message bodies.

A JMS message probably will travel over the network at some point in its life. It will definitely need to be handled by the JMS provider. All of these components contribute to the overall performance and reliability of the system. The amount of data contained in the body of a JMS message should be kept as small as possible to avoid impacting the performance of the network or the JMS provider.

- Minimize message processing time.

Recall from the discussion in 8.4.4, “Message-driven bean life cycle” on page 437 that instances of a message-driven bean are allocated from the method-ready pool to process incoming messages. These instances are not returned to the method-ready pool until message processing is complete. Therefore, the longer it takes for a message-driven bean to process a message, the longer it is unavailable for reallocation.

If an application is required to process a high volume of messages, the number of message-driven bean instances in the method-ready pool could be rapidly depleted if each message requires a significant processing. The EJB

container would then need to spend valuable CPU time creating additional message-driven bean instances for the method-ready pool, further impacting the performance of the application.

Additional care must be taken if other resources are enlisted into a global transaction during the processing of a message. The EJB container will not attempt to commit the global transaction until the MDB's `onMessage` method returns. Until the global transaction commits, these resources cannot be released on the resource managers in question.

For these reasons, the amount of time required to process each message should be kept to a minimum.

- Avoid dependencies on message ordering.

Try to avoid having an application making any assumptions with regard to the order in which JMS messages are processed. This is due to the fact that application servers enable the concurrent processing of JMS messages by MDBs and that some messages can take longer to process than others. Consequently, a message delivered later in a sequence of messages might finish message processing before a message delivered earlier in the sequence. It might be possible to configure the application server in such a way that messaging ordering is maintained within the application, but this is usually done at the expense of performance or architectural flexibility, such as the inability to deploy an application to a cluster.

- Be aware of poison messages.

Sometimes, a badly-formatted JMS message arrives at a destination. Such a message might cause an exception to be thrown within the MDB during message processing. An MDB that is making use of container-managed transactions then marks the transaction for rollback, as discussed in 8.4.5, "Message-driven beans and transactions" on page 439. The EJB container rolls back the transaction, causing the message to be placed back on the queue for redelivery. However, the same problem occurs within the MDB the next time the message is delivered. In this situation, such a message might be received, and then returned to the queue, repeatedly. These messages are known as *poison messages*.

Fortunately, some messaging providers have implemented mechanisms that can detect poison messages and redirect them to another destination. WebSphere MQ and the service integration bus are two such providers.

8.5 Managing WebSphere JMS providers

WebSphere Application Server V6 supports the following JMS providers:

- ▶ Default messaging
- ▶ WebSphere MQ
- ▶ Generic
- ▶ V5 default messaging

The sections that follow describe the first three of these JMS providers and how the WebSphere administrative console can be used to configure and administer them. Note that the V5 default messaging provider is supported for migration purposes only. We are not discussing that provider in this IBM Redbook. For information about the V5 default messaging provider, see *IBM WebSphere Application Server V5.1 System Management and Configuration*, SG24-6195.

New in V6.1: The path to access JMS resources from the administrative console has been shortened in some cases. For example, you can list the JMS queue connection factories and JMS queues without selecting a JMS provider first. A new option for scope (All scopes) allows you to display all of the selected resource types as opposed to only those defined at a specific scope.

8.5.1 Managing the default messaging JMS provider

WebSphere Application Server supplies a preconfigured JCA resource adapter implementation that can be used to communicate with a service integration bus. This resource adapter is installed as a fully-integrated component at all levels of the cell and needs no separate installation steps. The administered objects for this resource adapter also implement the corresponding interfaces of Version 1.1 of the JMS specification. This enables them to be used by JMS clients for both the Point-to-Point and Publish/Subscribe messaging models.

The WebSphere administrative console exposes a set of windows that you can use to configure the resource adapter as though it were purely a JMS provider, known as the default messaging JMS provider. These windows can be used to configure the following JMS resources:

- ▶ A JMS connection factory that can be used to connect to a service integration bus
- ▶ A JMS queue or topic destination that refers to a destination on a service integration bus

Such JMS queues and topics are available, over a long period of time, to all applications with access to the bus destination.

- A JMS activation specification that can be used to associate a message-driven bean with a JMS queue or topic destination

Note: You do not have to configure the underlying service integration bus resources before configuring the corresponding JMS resources. However, certain fields within the default messaging JMS provider administration windows are populated with relevant bus resources, if they exist. Therefore, to simplify the process of creating JMS resources for the default messaging JMS provider, we recommend that you create and configure the underlying service integration bus resources first.

The sections that follow discuss how to configure the resource adapter using the default messaging JMS provider windows. To view the properties of the default messaging JMS provider, use the administrative console to complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS** → **JMS Providers**.
2. Set the scope for the JMS Provider.
3. Click **Default messaging provider**.
4. The properties for the Default messaging JMS provider are displayed in the main content pane of the WebSphere administrative console, as shown in Figure 8-19 on page 452.

The screenshot displays the 'Configuration' window for the 'Default messaging provider'. It is divided into two main sections: 'General Properties' and 'Additional Properties'. The 'General Properties' section includes three text input fields: 'Scope' (containing 'Node=ITSOCellNode'), 'Name' (containing 'Default messaging provider'), and 'Description' (containing 'Default messaging provider'). The 'Additional Properties' section is a list of links: 'Connection factories', 'Queue connection factories', 'Topic connection factories', 'Queues', 'Topics', and 'Activation specifications'. An 'OK' button is located at the bottom left of the window.

Figure 8-19 Default messaging provider configuration properties

It is worth noting that the resource adapter can also be configured as a generic J2EE Connector Architecture resource adapter. However, the administration windows used for configuring a generic resource adapter are not specific to JMS resources and are, therefore, not as easy to use as the default messaging JMS provider administration windows. To view the properties of the resource adapter, use the administrative console to complete the following steps:

1. In the navigation tree, expand **Resources** → **Resource adapters**.
2. Set the scope for the resource adapter.
3. Set the **Preferences** to show built-in resources. Press **Apply**.
4. A list of resource adapters defined at this scope is displayed. Remember that the resource adapter for the service integration bus is defined at all levels within the cell. The list of resource adapters is shown in Figure 8-20 on page 453.

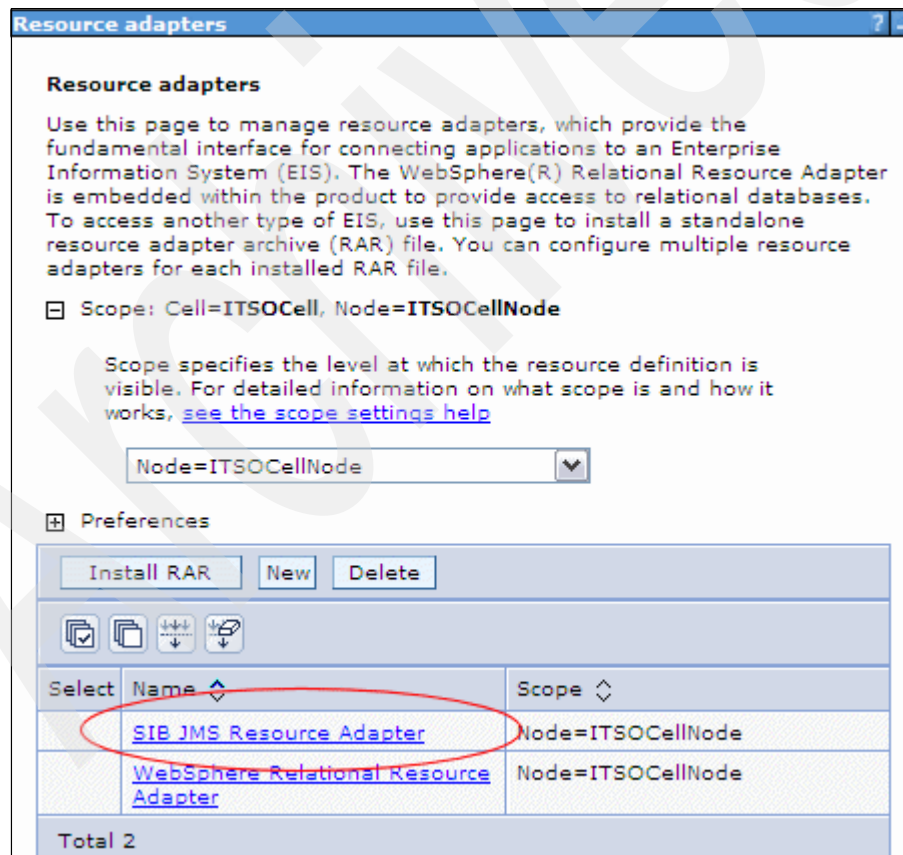


Figure 8-20 Resource adapters

5. Click **SIB JMS Resource Adapter**.
6. The properties for the resource adapter are displayed. These are shown in Figure 8-21 on page 454.

General Properties

- * Scope
cells:ITSOCell:nodes:ITSOCellNode
- * Name
SIB JMS Resource Adapter
- Description
Default messaging provider
- * Archive path
\${WAS_INSTALL_ROOT}/installedConnectors/sib.api.jmsra.rar
- Class path
\${WAS_INSTALL_ROOT}/installedConnectors/sib.api.jmsra.rar
- Native path
- Thread pool alias
Default

Additional Properties

- [J2C activation specifications](#)
- [J2C administered objects](#)
- [J2C connection factories](#)
- [Custom properties](#)
- [View Deployment Descriptor](#)

Apply OK Reset Cancel

Figure 8-21 SIB JMS Resource Adapter properties

The links under the **Additional Properties** section of the configuration window, shown in Figure 8-21, can be used to configure the following J2C resources at the relevant scope of the resource adapter:

- J2C activation specifications
- J2C administered objects
- J2C connection factories

Note: Using the generic resource adapter configuration windows to configure JMS resources for a service integration bus is not recommended. However, the following advanced properties for a JMS activation specification can be configured only using these windows:

- ▶ readAhead
- ▶ shareDataSourceWithCMP
- ▶ targetTransportChain

8.5.2 Managing the WebSphere MQ JMS provider

WebSphere Application Server V6 supplies a pre-configured JMS provider implementation for communicating with installations of the following products, using both the Point-to-Point and Publish/Subscribe messaging models:

- ▶ WebSphere MQ
- ▶ WebSphere Business Integration Event Broker
- ▶ WebSphere Business Integration Message Broker

Note: Publish/Subscribe functionality for WebSphere MQ is provided through the WebSphere MQ MA0C SupportPac™. However, use of MA0C is discouraged, because the other brokers provide a much more robust production publish/subscribe environment.

The WebSphere MQ JMS provider allows WebSphere solutions to be integrated into heterogeneous WebSphere MQ environments. It is also fully compliant with Version 1.1 of the JMS specification.

Note: Unlike the default messaging JMS provider, the WebSphere MQ JMS provider is not a J2EE Connector Architecture Version 1.5 compliant resource adapter. It simply provides an implementation of Version 1.1 of the JMS API, enabling JMS clients to communicate directly with WebSphere MQ.

However, the WebSphere MQ JMS provider is only partially integrated into WebSphere system management. While the WebSphere administration tools can be used to both configure and manage WebSphere MQ JMS administered objects, the creation and management of queue managers, channels, and queues must be performed using WebSphere MQ native tools.

To view the properties of the WebSphere MQ JMS provider, use the administrative console to do the following:

1. In the navigation tree, expand **Resources** → **JMS** → **JMS Providers**.
2. Set the scope for the JMS Provider.

- Click **WebSphere MQ messaging provider**, as shown in Figure 8-22 on page 456.

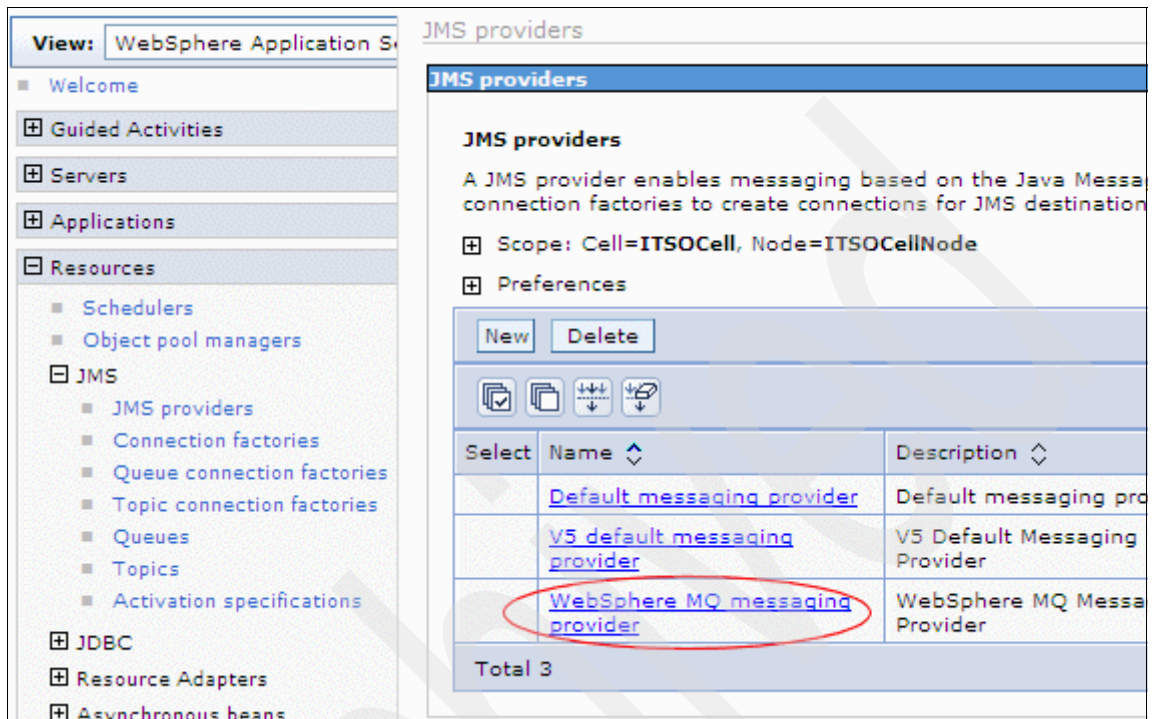
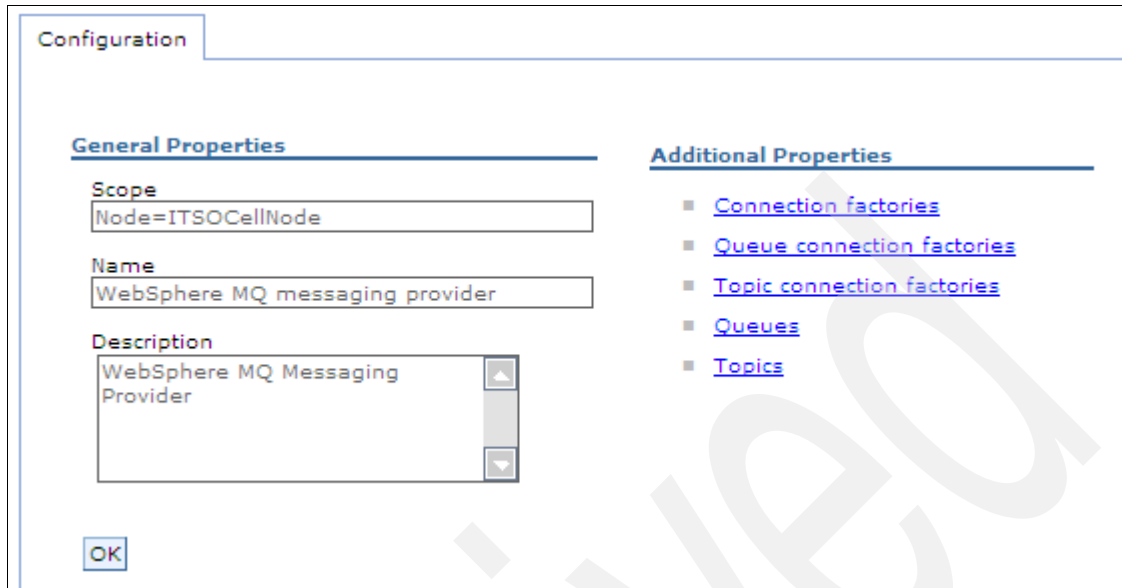


Figure 8-22 Finding the WebSphere MQ JMS provider in the navigation tree

- The properties for the WebSphere MQ JMS provider are displayed in the main content pane of the WebSphere administrative console, as shown in Figure 8-23 on page 457.



The image shows a 'Configuration' dialog box for the WebSphere MQ JMS provider. It is divided into two main sections: 'General Properties' and 'Additional Properties'. In the 'General Properties' section, there are three text input fields: 'Scope' with the value 'Node=ITSOCellNode', 'Name' with the value 'WebSphere MQ messaging provider', and 'Description' with the value 'WebSphere MQ Messaging Provider'. There is an 'OK' button at the bottom left. The 'Additional Properties' section on the right contains a list of links: 'Connection factories', 'Queue connection factories', 'Topic connection factories', 'Queues', and 'Topics'.

General Properties	Additional Properties
Scope Node=ITSOCellNode	■ Connection factories
Name WebSphere MQ messaging provider	■ Queue connection factories
Description WebSphere MQ Messaging Provider	■ Topic connection factories
	■ Queues
	■ Topics

OK

Figure 8-23 WebSphere MQ JMS provider configuration properties

8.5.3 Managing a generic JMS provider

WebSphere Application Server supports the use of third-party JMS providers within its run time environment through the use of a generic JMS provider. However, unlike the default messaging and WebSphere MQ JMS providers, a generic JMS provider must be defined to WebSphere Application Server before any JMS resources can be configured for that provider. Defining a generic JMS provider to WebSphere ensures that the JMS provider classes are available on the application server classpath at run time.

A generic JMS provider is recommended in the following situations:

- ▶ A non-WebSphere MQ messaging system already exists in the environment, and into which the WebSphere installation is required to integrate directly.
- ▶ A non-WebSphere MQ JMS provider supports functionality that is not available using the default messaging or WebSphere MQ JMS providers, and which would be useful for the user's messaging environment.

Note: WebSphere Application Server also supports the use of third-party JMS providers that are implemented as J2EE Connector Architecture resource adapters. The JMS resources for such JMS providers are configured using the generic resource adapter configuration windows.

If the third-party JMS provider is not implemented as a J2EE Connector Architecture resource adapter, we recommend that it supports the JMS Application Server Facilities described in 8.2.12, “Application Server Facilities” on page 421.

WebSphere interaction with a generic JMS provider

The JMS administered objects for a generic JMS provider are bound into the local JNDI name space within WebSphere Application Server. However, these JNDI entries act as aliases to the real JMS administered objects that have been configured in the external JNDI name space of the messaging provider. This is shown in Figure 8-24.

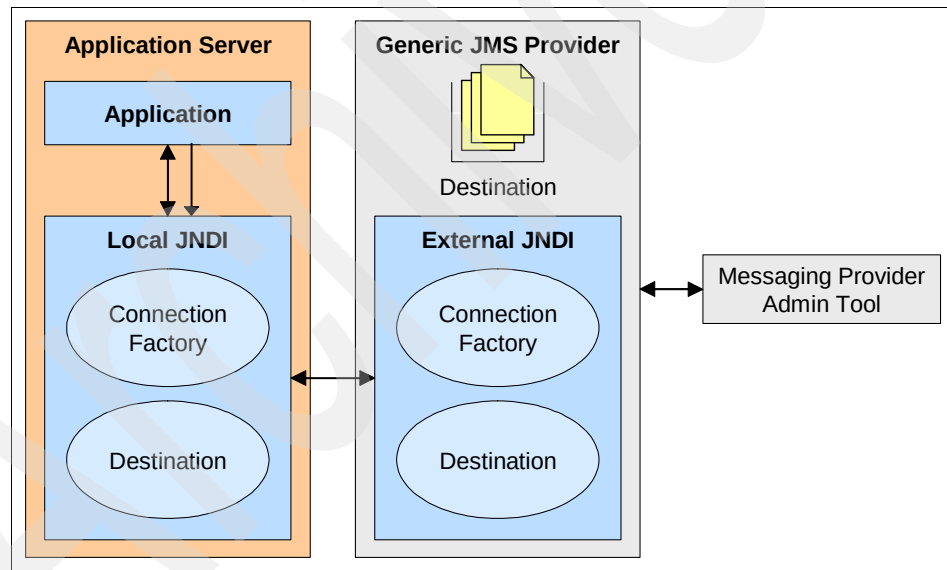


Figure 8-24 Generic JMS provider components

This indirection is achieved by providing additional JNDI information when configuring the JMS administered objects for the generic JMS provider. JMS client application code is not affected in any way. It is the responsibility of the WebSphere run time to resolve accesses to the real JNDI entries in the external name space.

However, WebSphere is not responsible for binding the JMS administered objects into the external name space. This administrative task, along with creating the underlying messaging objects, queues, and topics, must be performed using the tools provided by the generic JMS provider.

Defining a generic JMS provider

Before you can configure a generic JMS provider, you must install the underlying messaging provider software and configure it using the tools and information provided with the messaging provider.

To define a new generic messaging provider, use the administrative console to complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS** → **JMS Providers**.
2. Set the scope at which to define the generic JMS provider by using the relevant controls. Any existing generic JMS providers defined at this scope are displayed in the content pane.
3. Click **New** in the content pane.

4. Define the JMS provider by specifying the appropriate values in the **General Properties** section of the content pane, shown in Figure 8-25 on page 460. The properties are described in Table 8-9 on page 461.

The screenshot shows a configuration window titled "Configuration". Inside, there is a section titled "General Properties" with several input fields: "Scope" (containing "Node=ITSOCellNode"), "* Name" (empty), "Description" (empty), "Class path" (empty), "Native library path" (empty), "* External initial context factory" (empty), and "* External provider URL" (empty). To the right of these fields is a list of "Additional Properties" with checkboxes: "Connection factories", "Queue connection factories", "Topic connection factories", "Queues", "Topics", and "Custom properties". A note states: "The additional properties will not be available until the general properties for this item are applied or saved." At the bottom are buttons for "Apply", "OK", "Reset", and "Cancel".

Configuration

General Properties

Scope
Node=ITSOCellNode

* Name

Description

Class path

Native library path

* External initial context factory

* External provider URL

Additional Properties

- Connection factories
- Queue connection factories
- Topic connection factories
- Queues
- Topics
- Custom properties

The additional properties will not be available until the general properties for this item are applied or saved.

Apply OK Reset Cancel

Figure 8-25 Generic JMS provider configuration properties

Table 8-9 Generic JMS provider properties

Property	Description
Scope	The scope of the generic JMS provider.
Name	The name by which the generic JMS provider is known for administrative purposes.
Description	A description of the generic JMS provider, for administrative purposes within IBM WebSphere Application Server.
Class path	The list of paths or JAR file names that together form the location for the generic JMS providers classes.
Native library path	An optional path to any native libraries (.dll's, .so's) required by the generic JMS provider.
External initial context factory	This property is the Java classname of the generic JMS providers initial context factory. For example, this would be the <code>com.swiftmq.jndi.InitialContextFactoryImpl</code> for the SwiftMQ JMS provider.
External provider URL	This is the JMS provider URL for external JNDI lookups. The external provider URL specifies how the initial context factory should connect to the external naming service. The format of the external provider URL is <code><protocol>://<host name>:<port number></code> . Continuing with the example above, the provider URL <code>smqp://localhost:4001</code> indicates that the initial context factory connects to the SwiftMQ naming service using port 4001 on the local machine and using the smqp protocol.

5. Click **OK**.

6. Save the changes and synchronize them with the nodes.

Once the generic JMS provider has been defined, JMS administered objects can be configured for it. This is discussed in 8.6.5, "Configuring a generic JMS provider" on page 515.

8.6 Configuring WebSphere JMS administered objects

As discussed earlier, an administrator must configure JMS administered objects before they can be used within a JMS client application. JMS administered objects are configured using the WebSphere administrative console. The sections that follow discuss the properties exposed by the JMS administered objects supported by WebSphere.

8.6.1 Common administration properties

All of the JMS administered objects that can be configured within WebSphere Application Server expose a subset of properties that are common. These properties are used by WebSphere for administrative purposes. For example, the name and description properties are used for display purposes within the WebSphere administrative console. These common administration properties are shown in Table 8-10 on page 462.

Table 8-10 Common administration properties

Property	Description
Scope	This is the scope of the configured JMS administered object within the cell. The value of this property specifies the level at which this resource definition is visible to applications.
Provider	This is the name of the JMS provider associated with the JMS administered object.
Name	This property is the name by which the JMS administered object is known for administrative purposes.
JNDI name	The JNDI name is used to bind the JMS administered object into the application server's JNDI name space.
Description	This is an optional description for the JMS administered object.
Category	This is an optional category string to use when classifying or grouping the JMS administered object.

8.6.2 Configuring the default messaging JMS provider

The sections that follow describe how to configure connection factories and destinations for the default messaging JMS provider.

JMS connection factory properties

A JMS connection factory is used to create connections to a service integration bus. These connections form part of the common interfaces described in 8.2.3, “JMS domains” on page 407 and can be used by a JMS client to interact with a service integration bus using both the Point-to-Point and Publish/Subscribe messaging models. To remain compatible with JMS specification Version 1.0, there are two specific types of connection factories (prefixed with “Queue” and “Topic”) and a more general type of connection factory with no prefix. All three are configured in exactly the same way with minor exceptions noted below.

The sections that follow describe the properties of the JMS connection factory for the default messaging JMS provider. These properties have been grouped as follows:

- ▶ Connection properties
- ▶ Durable subscription properties
- ▶ Quality of service properties
- ▶ Advanced messaging properties
- ▶ Advanced administrative properties

Connection properties

A connection to a service integration bus is a connection to an individual messaging engine that is part of that bus. The connection properties for a connection factory determine to which messaging engine a JMS client connects. These connection properties provide an administrator with a range of possibilities when configuring a connection factory, from simply connecting to any suitable messaging engine within the named service integration bus, to using a highly specific messaging engine selection algorithm.

It is worth noting that, in its simplest form, the only connection property that must be specified is the name of the service integration bus with which to connect. It is anticipated that, in the majority of cases, a connection factory configured in such a way is suitable for the needs of most applications. For this reason, only a brief description of the connection properties is included here. For an in depth discussion of the connection properties and how they can be used to control messaging engine selection, refer to 8.7, “Connecting to a service integration bus” on page 520.

A brief description of the connection properties for a default messaging JMS provider connection factory are shown in Table 8-11.

Table 8-11 JMS connection factory connection properties

Property	Description
Bus name	This property is the name of the service integration bus to which to connect. The connection factory creates JMS connections to this service integration bus.
Target	This property specifies the name of a target that identifies a group of messaging engines.
Target type	This property specifies the type of target named in the Target property. If no target is specified, this property is ignored. The default value for this property is Bus member name, indicating that the target property specifies the name of a bus member.
Target significance	This property specifies whether it is required that the messaging engine selected is part of the named target group, or whether it is only preferred. If no target is specified, this property is ignored. The default value for this property is Preferred.
Target inbound transport chain	This property identifies the transport chain used by the JMS client when connecting remotely to a messaging engine. Only messaging engines that have this transport chain available are considered for selection. If no value is specified, the connection factory defaults to using the InboundBasicMessaging transport chain.
Provider endpoints	This property specifies a comma separated list of endpoints used by a JMS client to connect to a bootstrap server. It is only necessary to specify a provider endpoint list if the JMS client is not running within the WebSphere Application Server environment, or if the target bus is defined within another cell. For more information, see 8.7, “Connecting to a service integration bus” on page 520.
Connection proximity	This property defines the proximity of messaging engines that can accept connection requests, in relation to the JMS client or the bootstrap server.

Durable subscription properties

The default messaging JMS provider supports the concept of durable subscriptions, as required by the JMS specification. The durable subscription properties for a connection factory configure this support. These properties are described in Table 8-12.

Table 8-12 JMS connection factory durable subscription properties

Property	Description
Client identifier	JMS clients must provide a unique identifier when attempting to register a durable subscription. This identifier is used by the messaging provider to associate messages with a JMS client while it is inactive. When the JMS client becomes active again, it subscribes to the durable subscription, passing the same unique identifier. The messaging provider is then able to deliver persisted messages to the correct client. The unique identifier can either be provided programatically by a JMS client running inside the J2EE Client Container, or administratively by the connection factory. The client identifier property enables an administrator to specify the identifier that should be assigned to connections created by the connection factory. This identifier is then used if the JMS client attempts to register a durable subscription without programmatically providing a client identifier.
Durable subscription home	Messages that are published to a topic that has inactive durable subscribers registered must be stored by the messaging provider and delivered to each subscriber as and when they become active. The durable subscription home property enables an administrator to specify which messaging engine is responsible for persisting such messages. A suitable messaging engine must be specified in order to enable JMS clients to use durable subscriptions.

Quality of service properties

The JMS specification supports two modes of delivery for JMS messages: persistent and non-persistent. However, the service integration bus defines several levels of reliability that can be applied to both persistent and non-persistent messages. The levels of reliability defined by the service integration bus are discussed in more detail in “Reliability” on page 551. The quality of service properties enable an administrator to define the reliability applied to messages sent using connections created from this connection factory. These properties are described in Table 8-13.

Table 8-13 JMS connection factory quality of service properties

Property	Description
Nonpersistent message reliability	Reliability should be applied to non-persistent JMS messages sent using connections created from this connection factory. Different reliability options can be specified for individual destinations by setting the value of this property to As bus destination . The reliability is then defined by the reliability properties specified on the underlying bus destination to which the JMS destination is assigned. The default value for this property is Express nonpersistent .
Persistent message reliability	Reliability should be applied to persistent JMS messages sent using connections created from this connection factory. Different reliability options can be specified for individual destinations by setting the value of this property to As bus destination . The reliability is then defined by the reliability properties specified on the underlying bus destination to which the JMS destination is assigned. The default value for this property is Reliable nonpersistent .

Advanced messaging properties

The connection factory for the default messaging JMS provider also exposes a number of properties for advanced JMS users. These properties are described in Table 8-14 on page 467.

Note: The property “Temporary topic name prefix” does not appear when configuring a specific queue connection factory. In the same vein, the property “Temporary queue name prefix” does not appear when configuring a specific topic connection factory. Both properties will appear when configuring a non-specific connection factory.

Table 8-14 JMS connection factory advanced messaging properties

Property	Description
Read ahead	Read ahead is an optimization technique used by the default messaging JMS provider to reduce the time taken to satisfy requests from message consumers. It works by preemptively assigning messages to message consumers. Messages assigned to message consumers are locked on the server and sent to a proxy destination on the client, prior to the message consumer requesting them. The message consumer running within the client is then able to consume the messages from the local proxy destination. Messages that are locked on the server cannot be consumed by any other message consumers for that destination. Messages that are assigned to a message consumer, but not consumed before it is closed, are subsequently unlocked on the server and are then available for receipt by other message consumers. Valid values for this property are: ► Default Read ahead is enabled in situations where there can only be a single message consumer. That is, read ahead is enabled for message consumers on non-durable subscriptions and unshared durable subscriptions. This is the default value for this property. ► Enabled Read ahead is enabled for all message consumers. ► Disabled Read ahead is disabled for all message consumers. The read ahead property for the connection factory can be overridden by specifying a value for the read ahead property on a specific JMS destination.
Temporary queue name prefix	Enter the prefix to be used when generating the names of temporary queues created within JMS clients using this connection factory. The prefix can be up to twelve characters long. By default, no value is specified for this property, which causes temporary queues to be generated without any prefix.
Temporary topic name prefix	Enter the prefix to be used when generating the names of temporary topics created within JMS clients using this connection factory. The prefix can be up to twelve characters long. By default, no value is specified for this property, which causes temporary topics to be generated without any prefix.

Property	Description
Share durable subscriptions	This property specifies whether multiple TopicSubscribers, created using this connection factory, can consume messages simultaneously from a single durable subscription. Normally, only one session at a time can have a TopicSubscriber for a particular durable subscription. This property enables you to override this behavior, to enable a durable subscription to have multiple simultaneous consumers. Valid values for this property are: ► In cluster Allow sharing of durable subscriptions when connections are made from within a server cluster. This is the default value for this property. ► Always shared Share durable subscriptions across connections. ► Never shared Never share durable subscriptions across connections.

Advanced administrative properties

The connection factory for the default messaging JMS provider also exposes a number of advanced properties that are used for administrative purposes. These properties are described in Table 8-15.

Table 8-15 JMS connection factory advanced administrative properties

Property	Description
Component-managed authentication alias	Specify the J2C authentication data entry alias to be used to authenticate the creation of a new connection to the JMS provider. The alias encapsulates the user ID and password that will be used to authenticate the creation of the connection. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.
Log missing transaction contexts	Specify whether the Web or EJB container logs the fact that there is no transaction context associated with the thread on which a connection is obtained. This situation can occur if an application has created its own threads. The log entry is written to the SystemOut.log file. The default value for this property is false. The check box is not selected.

Property	Description
Manage cached handles	Specify whether the Web or EJB container tracks connection handles that have been cached by an application. An application caches connection handles by storing them in instance variables. If the application subsequently fails, the Web or EJB container will attempt to close any connections that it was using. However, tracking cached connection handles incurs a large run time performance overhead and should only be used for debugging purposes. The default value for this property is false (the check box is not selected).
Share data source with CMP	Use this property to enable the sharing of JDBC connections between the data store component of a messaging engine and container-managed persistence (CMP) entity beans. In order for this to provide a performance improvement, the data source used by the data store and the CMP entity bean must be the same. If this is the case, a JDBC connection can be shared within the context of a global transaction involving the messaging engine and the CMP entity bean. If no other resources are accessed as part of the global transaction, WebSphere is able to use local transaction optimization in an effort to improve performance. The default value for this property is false (the check box is not selected). Please refer to the WebSphere Information Center for a full description of this performance optimization.
XA recovery authentication alias	Specify the J2C authentication data entry alias to be used to authenticate the creation of a connection to the JMS provider during XA recovery processing. The alias encapsulates the user ID and password that will be used to authenticate the creation of the connection. During XA recovery processing, a connection might need to be made to a messaging engine within the service integration bus. If security is enabled for the bus, it might be necessary to authenticate the creation of the connection. The XA recovery authentication alias is used for this purpose.

JMS connection factory configuration

To configure a JMS connection factory for the default messaging JMS provider, complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS** → **Connection factories**.

- Set the scope. A list of any existing JMS connection factories defined at this scope will be displayed. This is shown in Figure 8-26.

Connection factories

Use this page to create connections to the associated JMS provider for JMS destinations.

⊕ Scope: Cell=ITSOCell, Node=ITSOCellNode

⊕ Preferences

New Delete

Ⓜ Ⓜ Ⓜ Ⓜ

Select	Name	JNDI name	Provider	Description	Scope
☐	BankJMSConnFactory	jms/BankJMSConnFactory	Default messaging provider	WebSphere Bank JMS Sample Connection Factory	Node=ITSOCellNode

Total 1

Figure 8-26 Default messaging JMS connection factory administered objects

In this example, we already have one JMS connection factory object defined, called BankJMSConnFactory. This connection factory object has all of the necessary properties configured in order to connect to a service integration bus.

- To create a new JMS connection factory object, click **New** and select the JMS provider. Alternatively, to change the properties of an existing JMS connection factory, click one of the connection factories displayed. Figure 8-27 on page 471 shows the top portion of the configuration page for the BankJMSConnFactory object.

Other than the standard JMS administered object properties, Name and JNDI name, the only property that we must specify a value for is Bus name. In Figure 8-27, the value specified for the Bus name property is SamplesBus. This specifies that the BankJMSConnFactory object will create connections to the SamplesBus service integration bus.

Configuration

General Properties

Administration

Scope
Node=ITSOCellNode

Provider
Default messaging provider

* Name
BankJMSConnFactory

* JNDI name
jms/BankJMSConnFactory

Description
WebSphere Bank JMS Sample Connection Factory

Category

Connection

* Bus name
ITSONodeSamplesBus

Target

Additional Properties

- [Connection pool properties](#)

Related Items

- [JAAS - J2C authentication data](#)
- [Buses](#)

Figure 8-27 Default messaging JMS connection factory properties

4. Enter the required configuration properties for the JMS connection factory.
5. Click **OK**.
6. Save the changes and synchronize them with the nodes.
7. For the changes to become effective, any application servers within the scope of the resources will need to be restarted.

JMS destination properties

Both queue and topic destinations can be configured for the default messaging JMS provider. The sections that follow describe the properties of the queue and topic destinations. These properties have been grouped as follows:

- ▶ Common connection properties
- ▶ Queue specific connection properties
- ▶ Topic specific connection properties
- ▶ Advanced destination properties

Common connection properties

JMS queue and JMS topic destinations share a number of common connection properties. These common properties are described in Table 8-16.

Table 8-16 JMS destination connection properties

Property	Description
Bus name	Use this property to specify the name of the service integration bus on which the destination is defined. The default behavior, if no value is specified for this property, is to assume that the destination is defined on the same service integration bus to which the application is connected. That is, the service integration bus will be determined from the connection factory that is used in conjunction with this JMS destination. The only situation in which a bus name must be specified is if the underlying destination that this JMS destination refers to is defined on a foreign bus. The foreign bus specified can refer to a service integration bus, or to WebSphere MQ. Please refer to 9.1.7, “Foreign buses” on page 555 for more information.

Property	Description
Delivery mode	Use this property to specify the delivery mode to be used for messages that are sent to this destination. This property allows an administrator to override the delivery mode specified by the JMS client when sending a message. Valid values for this property are: ► Application The persistence of messages sent to this destination is determined by the JMS client application when sending a message. This is the default value for this property. ► Nonpersistent All messages that are sent to this destination are treated as non-persistent. ► Persistent All messages that are sent to this destination are treated as persistent.
Time to live	Specify the length of time, in milliseconds, from its dispatch time that a message sent to this destination should be kept by the system. Specifying a time to live on a destination overrides the time to live specified by the JMS client when sending a message. A value of 0 (zero) means that messages are kept indefinitely. By default, no value is specified for this property, allowing the JMS client application to determine the time to keep messages.
Priority	Specify the relative priority for messages sent to this destination. Specifying a priority on a destination overrides the priority specified by the JMS client when sending a message. The JMS specification defines ten levels of priority ranging from 0 (zero) to 9. Zero is the lowest priority and 9 is the highest. By default, no value is specified for this property, allowing the JMS client application to determine the priority for a message. If the JMS client application does not specify a priority, the default JMS priority of 4 will be used.

Queue specific connection properties

The property that is specific to JMS queue destinations are described in Table 8-17 on page 474.

Table 8-17 JMS queue specific connection properties

Property	Description
Queue name	Use this property to specify the name of the queue destination on the underlying service integration bus or foreign bus. If this JMS destination refers to a destination defined on WebSphere MQ, through a foreign bus, special consideration must be given to the queue name specified. Refer to “Addressing destinations across the WebSphere MQ link” on page 587 for more information.

Topic specific connection properties

When configuring a JMS topic destination, it is possible to partition the topic space into a tree-like hierarchical structure. You can achieve this by defining multiple JMS topic destinations that refer to the same underlying topic space destination, but specifying different topic names. It is the topic name property on a JMS topic destination that is used to partition a topic space.

The topic name property also allows the use of wildcard characters. Figure 8-18 on page 474 describes the wildcard characters that can be used when specifying the topic name.

Table 8-18 Service integration bus topic wildcard characters

Topic name	Topics selected
A/B	Selects the B child of A.
A/*	Selects all children of A.
A//*	Selects all descendents of A.
A//.	Selects A and all descendents of A.
//*	Selects everything.
A./B	Equivalent to A/B.
A/*/B	Selects all B grandchildren of A.
A//B	Selects all B descendents of A.
//A	Selects all A elements at any level.
*	Selects all first level elements.

Note: The use of wildcards within a topic name for a JMS topic destination is only valid when the JMS topic destination is used by a message consumer. If a message producer attempts to use such a JMS topic destination, a JMS exception will be thrown to the JMS client application.

Refer to the WebSphere Information Center for a full description of using topic wildcards in topic expressions to retrieve topics provided by the default messaging provider and service integration bus.

The properties that are specific to JMS topic destinations are described in Table 8-19.

Table 8-19 JMS topic specific connection properties

Property	Description
Topic space	Use this property to specify the name of the topic space destination on the underlying service integration bus.
Topic name	The topic name property allows a topic space to be partitioned into a tree-like hierarchical structure. Several JMS topic destinations can be defined that refer to different nodes of this tree structure within the same underlying topic space on a service integration bus. By default, no value is specified for this property. In this situation, the topic name will default to the value specified for the Name property for this JMS topic destination.

Advanced destination properties

The JMS queue and JMS topic destinations for the default messaging JMS provider also exposes the advanced properties described in Table 8-20.

Table 8-20 JMS destination advanced properties

Property	Description
Read ahead	The read ahead property on a JMS destination enables an administrator to override the value of the read ahead property specified on the JMS connection factory. Valid values for this property are: ► Enabled Read ahead is enabled for all message consumers that are consuming messages from this destination. ► Disabled Read ahead is disabled for all message consumers that are consuming messages from this destination. ► Inherit from connection factory The value of the read ahead property specified on the JMS connection factory should be used. For information about the read ahead property, refer to Table 8-14 on page 467.

JMS queue configuration

To configure a queue destination for the default messaging JMS provider, complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS** → **Queues**.
2. Set the scope. A list of any existing queue destinations defined at this scope will be displayed. This is shown in Figure 8-28.

Queues

A JMS queue is used as a destination for point-to-point messaging.

⊕ Scope: Cell=ITSOCell, Node=ITSOCellNode

⊕ Preferences

New Delete

⊞ ⊞ ⊞ ⊞

Select	Name ↕	JNDI name ↕	Provider ↕	Description ↕	Scope ↕
☐	BankJMSQueue	jms/BankJMSQueue	Default messaging provider	WebSphere Bank Sample Queue (WebSphere Bank receives a message from this queue)	Node=ITSOCellNode
Total 1					

Figure 8-28 Default messaging queue destination administered objects

In this example, we already have one JMS queue destination object defined, called BankJMSQueue.

3. To create a new queue destination object, click **New**. Alternatively, to change the properties of an existing queue destination, click one of the queue destinations displayed. Figure 8-29 on page 478 shows part of the configuration page for the BankJMSQueue object.

Other than the standard JMS administered object properties, Name and JNDI name, the only property that we must specify a value for is Queue name. In Figure 8-29 on page 478, the value specified for the Queue name property is BankJSQueue. This must match the name of the queue destination defined on the corresponding service integration bus.

By default, no value is specified for the Bus name property. The default behavior when no bus name is specified is to assume that the queue destination is defined on the same service integration bus to which the application is connected. That is, the service integration bus will be determined from the connection factory that is used in conjunction with the JMS queue destination.

Configuration

General Properties

Administration

Scope

Node=ITSOCellNode

Provider

Default messaging provider

* Name

BankJMSQueue

* JNDI name

jms/BankJMSQueue

Description

WebSphere Bank Sample Queue
(WebSphere Bank receives a message from this queue)

Connection

Bus name

ITSONodeSamplesBus

* Queue name

BankJMSQueue

Delivery mode

Application

Time to live

milliseconds

Figure 8-29 Default messaging queue destination properties

4. Enter the required configuration properties for the JMS queue destination.
5. Click **OK**.
6. Save the changes and synchronize them with the nodes.
7. For the changes to become effective, restart any application servers within the scope of the resources.

JMS topic configuration

To configure a topic destination for the default messaging JMS provider, complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS** → **Topics**.
2. Set the scope. A list of any existing topic destinations defined at this scope will be displayed. This is shown in Figure 8-30.

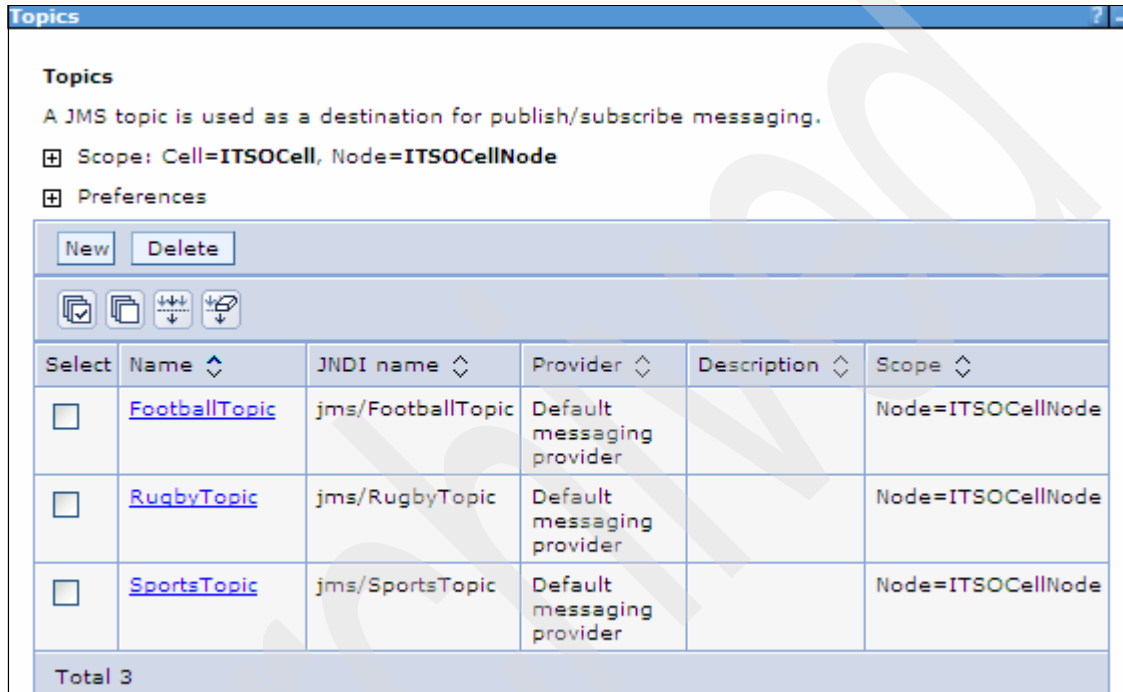


Figure 8-30 Default messaging topic destination administered objects

In this example, we already have three JMS topic destination objects defined, FootballTopic, RugbyTopic, and SportsTopic.

3. To create a new topic destination object, click **New**. Alternatively, to change the properties of an existing topic destination, click one of the topic destinations displayed. Figure 8-31 on page 480 shows part of the configuration page for the FootballTopic object.

Other than the standard JMS administered object properties, Name and JNDI name, the only property that we must specify a value for is Topic space. In Figure 8-31, the value specified for the Topic space property is SportsTopic. This must match the name of the topic space destination defined on the corresponding service integration bus.

The screenshot displays the 'Configuration' page for the 'FootballTopic' object. It is divided into two main sections: 'General Properties' and 'Connection'.

General Properties

- Administration**
 - Scope**: Node=ITSOCellNode
 - Provider**: Default messaging provider
 - Name**: FootballTopic
 - JNDI name**: jms/FootballTopic
 - Description**: (Empty text area)

Connection

- Topic name**: sports/football
- Bus name**: ITSONodeSamplesBus
- Topic space**: SportsTopic
- JMS delivery mode**: Application

Figure 8-31 Default messaging topic destination properties

By default, no value is specified for the Bus name property. The default behavior when no bus name is specified is to assume that the topic destination is defined on the same service integration bus to which the application is connected. The service integration bus will be determined from the connection factory that is used in conjunction with the JMS topic destination.

It is also worth noting that the Topic name property shown in Figure 8-31 has a value of sports/football. The topic name property allows a topic space to be partitioned into a tree-like hierarchical structure. The three JMS topic destinations shown in Figure 8-30 on page 479 all refer to the SportsTopic destination on the underlying service integration bus. However, they all specify different topic names, as shown in Table 8-21.

Table 8-21 Sample sports topic names

JMS topic destination	Topic name
SportsTopic	sports/*
FootballTopic	sports/football
RugbyTopic	sports/rugby

Effectively, this configuration partitions the SportsTopic topic space into the hierarchical structure shown in Figure 8-32.

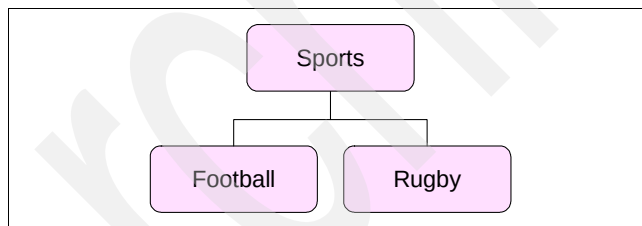


Figure 8-32 Sample sports topic hierarchy

If a subscriber subscribes to the FootballTopic JMS destination, which represents the sports/football topic name, it will only receive publications sent using the FootballTopic JMS destination, that map on to the same topic name.

However, the SportsTopic JMS destination defines a topic name that ends with a wildcard character. This allows a subscriber interested in all sports to subscribe to the SportsTopic destination. This subscriber would then receive publications sent to either the FootballTopic or RugbyTopic JMS destinations.

See “Topic specific connection properties” on page 474 for more information about using wild cards.

4. Enter the required configuration properties for the JMS topic destination.
5. Click **OK**.
6. Save the changes and synchronize them with the nodes.
7. For the changes to become effective, restart any application servers within the scope of the resources.

JMS activation specification properties

As we discussed in 8.4.7, “Associating a message-driven bean with a destination” on page 445, a JMS activation specification is used to configure an instance of an ActivationSpec JavaBean for the default messaging JMS provider. A JMS activation specification is then associated with a message-driven bean during application installation.

The JMS activation specification object defines all of the properties that the J2EE Connector Architecture requires or recommends an ActivationSpec JavaBean to support. For more information about these properties, please refer to 8.3.4, “JMS ActivationSpec JavaBean” on page 428. It also defines other properties specific to using it in conjunction with a service integration bus.

The sections that follow describe the properties of the JMS activation specification. These properties have been grouped as follows:

- ▶ Destination properties
- ▶ Additional properties
- ▶ Subscription durability properties
- ▶ Advanced properties

Note: JMS activation specifications also expose the following administration properties:

- ▶ Scope
- ▶ Provider
- ▶ Name
- ▶ JNDI name
- ▶ Description

A description for these properties can be found in 8.6.1, “Common administration properties” on page 462.

Destination properties

The JMS activation specification defines a number of properties that identify the destination with which a message-driven bean will be associated. These properties are described in Table 8-22 on page 483.

Table 8-22 JMS activation specification destination properties

Property	Description
Destination type	Use this property to specify the type of the JMS destination with which a message-driven bean will be associated. Valid values for this property are: ► Queue The target destination is a queue destination. This is the default value for this property. ► Topic The target destination is a topic destination
Destination JNDI name	You must specify a JNDI name for the target destination.
Message selector	This property specifies a JMS message selector that should be applied to the target JMS destination. Only messages that match this message selector will be delivered to the message-driven bean. By default, no message selector is specified for a JMS activation specification. Refer to “Message selectors” on page 414 for more information.
Bus name	This property is the name of the service integration bus on which the target destination is defined. This bus must exist within the same cell as the application server on which the message-driven bean is running, but this application server is not required to be a member of the bus. However, the best performance will be obtained if the application server on which the message-driven bean is running is a member of the bus specified. A value must be specified for this property.
Acknowledge mode	Use this property to specify how the EJB container acknowledges the receipt of a message by a message-driven bean instance that is using bean managed transactions. Valid values for this property are: ► Auto-acknowledge The EJB container automatically acknowledges the delivery of a message when the onMessage method of the message-driven bean successfully returns. ► Duplicates-ok auto-acknowledge The EJB container lazily acknowledges the delivery of messages to message-driven beans. This can improve performance, but can lead to a message-driven bean receiving a message more than once.
Target	This property specifies the name of a target that identifies a group of messaging engines.

Property	Description
Target Type	This property specifies the type of target named in the Target property. If no target is specified, this property is ignored. The default value for this property is Bus member name, indicating that the target property specifies the name of a bus member.
Target Significance	This property specifies whether it is required that the messaging engine selected is part of the named target group, or whether it is only preferred. If no target is specified, this property is ignored. The default value for this property is Preferred.
Target inbound transport chain	This property identified the transport chain used by the JMS client when connecting remotely to a messaging engine. Only messaging engines that have this transport chain available are considered for selection. If no value is specified, the connection factory defaults to using the InboundBasicMessageing transport chain.

Additional properties

The JMS activation specification for the default messaging JMS provider also exposes a group of additional properties, as described in Table 8-23 on page 484.

Table 8-23 JMS activation specification additional properties

Property	Description
Authentication alias	Use this property to specify the J2C authentication data entry alias to be used to authenticate the creation of a new connection to the JMS provider. The alias encapsulates the user ID and password that will be used to authenticate the creation of the connection.
Maximum batch size	Specify the maximum number of messages that can be received from a messaging engine in a single batch. These messages are then delivered serially to an instance of the message-driven bean that is associated with this JMS activation specification. Delivering messages in a batch can improve the performance of the JMS application. However, if message ordering must be maintained across failed deliveries, the batch size should be set to 1. If no value is specified for this property, it defaults to 1.

Property	Description
Maximum concurrent endpoints	This property specifies the maximum number of message endpoints to which messages are delivered concurrently. In the case of a JMS activation specification, a message endpoint is a JMS message-driven bean. Increasing this number can improve performance but will also increase the number of running threads within the application server. If message ordering must be maintained across failed deliveries, the number of maximum concurrent endpoints should be set to 1. If no value is specified for this property, it defaults to 10.

Subscription durability properties

A JMS activation specification can be configured with a destination type of Topic. It might be required that message-driven beans that are associated with such a JMS activation specification need to register durable subscriptions with the topic destination. However, a message-driven bean is not able to programatically configure a durable subscription. The subscription durability properties on a JMS activation specification enable the configuration properties for a durable subscription to be specified administratively. These properties are described in Table 8-24 on page 485.

Table 8-24 JMS activation specification subscription durability properties

Property	Description
Subscription durability	Use this property to specify whether a JMS topic subscription is durable or nondurable. Valid values for this property are: ► Durable The messaging provider stores messages while the message-driven bean is not available, and delivers the messages when the message-driven bean becomes available again. ► Nondurable The messaging provider does not store and redeliver messages if a message-driven bean is not available. This is the default value for this property.

Property	Description
Subscription name	JMS clients must provide a subscription name when attempting to register a durable subscription. Because a JMS client can create several durable subscriptions, the subscription name must be unique within the context of a particular client identifier (described within this table). A message-driven bean is not able to programatically specify a subscription name when it creates a durable subscription. A suitable subscription name must be specified in order to enable message-driven beans associated with this JMS activation specification to use durable subscriptions.
Client identifier	JMS clients must provider a unique identifier when attempting to register a durable subscription. This identifier is used by the messaging provider to associate messages with a JMS client while it is inactive. When the JMS client becomes active again, it subscribes to the durable subscription, passing the same unique identifier. The messaging provider is then able to deliver persisted messages to the correct client. A message-driven bean is not able to programatically specify a client identifier when it creates a durable subscription. A suitable client identifier must be specified in order to enable message-driven beans associated with this JMS activation specification to use durable subscriptions.
Durable subscription home	Messages that are published to a topic that has inactive durable subscribers registered must be stored by the messaging provider and delivered to each subscriber as and when they become active. The durable subscription home property enables an administrator to specify which messaging engine is responsible for persisting such messages. A suitable messaging engine must be specified in order to enable message-driven beans associated with this JMS activation specification to use durable subscriptions.

Advanced properties

The JMS activation specification for the default messaging JMS provider also exposes the advanced properties described in Table 8-25 on page 487.

Table 8-25 JMS activation specification advanced properties

Property	Description
Share durable subscriptions	The share durable subscriptions property for the JMS activation specification defines whether a durable subscription should be shared across connections. This property is only relevant if the value of the destination type property is Topic and the value of the subscription durability property is Durable. The default value for this property is In cluster. Refer to Table 8-14 on page 467 for more information.
Share data source with CMP	Use this property to enable the sharing of JDBC connections between the data store component of a messaging engine and container-managed persistence (CMP) entity beans. In order for this to provide a performance improvement, the data source used by the data store and the CMP entity bean must be the same. If this is the case, a JDBC connection can be shared within the context of a global transaction involving the messaging engine and the CMP entity bean. If no other resources are accessed as part of the global transaction, WebSphere is able to use local transaction optimization in an effort to improve performance. The default value for this property is false (the check box is not selected). Please refer to the WebSphere Information Center for a full description of this performance optimization.
Read ahead	Read ahead is an optimization technique used by the default messaging JMS provider to reduce the time taken to satisfy requests from message consumers. It works by preemptively assigning messages to message consumers. Messages assigned to message consumers are locked on the server and sent to a proxy destination on the client, prior to the message consumer requesting them. The message consumer running within the client is then able to consume the messages from the local proxy destination. Messages that are locked on the server cannot be consumed by any other message consumers for that destination. Messages that are assigned to a message consumer, but not consumed before it is closed, are subsequently unlocked on the server and are then available for receipt by other message consumers. Refer to Table 8-14 on page 467 for more information.

JMS activation specification configuration

To configure a JMS activation specification for the default messaging JMS provider, complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS** → **Activation specifications**.
2. Set the scope. A list of any existing activation specifications defined at this scope will be displayed. This is shown in Figure 8-33 on page 488.

Select	Name	JNDI name	Provider	Description	Scope
☐	BankActivationSpec	eis/BankActivationSpec	Default messaging provider	JMS Activation Specification for WebSphere Bank	Node=ITSOCellNode
Total 1					

Figure 8-33 Default messaging JMS activation specifications

In this example, we already have one JMS activation specification object defined, called BankActivationSpec.

3. To create a new JMS activation specification object, click **New**. Alternatively, to change the properties of an existing JMS activation specification, click one of the JMS activation specifications displayed. Figure 8-31 on page 480 shows the top portion of the configuration page for the BankActivationSpec object.

The JMS activation specification object is not, strictly speaking, a JMS administered object. However, it still exposes a number of the properties that are common among all JMS administered objects. These are scope, provider, name, JNDI name, and description. As with JMS administered objects, values for these properties are required by the WebSphere administrative console for administrative purposes.

Values must also be specified for all of the properties on the ActivationSpec JavaBean that are defined as required within the deployment descriptor for the default messaging resource adapter. Recall from Example 8-13 on page 427 that these properties are destination, destinationType and busName. The relevant mappings between these properties and the corresponding properties on the JMS activation specification are shown in Table 8-26.

Table 8-26 Required properties for a JMS activation specification object

ActivationSpec JavaBean property	JMS activation specification property	BankActivationSpec value
destination	Destination JNDI name	jms/BankJMSQueue
destinationType	Destination type	Queue
busName	Bus name	SamplesBus

Following our example, using the JMS queue defined in “JMS queue configuration” on page 476, we know that the BankJMSQueue object was bound into the JNDI name space with the name `jms/BankJMSQueue`. This JMS queue object maps on to the BankJSQueue on the SamplesBus service integration bus.

Therefore, if a message-driven bean is associated with this JMS activation specification, it would be invoked when messages arrived at the BankJMSQueue destination on the SamplesBus. See Figure 8-34 on page 490.

The screenshot displays the 'Configuration' tab in a web management console. Under the 'General Properties' section, there are two main panels: 'Administration' and 'Destination'. The 'Administration' panel includes fields for 'Scope' (Node=ITSOCellNode), 'Provider' (Default messaging provider), 'Name' (BankActivationSpec), 'JNDI name' (eis/BankActivationSpec), and 'Description' (JMS Activation Specification for WebSphere Bank). The 'Destination' panel includes fields for 'Destination type' (Queue), 'Destination JNDI name' (jms/BankJMSQueue), 'Message selector' (empty), 'Bus name' (ITSONodeSamplesBus), 'Acknowledge mode' (Auto-acknowledge), and 'Target' (empty).

Configuration	
General Properties	
Administration	
Scope	Node=ITSOCellNode
Provider	Default messaging provider
* Name	BankActivationSpec
* JNDI name	eis/BankActivationSpec
Description	JMS Activation Specification for WebSphere Bank
Destination	
* Destination type	Queue
* Destination JNDI name	jms/BankJMSQueue
Message selector	
Bus name	ITSONodeSamplesBus
Acknowledge mode	Auto-acknowledge
Target	

Figure 8-34 Default messaging JMS activation specification properties

4. Enter the required configuration properties for the JMS activation specification.
5. Click **OK**.
6. Save the changes and synchronize them with the nodes.
7. For the changes to become effective, restart any application servers within the scope of the resources.

8.6.3 Configuring the WebSphere MQ JMS provider

The WebSphere MQ JMS provider can be configured to communicate with WebSphere MQ using a bindings or client connection. These two connectivity options are described below:

► Bindings connection

When used in bindings mode, the WebSphere MQ JMS provider uses the Java Native Interface (JNI) to call directly into the existing queue manager API, rather than communicating through a network. This provides better performance when connecting to WebSphere MQ than using a client connection.

However, to use a bindings connection, WebSphere MQ and WebSphere Application Server must be installed on the same machine.

► Client connection

If it is not possible to colocate WebSphere Application Server and WebSphere MQ on the same machine, the WebSphere MQ JMS provider must be configured to connect to WebSphere MQ using TCP/IP. Using a client connection allows you to perform authorization checks.

Additional considerations must be taken into account when configuring the WebSphere MQ JMS provider to use a client connection, for example:

- Whether the connection needs to be secured by encrypting the data that flows over the connection
- Whether the connection will go through a firewall

The sections that follow describe the properties exposed by WebSphere MQ connection factories and destinations, and also how to configure connection factories and destinations for the WebSphere MQ JMS provider.

Note: As discussed in 8.5.2, “Managing the WebSphere MQ JMS provider” on page 455, WebSphere MQ resources, such as queue managers, channels, and queues, must be created using the tools provided with WebSphere MQ.

WebSphere MQ connection factory properties

A WebSphere MQ connection factory is used to create connections to WebSphere MQ. These connections form part of the common interfaces described in 8.2.3, “JMS domains” on page 407 and can be used by a JMS client to interact with WebSphere MQ using both the Point-to-Point and Publish/Subscribe messaging models.

However, because the WebSphere MQ connection factory is not specific to either JMS domain, it encapsulates all of the configuration information that might be required to communicate using either messaging model. Consequently, a large number of properties are exposed by the WebSphere MQ connection factory object. Fortunately, default values are defined for many of these properties.

To remain compatible with JMS specification 1.0, there are two specific types of connection factories (prefixed with “Queue” and “Topic”) and a more general type of connection factory with no prefix. The particular properties of specific types of connection factories will be a subset of the more general connection factory, but all are administered in the same way.

The sections that follow describe some of the more important properties that are exposed by the WebSphere MQ connection factory object. These properties have been grouped as follows:

- ▶ Bindings connection properties
- ▶ Client connection properties
- ▶ Queue connection specific properties
- ▶ Topic connection specific properties
- ▶ Connection security properties
- ▶ Advanced connection properties

Note: Not all of the properties of the WebSphere MQ connection factory are described. For a full description of all of the properties, please refer to the *WebSphere Information Center* and the *WebSphere MQ Using Java* manual, links for which are contained in 8.8, “References and resources” on page 536.

Bindings connection properties

With respect to the number of properties, setting up a bindings connection between a WebSphere MQ connection factory and WebSphere MQ is the simplest configuration. The properties required to configure a bindings connection for a WebSphere MQ connection factory object are shown in Table 8-27.

Table 8-27 WebSphere MQ connection factory bindings connection properties

Property	Description
Transport type	Use this property to specify whether a WebSphere MQ client TCP/IP connection or interprocess bindings connection is to be used to connect to the WebSphere MQ queue manager. Inter-process bindings can only be used to connect to a queue manager on the same physical machine. The transport type defaults to BINDINGS.
Queue manager	This property is the name of the WebSphere MQ queue manager for this connection factory. Connections created by this factory connect to the specified queue manager on the local machine. If no queue manager is specified, the connections created by this factory will connect to the default queue manager on the local machine if one exists.

Client connection properties

The properties required to configure a basic client connection for a WebSphere MQ connection factory object are shown in Table 8-28.

Table 8-28 WebSphere MQ connection factory client connection properties

Property	Description
Transport type	Use this property to specify whether a WebSphere MQ client TCP/IP connection or interprocess bindings connection is to be used to connect to the WebSphere MQ queue manager. To configure a WebSphere MQ client TCP/IP connection, a value of CLIENT must be specified.
Host	This property is the name of the host on which the WebSphere MQ queue manager runs.
Port	This property defines the TCP/IP port number used for connection to the WebSphere MQ queue manager. This port number should match the listener port defined for the queue manager. The default value for the port property is 0 (zero). The default port for a WebSphere MQ queue manager listener is 1414.
Channel	Specify the name of the channel used for connection to the WebSphere MQ queue manager. If no channel is specified, the channel defaults to a standard server connection channel defined by all queue managers called SYSTEM.DEF.SVRCONN.

Property	Description
Local server address	In some network configurations, firewalls are configured to prevent connection attempts unless they originate from specific ports or range of ports. The local server address property allows a port or range of ports to be specified for the WebSphere MQ connection factory to use when creating the outbound client connection. The local server address property defaults to null.

Queue connection properties

A number of the properties defined by the WebSphere MQ connection factory object are specific to WebSphere MQ queue destinations. Table 8-29 on page 494 describes these properties.

Table 8-29 WebSphere MQ connection factory queue connection specific properties

Property	Description
Enable message retention	Check this box to specify that unwanted messages are to be left on the queue. Otherwise, unwanted messages are dealt with according to their disposition options. By default, this means that a message is sent to the queue manager's dead-letter queue. It is also possible to specify that unwanted messages be discarded. The default value for the enable message retention property is true. The box is checked.
Model queue definition	This property is the name of the model queue from which WebSphere MQ dynamic queues are created. The model queue acts a template for the WebSphere MQ dynamic. WebSphere MQ dynamic queues are created as a result of the JMS client invoking the createTemporaryQueue method on the Session object. If no model queue definition is specified, it defaults to a standard model queue defined by all queue managers called SYSTEM.DEFAULT.MODEL.QUEUE.
Temporary queue prefix	The prefix that is used to form the name of a WebSphere MQ dynamic queue. The prefix must end in an asterisk (*) and be no more than 33 characters in length, including the asterisk. If no temporary queue prefix is specified, it defaults to AMQ.*.

Topic connection properties

A large number of the properties defined by the WebSphere MQ connection factory object are specific to WebSphere MQ topic destinations. Table 8-30 describes some of the more important topic connection specific properties.

Table 8-30 WebSphere MQ connection factory topic connection specific properties

Property	Description
Broker queue manager	Use this property to define the name of the WebSphere MQ queue manager that is hosting WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker. This can be different from the value specified for the queue manager property. However, if it is different, server channels must be defined between the two queue managers. If no broker queue manager is specified, it defaults to having the same value as the queue manager property.
Broker control queue	Define the name of the queue on the broker queue manager to which subscription requests should be sent. If no broker control queue is specified, it defaults to a standard control queue on the broker queue manager called <code>SYSTEM.CONTROL.BROKER.QUEUE</code> .
Broker publication queue	This property defines the name of the queue on the broker queue manager to which publications should be sent. If no broker publication queue definition is specified, it defaults to a standard publication queue on the broker queue manager called <code>SYSTEM.BROKER.DEFAULT.STREAM</code> .
Broker subscription queue	Specify the name of the queue on the broker queue manager from which non-durable subscription messages are retrieved. If no broker subscription queue is specified, it defaults to a <code>SYSTEM.JMS.ND.SUBSCRIBER.QUEUE</code> .
Client ID	Define the client identifier used when creating durable subscriptions to a topic. The client identifier is ignored for point-to-point connections.

Connection security properties

Security is an additional consideration when configuring a bindings connection between a WebSphere MQ connection factory and WebSphere MQ. Table 8-31 describes the properties of a WebSphere MQ connection factory that relate to security.

Table 8-31 WebSphere MQ connection factory connection security properties

Property	Description
Component-managed authentication alias	The component-managed authentication alias drop down can be used to specify a J2C authentication data entry. If the resource reference used within the JMS client application specifies a res-auth of Application, the user ID and password defined by the J2C authentication data entry will be used to authenticate the creation of a connection. The component-managed authentication alias defaults to none. If no component-managed authentication alias is specified and the WebSphere MQ queue manager requires the user ID and password to get a connection, then an exception will be thrown when attempting to connect.
SSL cipher suite	Enter the SSL cipher suite used to encrypt the communication with the queue manager. If set, the value of this property must be a valid CipherSuite provided by the JSSE provider configured within WebSphere Application Server. It must also be equivalent to the CipherSpec specified on the server connection channel within WebSphere MQ, named by the CHANNEL property. By default, no value is specified for this property.
SSL CRL	The SSL CRL property specifies zero or more Certificate Revocation List (CRL) servers. These are LDAP servers that are used to check whether a SSL certificate has been revoked. If SSLCRL is not set, which is the default, no such checking is performed. Also, SSL CRL is ignored if no SSL cipher suite is specified.
SSL peer name	The SSL peer name property specifies a distinguished name that must match the SSLPEER parameter specified on the server connection channel named by the CHANNEL property. If the SSL peer name property is not set, which is the default, no such checking is performed. Also, SSL peer name is ignored if no SSL cipher suite is specified.

Advanced connection properties

The WebSphere MQ connection factory object also exposes a number of properties that affect how the WebSphere MQ JMS provider interacts with WebSphere MQ. In order to fully understand these properties, an advanced knowledge of WebSphere MQ is required. Some of the more important properties are described in Table 8-32.

Table 8-32 WebSphere MQ connection factory advanced properties

Property	Description
CCSID	Use this property to define the coded-character-set-ID to be used on connections. The value for this property defaults to null. This indicates to the WebSphere MQ JMS provider that its default CCSID should not be overridden. The default CCSID within the WebSphere MQ JMS provider is 819, which represents the ASCII character set. Changing this value affects the way in which the queue manager that this connection factory creates connections for translates information in the WebSphere MQ headers.
XA enabled	Specify whether the resources of WebSphere MQ can be enlisted into a distributed transaction. The default value for the XA enabled property is true. The box is checked. If the XA enabled check box is not selected, the JMS session is still enlisted in a transaction, but uses the resource manager local transaction calls (session.commit and session.rollback) instead of XA calls. This can lead to an improvement in performance. However, unless last participant support is used, this means that only a single resource can be enlisted in a transaction in WebSphere Application Server.
Enable return methods during shutdown	Define whether a JMS client application returns from a method call if the queue manager has entered a controlled shutdown. The default value for the enable return methods during shutdown property is true (the check box is selected).
Polling interval	The polling interval property specifies the interval, in milliseconds, between scans of all receivers during asynchronous message delivery. The polling interval property defaults to 5000.
Rescan interval	The rescan interval property specifies the interval in milliseconds between which a queue is scanned to look for messages that have been added to a queue out of order. This interval controls the scanning for messages that have been added to a queue out of order with respect to a WebSphere MQ browse cursor. The rescan interval property defaults to 5000.

Property	Description
Enable MQ connection pooling	This property specifies whether MQ connection pooling should be used to pool the connections to the WebSphere MQ queue manager. If MQ connection pooling is used, when a connection is no longer required, instead of destroying it, it can be pooled, and later reused. This can provide a substantial performance enhancement for repeated connections to the same queue manager. The default value for the enable MQ connection pooling property is true. The box is checked.

WebSphere MQ connection factory configuration

To configure a connection factory for the WebSphere MQ JMS provider, complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS** → **Connection factories**.
2. Set the scope A list of any existing connection factories defined at this scope will be displayed. This is shown in Figure 8-35.

Connection factories

Connection factories

Use this page to create connections to the associated JMS provider for JMS destinations.

⊕ Scope: Cell=ITSOCell, Node=ITSOCellNode

⊕ Preferences

NewDelete



Select	Name ↕	JNDI name ↕	Provider ↕	Description ↕	Scope
☐	BankJMSConnFactory	jms/BankJMSConnFactory	Default messaging provider	WebSphere Bank JMS Sample Connection Factory	Node
☐	BankMQJMSConnFactory	jms/BankMQJMSConnFactory	WebSphere MQ messaging provider	JMS MQ Connection Factory for WebSphere Bank	Node

Total 2

Figure 8-35 WebSphere MQ connection factory administered objects

In this example, we already have one WebSphere MQ connection factory object defined called BankMQJMSConnFactory. This connection factory object has all of the necessary properties configured in order to connect to a full WebSphere MQ JMS provider using a client connection. Note the different providers in the list above.

3. To create a new connection factory object, click **New**, specify the type of provider in the next window, and click **Next**. Alternatively, to change the properties of an existing connection factory, click one of the connection factories displayed. Figure 8-36 on page 500 shows the top portion of the configuration page for BankMQJMSConnFactory object.

Other than the standard JMS administered object properties, Name and JNDI name, the only other properties that we must specify values for, in order to configure a client connection to WebSphere MQ, are as follows:

- Transport type

A transport type of CLIENT has been specified to indicate that we will connect to WebSphere MQ using a WebSphere MQ client TCP/IP connection.

- Host

The WebSphere MQ queue manager is running on host kccg1d6.

- Port

The WebSphere MQ queue manager listener is listening on port 1414.

The BankMQJMSConnFactory object uses the default value for the Channel property, which is SYSTEM.DEF.SVRCONN.

Configuration

General Properties

Scope

Node=ITSOCellNode

Provider

WebSphere MQ messaging provider

* Name

BankMQJMSConnFactory

* JNDI name

jms/BankMQJMSConnFactory

Description

JMS MQ Connection Factory for WebSphere Bank

Category

Component-managed authentication alias

(none)

Container-managed authentication alias

(none)

Mapping-configuration alias

DefaultPrincipalMapping

Queue manager

Host

kcgg1d6

Port

1414

Channel

Additional Properties

Custom properties

Connection pool

Session pools

Related Items

JAAS - J2C authentication data

Figure 8-36 WebSphere MQ connection factory properties

4. Enter the required configuration properties for the WebSphere MQ connection factory.
5. Click **OK**.

6. Save the changes and synchronize them with the nodes.
7. For the changes to become effective, restart any application servers within the scope of the resources.

WebSphere MQ destination properties

Both queue and topic destinations can be configured for the WebSphere MQ JMS provider. The sections that follow describe the properties of the queue and topic destinations. These properties have been grouped as follows:

- ▶ Basic destination connection properties
- ▶ Queue specific destination properties
- ▶ Topic specific destination properties
- ▶ Advanced destination properties

Basic destination properties

The WebSphere MQ queue and WebSphere MQ topic destinations share a number of basic common properties. These properties are described in Table 8-33.

Table 8-33 Basic WebSphere MQ destination properties

Property	Description
Persistence	Use this property to specify whether the messages sent to this destination are persistent, non-persistent, or have their persistence defined by the application or queue. The default value for the persistence property is APPLICATION DEFINED. This specifies that the messages on the destination have their persistence defined by the application that put them onto the queue.
Priority	Use this property to specify whether the message priority for this destination is defined by the application, queue, or the Specified priority property. The default value for the priority property is APPLICATION DEFINED. This specifies that the priority of messages on this destination is defined by the application that put them onto the destination.
Specified priority	If the Priority property is set to Specified , the value of this property determines the message priority for messages sent to this destination. Priorities range from 0 (lowest) through 9 (highest).

Property	Description
Expiry	Specify whether the expiry timeout for this destination is defined by the application or the Specified Expiry property, or messages on the destination never expire (have an unlimited expiry timeout). The default value for the expiry property is APPLICATION DEFINED. This specifies that the expiry timeout of messages on this destination is defined by the application that put them onto the destination.
Specified expiry	If the Expiry Timeout property is set to Specified , the value of this property determines the number of milliseconds (greater than 0) after which messages on this destination expire.

Queue specific destination properties

The properties specific to WebSphere MQ queue destination objects are shown in Table 8-34.

Table 8-34 WebSphere MQ queue destination properties

Property	Description
Base queue name	Use this property to specify the name of the queue to which messages are sent, on the queue manager specified by the Base Queue Manager Name property.
Base queue manager name	Specify the name of the WebSphere MQ queue manager to which messages are sent. This queue manager provides the queue specified by the Base queue name property. The default value for this property is null, in which case the queue manager is assumed to be that of the connection factory object used to connect to WebSphere MQ.
Queue manager host	This property is the name of the host on which the WebSphere MQ queue manager runs.
Queue manager port	This property defines the TCP/IP port number used for connection to the WebSphere MQ queue manager. The port number should match the listener port defined for the queue manager. The default value is 0 (zero); however, the default port for a WebSphere MQ queue manager listener is 1414.
Server connection channel name	Specify the name of the channel used for connection to the WebSphere MQ queue manager. If no channel is specified, the channel defaults to a standard server connection channel defined by all queue managers, called SYSTEM.DEF.SVRCONN.

Property	Description
User ID	This property is used in conjunction with the Password property to provide authentication when connecting to the WebSphere MQ queue manager.
Password	This property is used in conjunction with the User ID property to provide authentication when connecting to the WebSphere MQ queue manager.

Topic specific destination properties

The properties specific to WebSphere MQ topic destination objects are shown in Table 8-35.

Table 8-35 WebSphere MQ topic destination properties

Property	Description
Base topic name	Use this property to specify the name of the topic on the underlying queue manager that JMS clients will publish or subscribe to.
Broker durable subscription queue	Define the name of the brokers queue from which durable subscription messages are retrieved. The subscriber specifies the name of the queue when it registers a subscription.
Broker CC durable subscription queue	Specify the name of the brokers queue from which durable subscription messages are retrieved for a ConnectionConsumer.
Enable multicast transport	Indicate whether or not this topic destination uses multicast transport if supported by the connection factory.

Advanced destination properties

The WebSphere MQ queue and WebSphere MQ topic destinations share a number of advanced common properties. These properties are described in Table 8-36.

Table 8-36 Advanced WebSphere MQ destination properties

Property	Description
CCSID	Use this property to identify the coded character set identifier for use with the WebSphere MQ queue manager. This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.
Use native encoding	Indicate whether or the destination should use native encoding, using appropriate encoding values for the Java platform.
Integer encoding	If native encoding is not enabled, select whether integer encoding is normal or reversed.
Decimal encoding	If native encoding is not enabled, select whether decimal encoding is normal or reversed.
Floating point encoding	If native encoding is not enabled, select the type of floating point encoding.
Target client	Indicate whether the receiving application is JMS-compliant or is a traditional WebSphere MQ application.

WebSphere MQ queue destination configuration

To configure a queue destination for the WebSphere MQ JMS provider, complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS** → **Queues**.
2. Set the scope. A list of any existing queue destinations defined at this scope will be displayed. This is shown in Figure 8-37 on page 505.

Queues					
Queues A JMS queue is used as a destination for point-to-point messaging.					
Scope: Cell=ITSOCell, Node=ITSOCellNode					
Preferences					
New Delete    					
Select	Name	JNDI name	Provider	Description	Scope
☐	BankJMSQueue	jms/BankJMSQueue	Default messaging provider	WebSphere Bank Sample Queue (WebSphere Bank receives a message from this queue)	Node=ITSOCell
☐	BankMQJMSQueue	jms/BankMQJMSQueue	WebSphere MQ messaging provider	WebSphere MQ Queue for WebSphere Bank	Node=ITSOCell
Total 2					

Figure 8-37 WebSphere MQ queue destination administered objects

In this example, we already have one WebSphere MQ queue destination object defined, called BankMQJMSQueue. Note the provider types.

3. To create a new queue destination object, click **New**, specify the type of provider in the next window, and click **Next**. Alternatively, to change the properties of an existing queue destination, click one of the queue destinations displayed. Figure 8-38 on page 507 shows the top portion of the configuration page for BankMQJMSQueue object.

Other than the standard JMS administered object properties, Name and JNDI name, the only property that we must specify a value for is Base queue name. The value specified for the Base queue name property must match the name of the queue defined on the WebSphere MQ queue manager to which we are connecting.

Configuration

General Properties	Additional Properties
Scope Node=ITSOCellNode Provider WebSphere MQ messaging provider * Name BankMQJMSQueue * JNDI name jms/BankMQJMSQueue Description WebSphere MQ Queue for WebSphere Bank Category Persistence APPLICATION DEFINED Priority APPLICATION DEFINED Specified priority 0 Expiry APPLICATION DEFINED Specified expiry 0 milliseconds * Base queue name BankJMSQueue Base queue manager name	■ Custom properties ■ MQ Config

Figure 8-38 WebSphere MQ queue destination properties

4. Enter the required configuration properties for the WebSphere MQ queue destination.
5. Click **OK**.

6. Save the changes and synchronize them with the nodes.
7. For the changes to become effective, restart any application servers within the scope of the resources.

WebSphere MQ topic destination configuration

To configure a topic destination for the WebSphere MQ JMS provider, complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS** → **Topics**.
2. Set the scope. A list of any existing topic destinations defined at this scope will be displayed. This is shown in Figure 8-39.

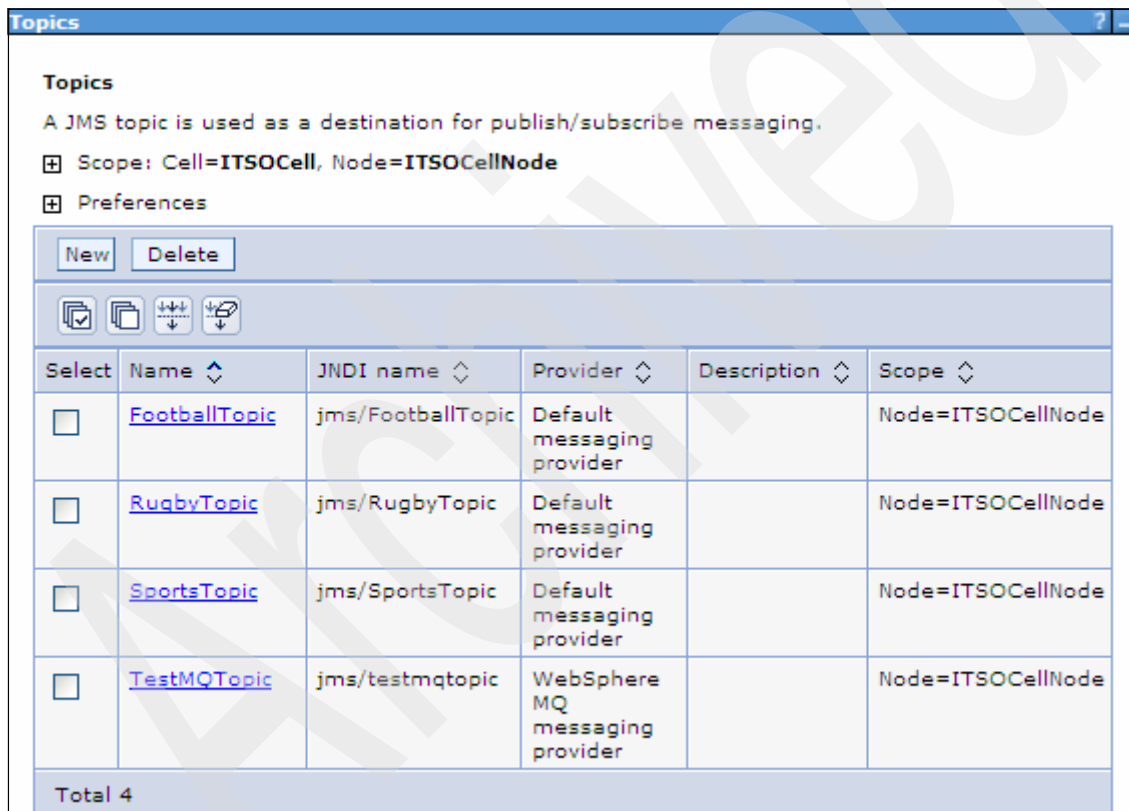


Figure 8-39 WebSphere MQ topic destination administered objects

In this example, we already have one WebSphere MQ topic destination object defined, called TestMQTopic. Note the provider type.

3. To create a new topic destination object, click **New**, specify the type of provider in the next window, and click **Next**. Alternatively, to change the properties of an existing topic destination, click one of the topic destinations displayed. Figure 8-40 on page 510 shows the top portion of the configuration page for TestMQTopic object.

Other than the standard JMS administered object properties, Name and JNDI name, the only property that we must specify a value for is Base topic name. In Figure 8-40, the value specified for the Base topic name property is TestTopic. This must match the name of the topic defined on the broker.

[JMS providers](#) > [WebSphere MQ messaging provider](#) > [Topics](#) > **TestMQTopic**

Topic destinations provided for publish and subscribe messaging by the WebSphere MQ JMS provider. Use WebSphere MQ topic destination administrative objects to manage topic destinations for the WebSphere MQ JMS provider.

Configuration

General Properties	Additional Properties
Scope Node=ITSOCellNode	☒ Custom properties
Provider WebSphere MQ messaging provider	
* Name TestMQTopic	
* JNDI name jms/TestMQTopic	
Description 	
Category 	
Persistence APPLICATION DEFINED	
Priority APPLICATION DEFINED	
Specified Priority 0	
Expiry APPLICATION DEFINED	
Specified Expiry 0 milliseconds	

Figure 8-40 WebSphere MQ topic destination properties

4. Enter the required configuration properties for the WebSphere MQ topic destination.
5. Click **OK**.

6. Save the changes and synchronize them with the nodes.
7. For the changes to become effective, restart any application servers within the scope of the resources.

8.6.4 Configuring listener ports

As discussed in 8.4.7, “Associating a message-driven bean with a destination” on page 445, a listener port is used to associate a message-driven bean with a connection factory and a destination for the WebSphere MQ JMS provider. A listener must be defined on the application server on which the message-driven application will be installed. To configure a listener port, complete the following steps:

1. In the navigation tree, expand **Servers**.
2. Click **Application servers**.
3. A list of the application servers defined within the cell will be displayed. This is shown in Figure 8-41.

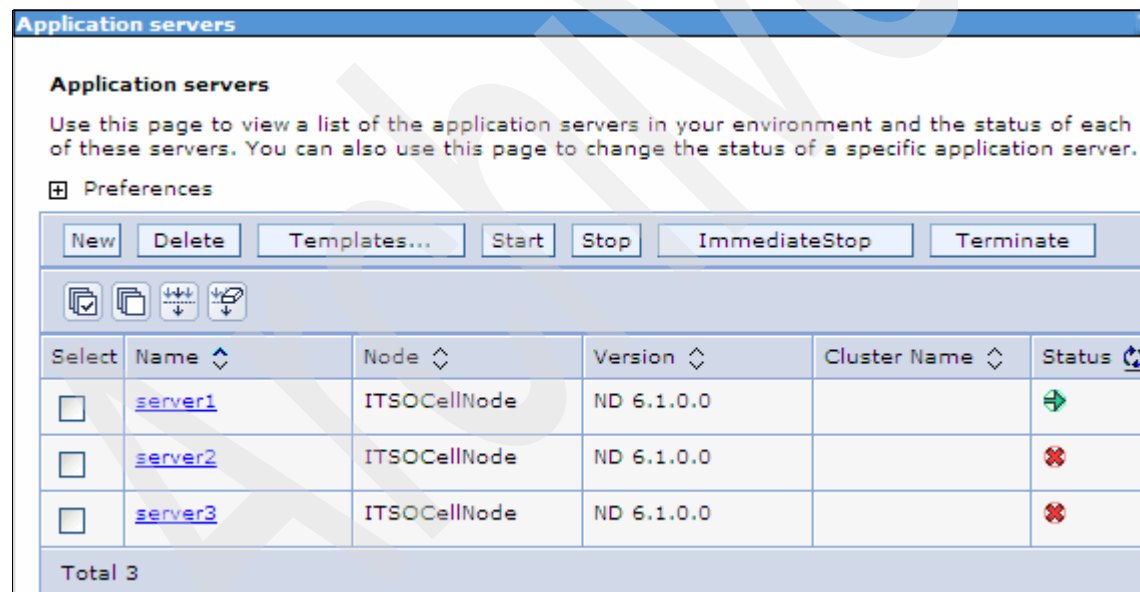


Figure 8-41 Application servers defined within the cell

4. Click the application server on which to create the listener port.
5. The configuration properties for the application server will be displayed. In the Communications section, expand **Messaging**.

6. Click **Message Listener Service**, as shown in Figure 8-42 on page 512.

The screenshot displays the 'Configuration' tab of the WebSphere Configuration Console. The interface is divided into several sections:

- General Properties:** Includes fields for 'Name' (server1) and 'Node Name' (ITSOCellNode). Checkboxes for 'Run in development mode' (unchecked) and 'Parallel start' (checked). A dropdown for 'Access to internal server classes' is set to 'Allow'.
- Server-specific Application Settings:** Includes a dropdown for 'Classloader policy' (Multiple) and a dropdown for 'Class loading mode' (Parent first).
- Buttons:** 'Apply', 'OK', 'Reset', and 'Cancel' are located at the bottom of the General Properties section.
- Container Settings:** A list of expandable sections: Session management, SIP Container Settings, Web Container Settings, Portlet Container Settings, EJB Container Settings, Container Services, and Business Process Services.
- Applications:** Includes a link for 'Installed applications'.
- Server messaging:** Includes links for 'Messaging engines', 'Messaging engine inbound transports', 'WebSphere MQ link inbound transports', and 'SIB service'.
- Server Infrastructure:** Includes expandable sections for 'Java and Process Management' and 'Administration'.
- Communications:** Includes expandable sections for 'Ports' and 'Messaging'. The 'Messaging' section is expanded, and the 'Message Listener Service' link is highlighted with a red oval.
- Performance:** A section header at the bottom.

Figure 8-42 Message listener service link

7. The configuration properties for the message listener service will be displayed.

- Click **Listener Ports**. A list of listener ports that are currently defined for this application server will be displayed. This is shown in Figure 8-43 on page 513.

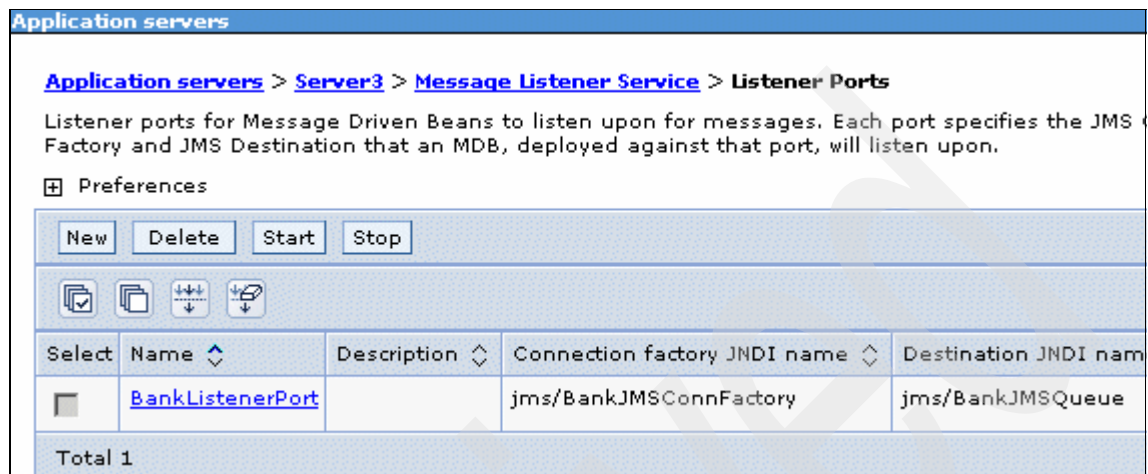


Figure 8-43 Listener ports

In this example, we already have one listener port defined, called BankListenerPort.

9. To create a new listener port, click **New**. Alternatively, to change the properties of an existing listener port, click one of the listener ports displayed. Figure 8-44 on page 514 shows the configuration page for the BankListenerPort object. Values must be specified for the Name, Initial State, Connection factory JNDI name, and Destination JNDI name properties.

Application servers

[Application servers](#) > [server1](#) > [Message Listener Service](#) > [Listener Ports](#) > **BankListenerPort**

Use this page to configure listener ports upon which message-driven beans listen for messages. Each port specifies the JMS connection factory and JMS destination the message-driven bean, deployed against that port, listens upon.

General Properties

* **Name**

* **Initial State**
 ▼

Description

* **Connection factory JNDI name**

* **Destination JNDI name**

Maximum sessions

Maximum retries

Maximum messages

Figure 8-44 Listener port properties

10. Enter the required configuration properties for the JMS activation specification.

11. Click **OK**.
12. Save the changes and synchronize them with the nodes.
13. For the changes to become effective, restart any application servers within the scope of the resources.

Following our example, using the WebSphere MQ connection factory defined in “WebSphere MQ connection factory configuration” on page 498 and the WebSphere MQ queue defined in “WebSphere MQ queue destination configuration” on page 504, we know that the BankMQJMSConnFactory object was bound into the JNDI name space with the name `jms/BankJMSConnFactory`. This JMS connection factory maps to a WebSphere MQ Queue Manager running on host `kcgg1d6` and listening on port 1414. We also know that the BankMQJMSQueue object was bound into the JNDI name space with the name `jms/BankJMSQueue`. This JMS queue maps on to the BankJSQueue on this WebSphere MQ Queue Manager.

Therefore, if a message-driven bean is associated with this listener port, it would be invoked when messages arrived at the BankJSQueue destination on the WebSphere MQ Queue Manager listening on port 1414 of host `kcgg1d6`.

8.6.5 Configuring a generic JMS provider

If you use a generic JMS provider, the WebSphere administrative console can still be used to configure JMS administered objects within the JNDI name space of the application server. The sections that follow describe how the WebSphere administrative console can be used to specify a JMS provider, and also to configure JMS connection factories and JMS destinations for that JMS provider.

JMS connection factory configuration

To remain compatible with JMS specification 1.0, there are two specific types of connection factories (prefixed with “Queue” and “Topic”) and a more general type of connection factory with no prefix (JMS 1.1). The particular properties of specific types of connection factories will be a subset of the more general connection factory, but all are administered in the same way.

To configure a JMS connection factory for a generic JMS provider, complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS** → **Connection factories**.
2. Set the scope. A list of any existing connection factories defined at this scope will be displayed.
3. To create a new connection factory object, click **New**, specify the previously defined generic provider in the next window, and click **Next**. Alternatively, to

change the properties of an existing connection factory, click one of the connection factories displayed. Figure 8-45 shows the configuration page for a connection factory object.

General Properties

Scope
Node=ITSOCellNode

Provider
SwiftMQ JMS Provider

* Name

* Type
UNIFIED

* JNDI name

Description

Category

* External JNDI name

Component-managed authentication alias
(none)

Container-managed authentication alias
(none)

Mapping-configuration alias
DefaultPrincipalMapping

Apply OK Reset Cancel

The additional properties will not be available until the general properties of this item are applied and saved.

Additional Properties

- Custom properties
- Connection pools
- Session pools

Related Items

- JAAS - J2C authentication data

Figure 8-45 Generic JMS provider connection factory configuration window

4. Enter the required configuration properties for the JMS connection factory. The common properties are described in 8.6.1, “Common administration properties” on page 462. The properties specific to the generic JMS connection factory object are shown in Table 8-37 on page 517.

Table 8-37 Generic JMS provider connection factory properties

Property	Description
Type	This is a read-only property that is set according to the type of connection factory being configured. For a JMS 1.1 general connection factory, the property will be UNIFIED. For the Queue Connection Factory and Topic Connection Factory, the property will be QUEUE or TOPIC, respectively.
External JNDI name	Specify the JNDI name used to bind the JMS connection factory into the name space of the messaging provider.
Component managed authentication alias	The component-managed authentication alias list can be used to specify a Java 2 Connector authentication data entry. If the resource reference used within the JMS client application specifies a res-auth of Application, the user ID and password defined by the Java 2 Connector authentication data entry will be used to authenticate the creation of a connection. The component-managed authentication alias defaults to none. If no component-managed authentication alias is specified and the messaging provider requires the user ID and password to get a connection, then an exception will be thrown when attempting to connect. If using a Component managed alias, the Container managed alias (below) should not be used.
Container managed authentication alias	The container-managed authentication alias list can be used to specify a Java 2 Connector authentication data entry. Please refer to the Component managed authentication alias description (above) for more detail. If using a Container managed alias, the Component managed alias (above) should not be used.
Mapping-configuration alias	This property provides a list of modules defined at Security → Java Authentication and Authorization Service → Application Logins. The DefaultPrincipalMapping JAAS configuration maps the authentication alias to the user ID and password required by the JMS Provider resource. Other mappings can be defined and used.

5. Click **OK**.
6. Save the changes and synchronize them with the nodes.
7. For the new connection factory to be bound into the JNDI name space at the correct scope, restart the relevant application servers.

JMS destination configuration

There are two types of Generic JMS provider destinations: Queue and Topic. The properties for both are precisely the same, so only creation of the Queue will be described here.

To configure a JMS destination for a generic JMS provider, complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS** → **Queues**.
2. Set the scope. A list of any existing queues defined at this scope will be displayed.

3. To create a new destination, click **New**, specify the previously defined generic provider in the next window, and click **Next**. Alternatively, to change the properties of an existing destination, click one of the destinations displayed. Figure 8-46 on page 519 shows the configuration page for a destination object.

JMS providers > SwiftMQ JMS Provider > Queues > New

A Generic JMS destination defines the configuration properties of either a queue (for point messaging) or a topic (for Publish and subscribe messaging) provided by the generic JMS provider.

Configuration

General Properties

* **Scope**
Node=ITSOCellNode

Provider
SwiftMQ JMS Provider

* **Name**

* **Type**
QUEUE

* **JNDI name**

Description

Category

* **External JNDI Name**

Additional Properties

■ Custom p

Buttons: Apply OK Reset Cancel

Figure 8-46 Generic JMS provider Queue destination configuration window

4. Enter the required configuration properties for the JMS destination. The common properties are described in 8.6.1, “Common administration properties” on page 462. The properties specific to the generic JMS destination object are shown in Table 8-38.

Table 8-38 Generic JMS provider destination properties

Property	Description
Type	This is a read-only property set to QUEUE or TOPIC depending on the type of destination being configured.
External JNDI name	Define the JNDI name used to bind the JMS connection factory into the name space of the messaging provider.

5. Click **OK**.
6. Save the changes and synchronize them with the nodes.
7. For the new destination to be bound into the JNDI name space at the correct scope, restart the relevant application servers.

8.7 Connecting to a service integration bus

A JMS client obtains connections to a service integration bus using a suitably configured JMS connection factory, defined for the default messaging JMS provider. However, the selection of which messaging engine within a particular service integration bus a JMS client will connect to depends on the connection properties defined within the JMS connection factory. The options available can range from simply connecting to any suitable messaging engine within the named service integration bus, to using a highly specific connection selection algorithm. The sections that follow describe the mechanisms used to determine the most suitable messaging engine when a JMS client is connecting to a service integration bus.

Note: None of the messaging engine selection processes discussed in this section affect the JMS client in any way. As far as the JMS client is concerned, the ConnectionFactory simply returns a connection to the underlying messaging provider, in this case, a service integration bus. The process of configuring a ConnectionFactory in order to tailor the messaging engine that is selected, is a purely administrative task.

8.7.1 JMS client run time environment

Regardless of the environment on which a JMS client is executing, it will always perform the same steps in order to connect to a JMS provider. These steps are:

1. Obtain a reference to a JMS connection factory from the JNDI name space.
2. Invoke the `createConnection` method on the JMS connection factory.

The important point here is that the JMS connection factory object will always execute within the same process as the JMS client. However, the JMS client, and therefore the JMS connection factory, might be executing inside of a WebSphere process, or they might be executing within a stand-alone JVM. In the case of the connection factory for the default messaging JMS provider, the behavior of the connection factory depends on the environment on which it is executing.

- Clients running inside of WebSphere Application Server

When the connection factory is executing within the WebSphere Application Server V6 environment, it is able to communicate with components of the WebSphere run time in order to determine which messaging engines are defined within the specified service integration bus, and where these messaging engines are currently located. The relevant connection properties configured on the connection factory can then be used to select a suitable messaging engine to which to connect.

Note: The connection factory is only able to determine the location of messaging engines that are defined within the same WebSphere cell. If the target bus is defined within another cell, then a list of suitable provider endpoints must be configured on the connection factory.

- Clients running outside of WebSphere Application Server

When the connection factory is executing outside of the WebSphere Application Server V6 environment, or in a WebSphere Application Server V6 environment on a different cell to the target bus, it is not able to determine which messaging engines are defined within the specified service integration bus or where they are currently located. In order to obtain this information, the connection factory must connect to an application server within the same cell as the target bus. This application server is known as a *bootstrap server*.

A bootstrap server is simply an ordinary application server that is running the SIB service. The SIB service is the component within an application server that manages the service integration bus resources for that application server. It is the SIB service that enables an application server to act as bootstrap server for default messaging JMS provider connection factories. However, while the bootstrap server needs to be running the SIB service, it does not

necessarily need to be hosting any messaging engines. This is shown in Figure 8-47 on page 522.

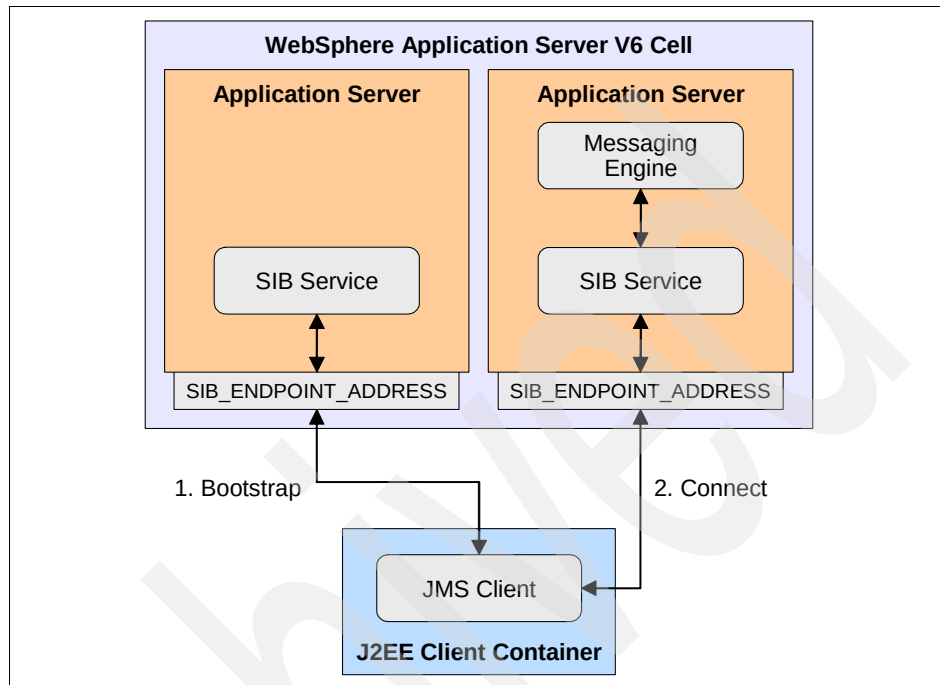


Figure 8-47 Using a bootstrap server with a messaging engine

Use the provider endpoints property to configure the bootstrap servers to which a connection factory can connect.

Provider endpoints

The provider endpoints property of the connection factory allows an administrator to specify a comma-separated list of suitable bootstrap servers for the connection factory. Each bootstrap server in the list is specified as a triplet of the form:

hostname : port : transport chain

The different elements are:

- ▶ **hostname** is the name of the host on which the bootstrap server is running. If a host name is not specified, the value will default to localhost.
- ▶ **port** is the port number that the SIB service for the bootstrap server is listening on. This can be determined from the relevant messaging engine inbound transport that will be used for the bootstrap request. If no port is

specified, the value will default to 7276 (the default port number for SIB_ENDPOINT_ADDRESS).

- `transport_chain` specifies the transport chain that will be used to send the bootstrap request to the bootstrap server. Valid values for transport chain are:

- `BootstrapBasicMessaging`

The bootstrap request will be sent to the bootstrap server using a standard TCP/IP connection to the `InboundBasicMessaging` transport chain.

- `BootstrapSecureMessaging`

The bootstrap request will be sent to the bootstrap server over a secure TCP/IP connection to the `InboundSecureMessaging` transport chain.

- `BootstrapTunneledMessaging`

The bootstrap request will be tunneled to the bootstrap server over an HTTP connection. Before you can use this transport chain, you must define a corresponding transport chain on the bootstrap server.

- `BootstrapTunneledSecureMessaging`

The bootstrap request will be tunneled to the bootstrap server over a secure HTTP connection. Before you can use this transport chain, you must define a corresponding transport chain on the bootstrap server.

If no transport chain is specified the value will default to `BootstrapBasicMessaging`.

If no value is specified, for the provider endpoint property, the connection factory will use the following default provider endpoint address:

`localhost:7276:BootstrapBasicMessaging`

Dedicated bootstrap servers

Because the location of a bootstrap server is defined explicitly within the provider endpoints property of a connection factory, consideration must be given to the availability of the bootstrap server. By specifying a list of bootstrap servers in the provider endpoints property, a connection factory is able to transparently bootstrap to another server in the list in the event that one of the bootstrap servers fails. The connection factory attempts to connect to a bootstrap server in the order in which they are specified in the provider endpoints list. However, you want to avoid specifying a long list of bootstrap servers. Consider configuring only a few highly available application servers as dedicated bootstrap servers.

8.7.2 Controlling messaging engine selection

The remaining connection properties that can be specified on a connection factory for the default messaging JMS provider are used to control how the connection factory selects the messaging engine to connect to on the specified service integration bus. The sections that follow discuss these properties in more detail.

Bus name

The only connection property that is required when configuring a connection factory for the default messaging JMS provider is the `bus name` property. The value of the `bus name` property specifies the name of the bus to which the connection factory will create JMS connections.

In the absence of any other connection properties, the connection factory returns a connection to any available messaging engine in the bus. However, despite the freedom to connect to any available messaging engine in the bus, the connection factory applies a few simple rules to find the most suitable messaging engine with which to connect. The rules are as follows:

1. The connection factory looks for a messaging engine within the specified service integration bus that is in the same server process as the JMS client. If a messaging engine within the specified bus is found in the same application server process, then a direct *in-process* connection is made from the JMS client to the messaging engine. This is shown in Figure 8-48.

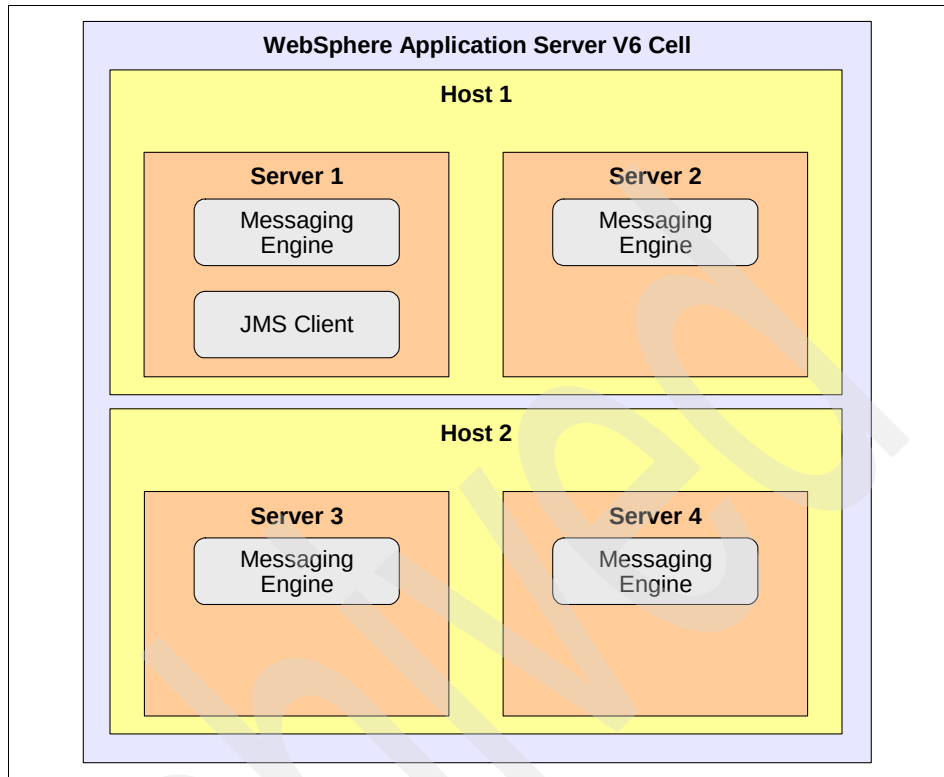


Figure 8-48 In-process connection for a JMS client and a messaging engine

Note: A direct in-process connection provides the best performance when connecting a JMS client to a messaging engine.

2. If it is not possible for the connection factory to create a connection to a messaging engine in the same application server process, the connection factory looks for a messaging engine that is running on the same host as the JMS client. If a messaging engine within the specified bus is found on the same host, then a remote connection is made from the JMS client to the messaging engine. This is shown in Figure 8-49.

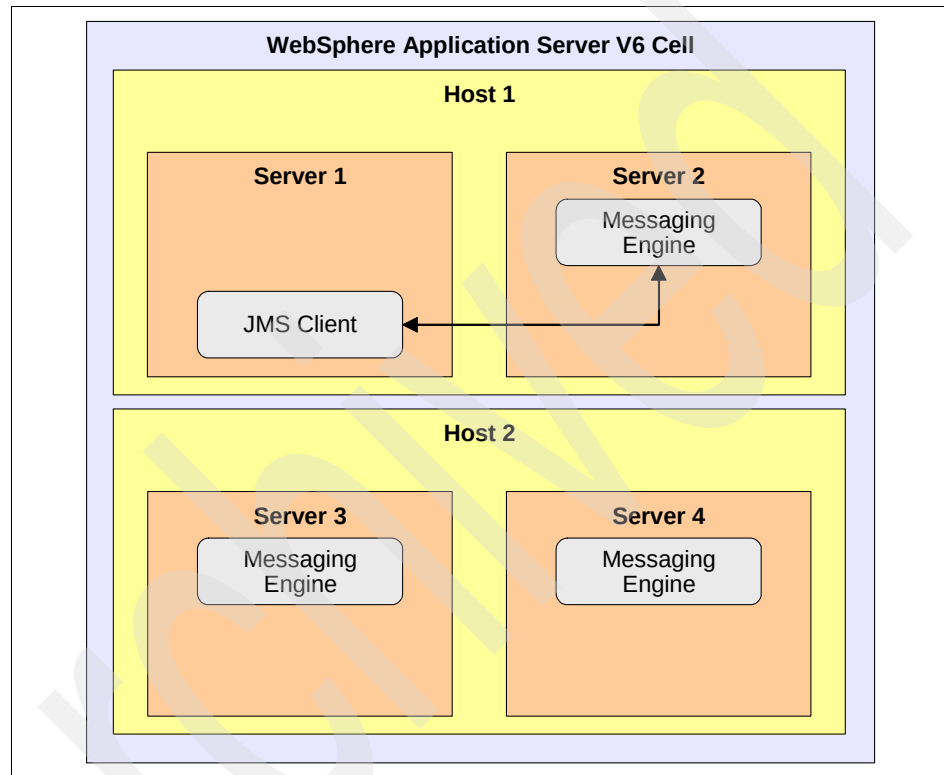


Figure 8-49 Remote connection on the same host

Note: If multiple messaging engines are available on the same host as the JMS client, new connections to the target bus will be load-balanced across them.

3. If it is not possible for the connection factory to create a connection to a messaging engine on the same host as the JMS client, the connection factory looks for any other messaging engine that is part of the specified service integration bus. This is shown in Figure 8-50.

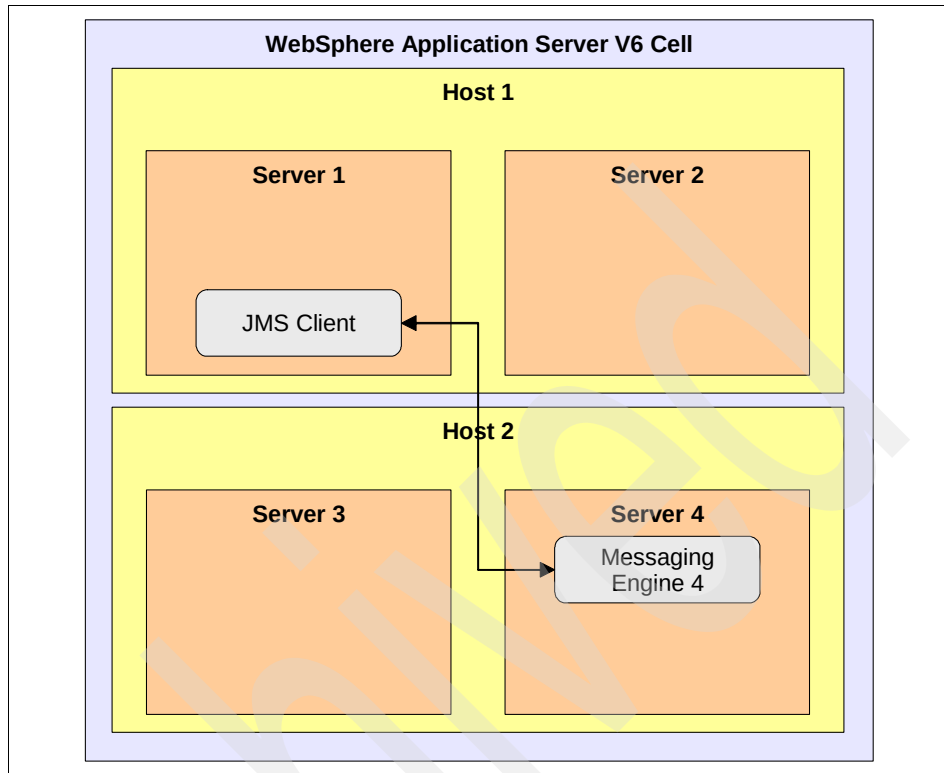


Figure 8-50 Remote connection on a different host

Note: If multiple messaging engines are available within the target bus, new connections to the target bus will be load balanced across them.

4. If it is not possible for the connection factory to create a connection to any of the messaging engines that make up the specified service integration bus, the connection factory throws a `javax.jms.JMSEException` to the JMS client. The `javax.jms.JMSEException` contains a linked exception to a service integration bus specific exception, similar to that shown in Example 8-21.

Example 8-21 Failure to connect to a messaging engine

```
com.ibm.websphere.sib.exception.SIResourceException: CWSIT0019E: No suitable  
messaging engine is available in bus SamplesBus.
```

Target inbound transport chain

The target inbound transport chain property for a connection factory specifies the transport chain that the JMS client should use when establishing a remote connection to a messaging engine. Suitable values for this property are:

- **InboundBasicMessaging**

The JMS client establishes a standard TCP/IP connection to the messaging engine. This is the default value for the target inbound transport chain property.

- **InboundSecureMessaging**

The JMS client establishes a secure TCP/IP connection to the messaging engine.

The process of selecting a suitable messaging engine takes into account the inbound transport chains that are currently available to those messaging engines under consideration. There is no point in selecting a messaging engine that cannot be contacted using the target transport chain specified, so a final selection is made only from those messaging engines that have the specified target transport chain available to them.

Connection proximity

The messaging engine selection process performed by the connection factory can be subtly altered by specifying different connection proximities. The connection proximity property is used to restrict the set of available messaging engines considered for selection by the connection factory. The set of available messaging engines is restricted based on their proximity to the JMS client or the bootstrap server acting on behalf of the JMS client. The valid values for the connection proximity property are as follows:

- **Bus**

The set of available messaging engines will include all messaging engines defined within the target service integration bus. This is the default value for the connection proximity property and, in effect, does not restrict the set of

available messaging engines in any way. When a connection proximity of Bus is specified, the messaging engine selection process described in “Bus name” on page 524 is used.

- ▶ Cluster

The set of available messaging engines for the target service integration bus only includes those messaging engines defined within the same cluster as the JMS client or bootstrap server.

- ▶ Host

The set of available messaging engines for the target service integration bus only includes those messaging engines running on the same host as the JMS client or bootstrap server.

- ▶ Server

The set of available messaging engines for the target service integration bus only includes those messaging engines running within the same application server process as the JMS client or bootstrap server.

To see how the value of the connection proximity property affects the messaging engine selection process, consider the configuration shown in Figure 8-51. All of the messaging engines shown in Figure 8-51 exist within the same service integration bus.

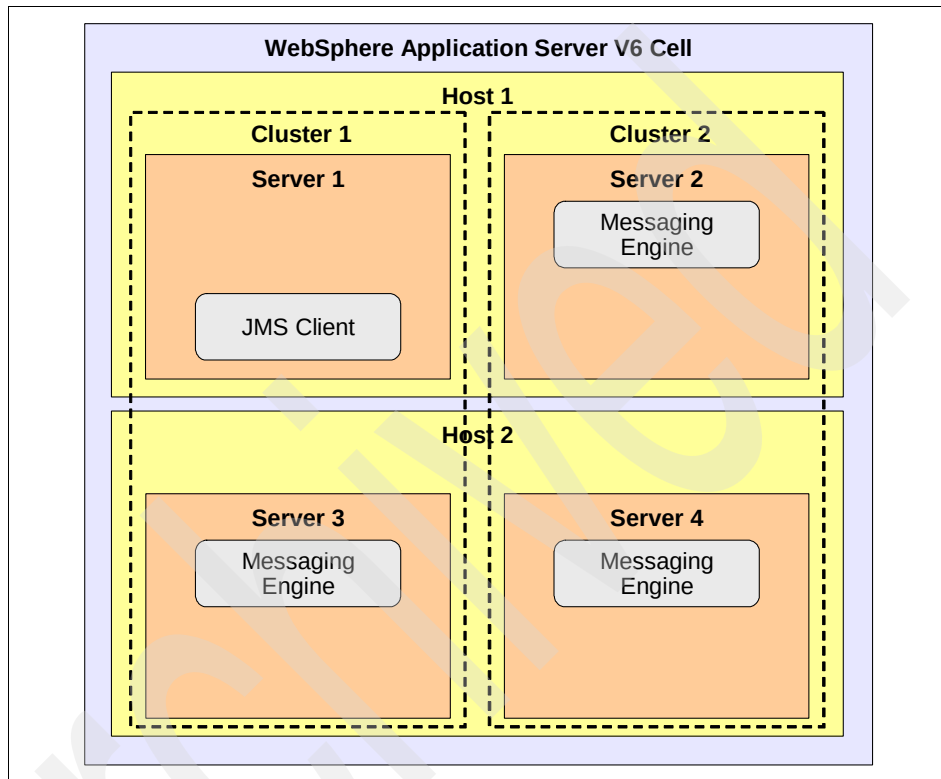


Figure 8-51 Sample topology for a service integration bus

The effect of the value of the connection proximity property on messaging engine selection is described in Table 8-39 on page 531.

Table 8-39 Effect of connection proximity on messaging engine selection

Connection proximity value	Messaging engine selected
Bus	The JMS client connects to the messaging engine on Server 2, following the rules described in “Bus name” on page 524.
Cluster	The JMS client connect to the messaging engine on Server 3, because this is the only messaging engine in the same cluster as the client.
Host	The JMS client connects to the messaging engine on Server 2, because this is the only messaging engine on the same host as the client.
Server	The JMS client fails to connect to the service integration bus, because there is no messaging engine in the same server as the client.

Target groups

Target groups provide a further means of controlling the selection of a suitable messaging engine by restricting the messaging engines available for consideration during the connection proximity check. Before the connection proximity search is performed, the set of messaging engines that are members of the specified target group is determined. The connection proximity check is then restricted to these messaging engines.

The use of target groups is controlled through the target, target type, and target significance properties of the connection factory, the descriptions for which are as follows:

► Target

The target property identifies a group of messaging engines that should be used when determining the set of available messaging engines. If no target group is specified, then no sub-setting of the available messaging engines takes place and every messaging engine within the bus is considered during the connection proximity check. By default, no target group is specified.

► Target type

The target type property specifies the type of the group identified by the target property. Valid values for the target type property are:

– Bus member name

Bus member name indicates that the target property specifies the name of a bus member. Because bus members can only be application servers or application server clusters, the value of the target property must be an application server name of the form <node name>.<server name> or the name of the cluster.

– Custom messaging engine group name

This value indicates that the target property specifies the name of a user defined custom group of messaging engines. A messaging engine is registered with a custom group by specifying the name of the group in the target groups property for the messaging engine. The registration of the messaging engine takes place when the messaging engine is started.

– Messaging engine name

Choosing this value indicates that the target property specifies the name of a specific messaging engine. This is the most restrictive target type that can be specified.

► Target significance

The target significance property allows the connection factory to relax the rules that are applied regarding the target group. The valid values for this property are as follows:

– Preferred

Use Preferred to indicate that a messaging engine be selected from the target group. A messaging engine in the target group is selected if one is available. If a messaging engine in the target group is not available, an available messaging engine within the specified service integration bus, but outside of the target group, is selected.

– Required

Use Required to indicate that a messaging engine be selected from the target group. A messaging engine in the target group is selected if one is available. If a messaging engine in the target group is not available, the connection process fails.

To see how the values of the target group properties affect the messaging engine selection process, consider the configuration shown in Figure 8-52 on page 533. All of the messaging engines shown in Figure 8-52 exist the same service integration bus.

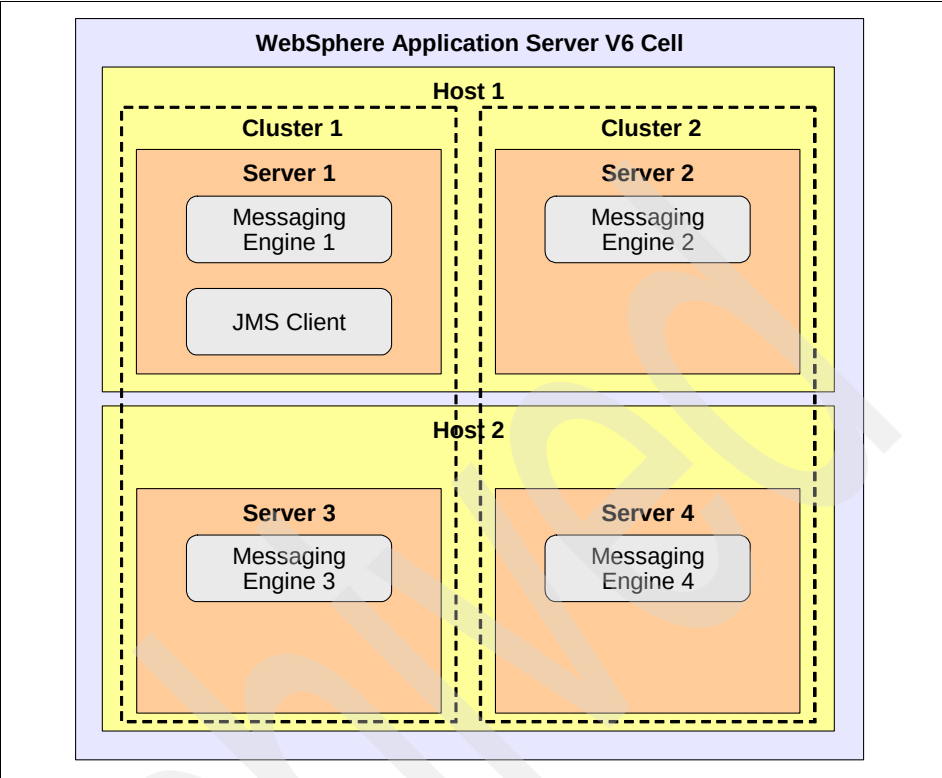


Figure 8-52 Sample topology for a service integration bus

The effect of the value of the connection proximity property on messaging engine selection is described in Table 8-40.

Table 8-40 Effect of target group properties on messaging engine selection

Connection property		Messaging engine selected
Name	Value	
Target	Cluster 2	The set of available messaging engines in the target group, Cluster 2, is: {Messaging Engine 2, Messaging Engine 4}. Because a connection proximity of Bus has been specified, the JMS client would connect to Messaging Engine 2. This is the only messaging engine in the set that is on the same host as the client.
Target type	Bus member name	
Target significance	Required	
Connection proximity	Bus	

Connection property		Messaging engine selected
Name	Value	
Target	Cluster 2	The set of available messaging engines in the target group, Cluster 2, is: {Messaging Engine 2, Messaging Engine 4}. Because a connection proximity of Server and a target significance of Required have been specified, the JMS client would fail to connect to the service integration bus, because there are no messaging engines in the target group that are on the same server as the client.
Target type	Bus member name	
Target significance	Required	
Connection proximity	Server	
Target	Cluster 2	By relaxing the target significance to Preferred, the JMS client is now able to connect to an alternative messaging engine that does not necessarily meet the connection proximity constraint. In this case, the JMS client would connect to Messaging Engine 1.
Target type	Bus member name	
Target significance	Preferred	
Connection proximity	Server	

8.7.3 Load balancing bootstrapped clients

JMS clients that connect to a service integration bus using a bootstrap server, which is itself running a suitable messaging engine, always connect to the messaging engine running in the bootstrap server. This is because this messaging engine is the closest suitable messaging engine to the bootstrap server.

Note: The term *suitable messaging engine* describes a messaging engine that matches all of the target group and connection proximity rules described in 8.7.2, “Controlling messaging engine selection” on page 524.

If there are many JMS clients using the same connection factory, they all bootstrap using the same list of bootstrap servers. Because the connection factory attempts to connect to a bootstrap server in the order in which they are specified in the provider endpoints list, it is likely that all of the JMS clients will be connected to the same messaging engine in the first available bootstrap server. The JMS clients will not be load-balanced across the set of suitable messaging engines. This is shown in Figure 8-53 on page 535.

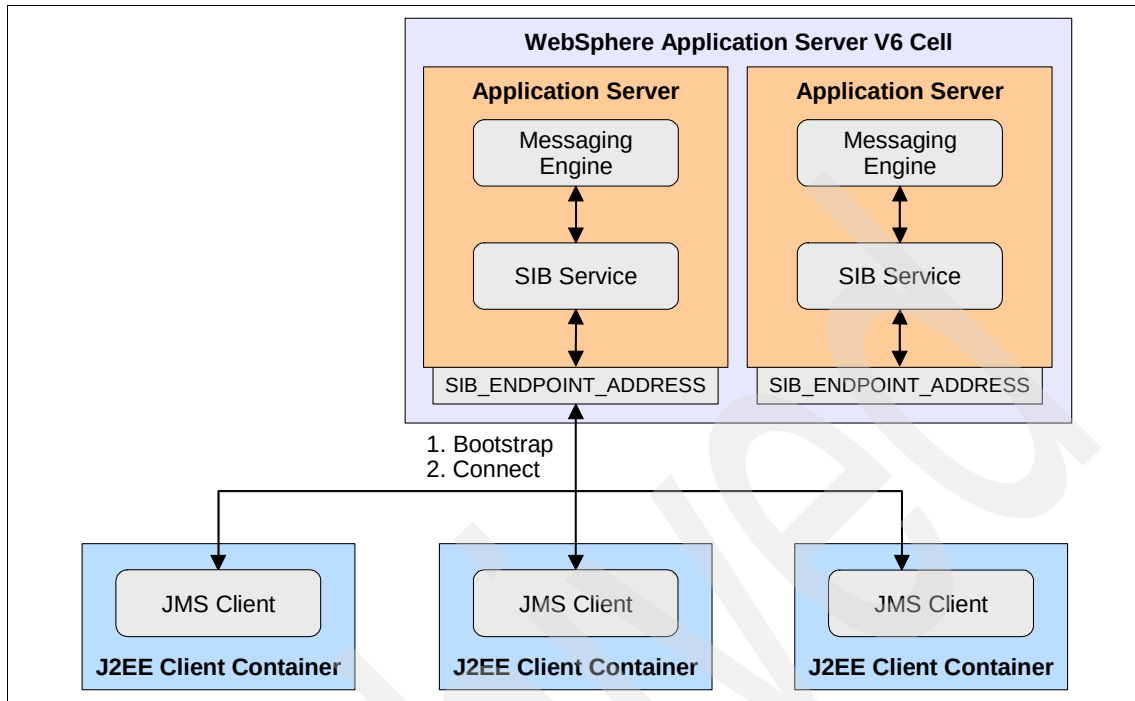
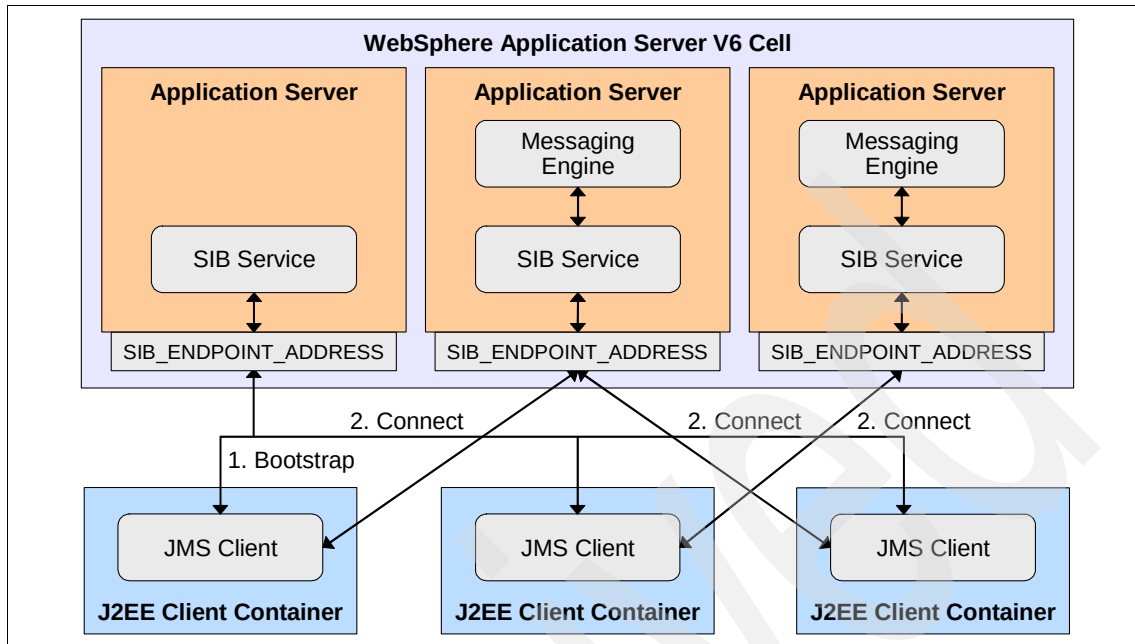


Figure 8-53 Bootstrapped JMS clients connecting to a single messaging engine

A solution to this problem is to make use of a dedicated bootstrap server that is not running a messaging engine for the target bus. This ensures that the connections established for JMS client are load-balanced across the available messaging engines for the target bus. This is shown in Figure 8-54 on page 536.

We expect that a future release will support the automatic load-balancing of bootstrapped JMS clients across the set of suitable messaging engines, thus reducing the tendency for bootstrapped JMS clients to congregate at a single bootstrap server.



8.8 References and resources

These documents and Web sites are contain relevant information:

- ▶ **WebSphere Information Center**
<http://www.ibm.com/software/webservers/appserv/infocenter.html>
- ▶ **Java 2 Platform Enterprise Edition Specification, v1.4**
http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf
- ▶ **J2EE Connector Architecture**
<http://java.sun.com/j2ee/connector/>
- ▶ **WebSphere MQ Using Java**
<http://www-306.ibm.com/software/integration/mqfamily/library/manualsa/manuals/crosslatest.html>
- ▶ **Java Message Service (JMS)**
<http://java.sun.com/products/jms>

- ▶ Yusuf, *Enterprise Messaging Using JMS and WebSphere*, Pearson Education, 2004, ISBN 0131468634
- ▶ Monson-Haefel, et al, *Java Message Service*, O'Reilly Media, Incorporated, 2000, ISBN 0596000685
- ▶ Giotto, et al, *Professional JMS*, Wrox Press Inc., 2001, ISBN 1861004931
- ▶ Monson-Haefel, et al, *Enterprise JavaBeans, Fourth Edition*, O'Reilly Media, Incorporated, 2004, ISBN 059600530X
- ▶ Marinescu, et al, *EJB Design Patterns*, Wiley, John & Sons, Incorporated, 2002, ISBN 0471208310

Default messaging provider

WebSphere Application Server V6 introduced a new component called the service integration bus. In this chapter, we describe the concepts behind the service integration bus, focusing on its role as the default messaging provider within WebSphere Application Server. We cover:

- ▶ Concepts and architecture
- ▶ Run time components
- ▶ High availability and workload management
- ▶ Service integration bus topologies
- ▶ Service integration bus and message-driven beans
- ▶ Service integration bus security
- ▶ Problem determination
- ▶ Configuration and management

9.1 Concepts and architecture

The service integration bus provides a managed communications framework that supports a variety of message distribution models, reliability options, and network topologies. It provides support for traditional messaging applications, as well as enabling the implementation of service-oriented architectures within the WebSphere Application Server environment.

The service integration bus is the underlying messaging provider for the default messaging JMS provider, replacing the embedded messaging provider that was supported in WebSphere Application Server V5.

The service integration bus introduces a number of new concepts. The sections that follow discuss each of these concepts in more detail.

9.1.1 Buses

A *service integration bus*, or *bus*, is simply an architectural concept. It gives an administrator the ability to group a collection of resources together that provide the messaging capabilities of the bus. At run time, the bus presents these cooperating messaging resources to applications as a single entity, hiding from those applications the details of how the bus is configured and where on the bus the different resources are located.

A bus is defined at the cell level. It is anticipated that, in a standard configuration, no more than one bus will be required within a cell. However, a cell can contain any number of buses.

Resources are created within, or added to, the scope of a specific bus. Simply defining a bus within a cell has no run time impact on any of the components running within a cell. It is not until members are added to a bus that any of the run time components within an application server are affected.

Figure 9-1 on page 541 shows a bus defined within a cell.

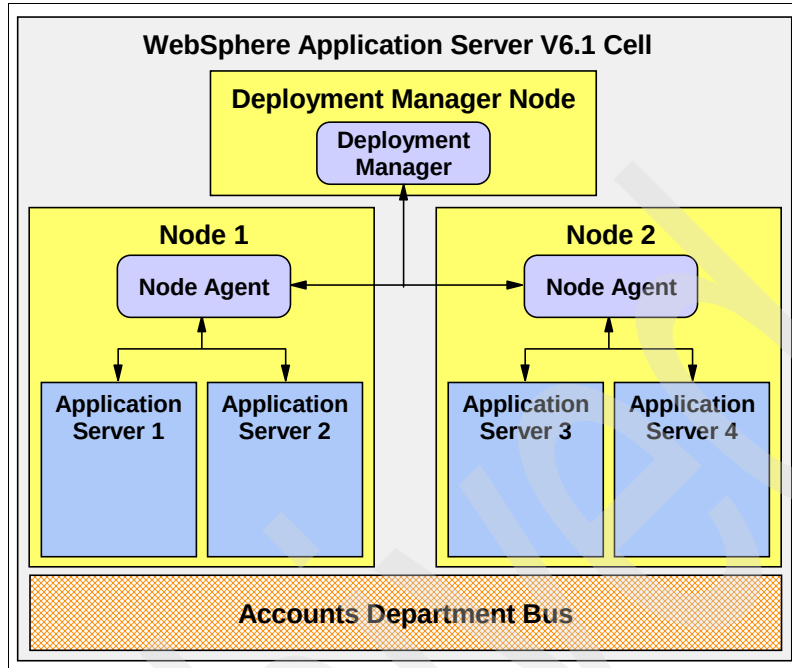


Figure 9-1 Service integration buses within a cell

9.1.2 Bus members

A *bus member* is simply an application server, or cluster of application servers, that has been added as a member of a bus. Adding an application server, or cluster of application servers, as a member of a bus automatically defines a number of resources on the bus member in question. In terms of the functionality provided by a bus, the most important of the resources that are automatically defined is a messaging engine.

9.1.3 Messaging engines

A *messaging engine* is the component within an application server that provides the core messaging functionality of a bus. At run time, it is the messaging engines within a bus that communicate and cooperate with each other to provide the messaging capabilities of the bus. A messaging engine is responsible for managing the resources of the bus and it also provides a connection point to which local and remote client applications can connect.

A messaging engine is associated with a bus member. When an application server is added as a member of a bus, a messaging engine is automatically

created and associated with this application server. Figure 9-2 on page 542 shows a cell that contains two buses, each of which has two application servers defined as bus members.

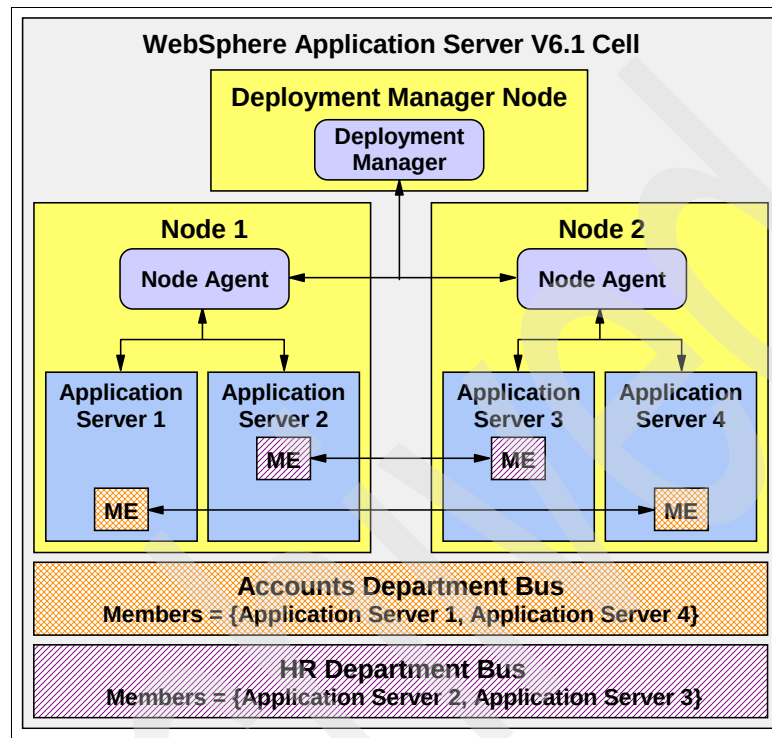


Figure 9-2 Messaging engines within bus members

A messaging engine is a relatively lightweight run time object. This allows a single application server to host several messaging engines. If an application server is added as a member of multiple buses, that application server is associated with multiple messaging engines, one messaging engine for each bus of which it is a member. This is shown in Figure 9-3 on page 543.

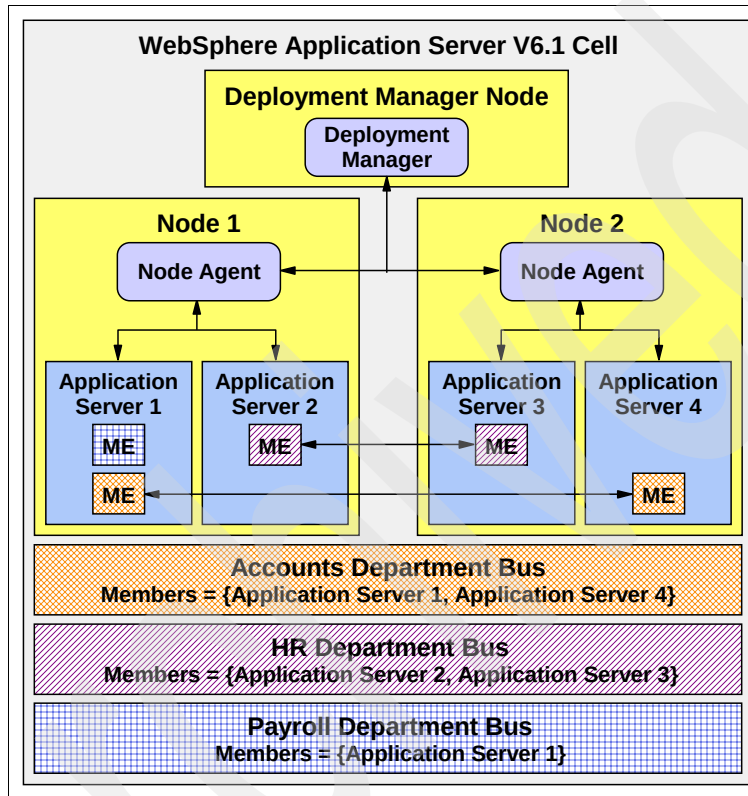


Figure 9-3 Multiple messaging engines within a single application server

When a cluster of application servers is added as a member of bus, a single messaging engine is automatically created and associated with the application server cluster, regardless of the number of application servers defined as members of the cluster. At run time, this messaging engine is activated within a single application server within the cluster. The application server that is chosen to host the messaging engine will be the first cluster member to start. This is shown in Figure 9-4.

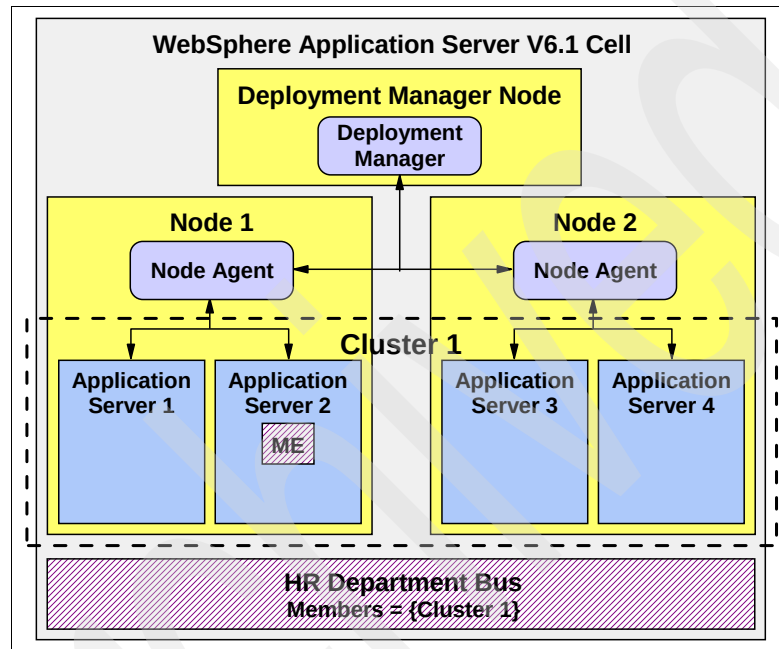


Figure 9-4 An application server cluster as a bus member

However, this messaging engine is able to run within any of the application servers defined as members of the cluster. If the messaging engine, or the application server within which it is running, should fail, the messaging engine is activated on another available server in the cluster. Therefore, adding an application server cluster as a member of a bus enables failover for messaging engines that are associated with that cluster. This is shown in Figure 9-5 on page 545.

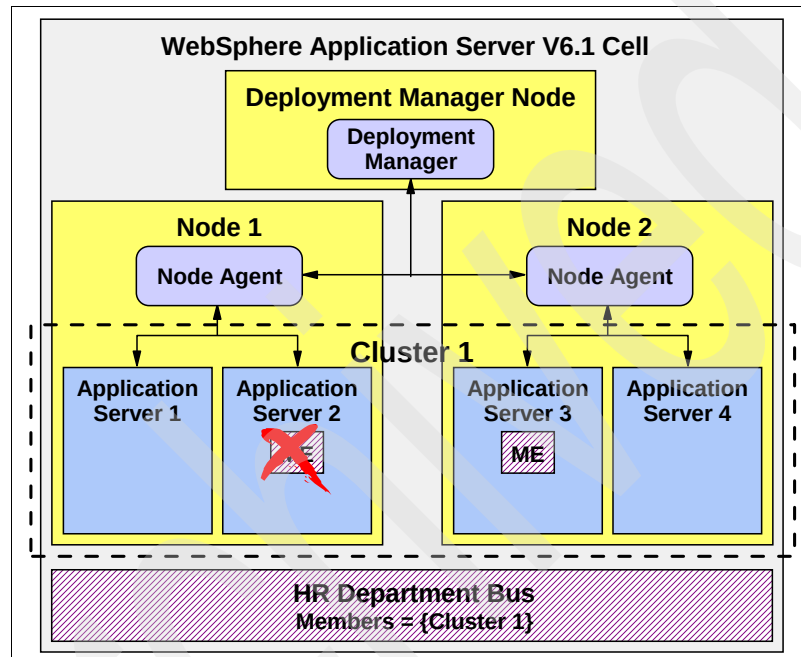


Figure 9-5 Messaging engine failover within an application server cluster

Once an application server cluster has been added as a member of a bus, it is also possible to create additional messaging engines and associate them with the cluster. These additional messaging engines can then be configured to run within a specific cluster member, if required. Such a configuration enables a bus to be scaled to meet the needs of applications that generate high message volumes. It also improves the availability of the bus in question. This is shown in Figure 9-6 on page 546.

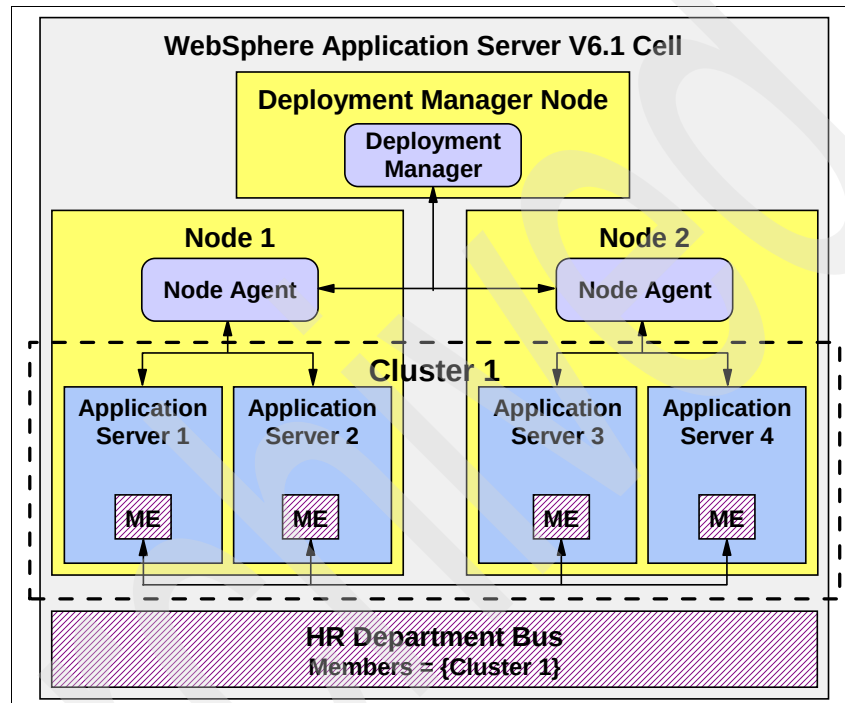


Figure 9-6 Messaging engine scalability within an application server cluster

For more information about failover and scalability within the bus, refer to 9.3, “High availability and workload management” on page 594.

Messaging engine naming

As discussed previously, when a member is added to a bus, a messaging engine is automatically created and associated with the new bus member. The name of the new messaging engine is generated based on the details of the new bus member, as follows:

► Application server bus members

The format of the messaging engine name generated when an application server is added as a member of a bus is as follows:

`<node>.<server>-<bus>`

The elements are defined as:

- `<node>` is the name of the node on which the new bus member is defined.
- `<server>` is the name of the new application server bus member.
- `<bus>` is the name of the bus to which the new bus member has been added.

We can use it in an example, such as:

ITSONode.Server 1-ITSOBUS

► Application server cluster bus members

The format of the messaging engine name generated when an application server cluster is added as a member of a bus is as follows:

`<cluster>.<X>-<bus>`

The elements of this format are:

- `<cluster>` is the name of the new application server cluster bus member.
- `<X>` is a number that is used to uniquely identify the messaging engine within the cluster. This value starts at 000 and is incremented each time a new messaging engine is added to the cluster.
- `<bus>` is the name of the bus to which the new bus member has been added.

We can use it in an example, such as:

ITSOCcluster.000-ITSOBUS

9.1.4 Message stores

New in V6.1: In V6.0, the message store was backed by a relational database. In V6.1, you have the option of using a flat file backed by the operating system.

Every messaging engine defined within a bus has a *message store* associated with it. A messaging engine uses this message store to persist durable data, such as persistent messages and transaction states. Durable data written to the message store survives the orderly shutdown, or failure, of a messaging engine, regardless of the reason for the failure.

It can also use the message store to reduce run time resource consumption. For example, the messaging engine can write non-persistent messages to the message store in order to reduce the size of the Java heap when handling high message volumes. This is known as *spilling*.

Message stores can be implemented as a set of database tables (known as a *data store*), or as flat files (known as a *file store*). Figure 9-7 on page 549 shows messaging engines associated with message stores. Two of the messaging engines shown in Figure 9-7 are associated with data stores that exist within the same database, each with its own set of tables and schema. The other messaging engine uses a file store on the local file system. There are certain considerations you must take into account when deciding the message store topology. These considerations are discussed in more detail in 9.2.3, “Message stores” on page 568, as part of the description of the run time components of the bus.

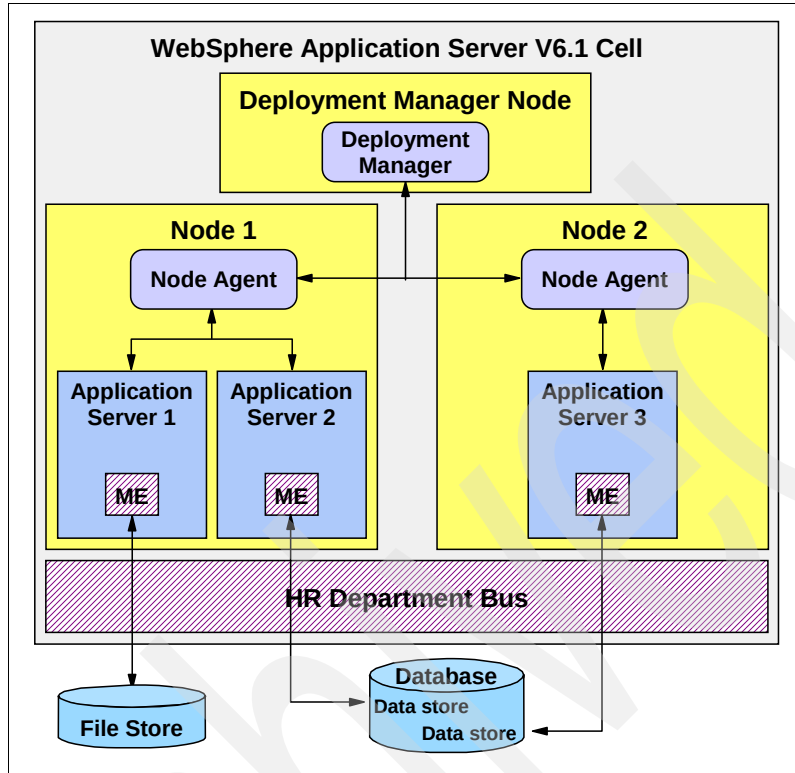


Figure 9-7 Messaging engine data stores

9.1.5 Destinations

A *destination* within a bus is a logical address to which applications can attach as message producers, message consumers, or both, in order to exchange messages. The main types of destination that can be configured on a bus are:

- Queue destinations

Queue destinations are destinations that can be configured for point-to-point messaging.

- Topic space destinations

Topic space destinations are destinations that can be configured for publish/subscribe messaging.

- Alias destinations

Alias destinations are destinations that can be configured to refer to another destination, potentially on a foreign bus. They can provide an extra level of indirection for messaging applications. An alias destination can also be used

to override some of the values specified on the target destination, such as default reliability and maximum reliability. Foreign buses are discussed in 9.1.7, “Foreign buses” on page 555.

- Foreign destinations

Foreign destinations are not destinations within a bus, but they can be used to override the default reliability and maximum reliability properties of a destination that exists on a foreign bus. Foreign buses are discussed in 9.1.7, “Foreign buses” on page 555.

Message points

When a destination is configured on a bus, it simply defines a logical address to which applications can attach. Queue and topic space destinations must be associated with a messaging engine in order for any persistent messages directed at those destinations to be persisted to an underlying message store. These destinations are associated with a messaging engine using a *message point*. A message point is a physical representation of a destination defined on a bus. A message point can be configured to override some of the properties inherited from the bus destination.

The two main types of message point that can be contained with a messaging engine are:

- Queue points

A *queue point* is the message point for a queue destination. When creating a queue destination on a bus, an administrator specifies the bus member that will hold the messages for the queue. This action automatically defines a queue point for each messaging engine associated with the specified bus member.

If the bus member is an application server, a single queue point will be created and associated with the messaging engine on that application server. All of the messages that are sent to the queue destination will be handled by this messaging engine. In this configuration, message ordering is maintained on the queue destination.

If the bus member is a cluster of application servers, a queue point is created and associated with each messaging engine defined within the bus member. The queue destination is partitioned across the available messaging engines within the cluster. In this configuration, message ordering is not maintained on the queue destination. For more information about partitioned destinations within the bus, please refer to 9.3, “High availability and workload management” on page 594.

► Publication points

A *publication point* is the message point for a topic space. When creating a topic space destination, an administrator does not need to specify a bus member to hold messages for the topic space. Creating a topic space destination automatically defines a publication point on each messaging engine within the bus.

Figure 9-8 on page 551 shows a queue destination and a topic space destination and their associated queue and publication points.

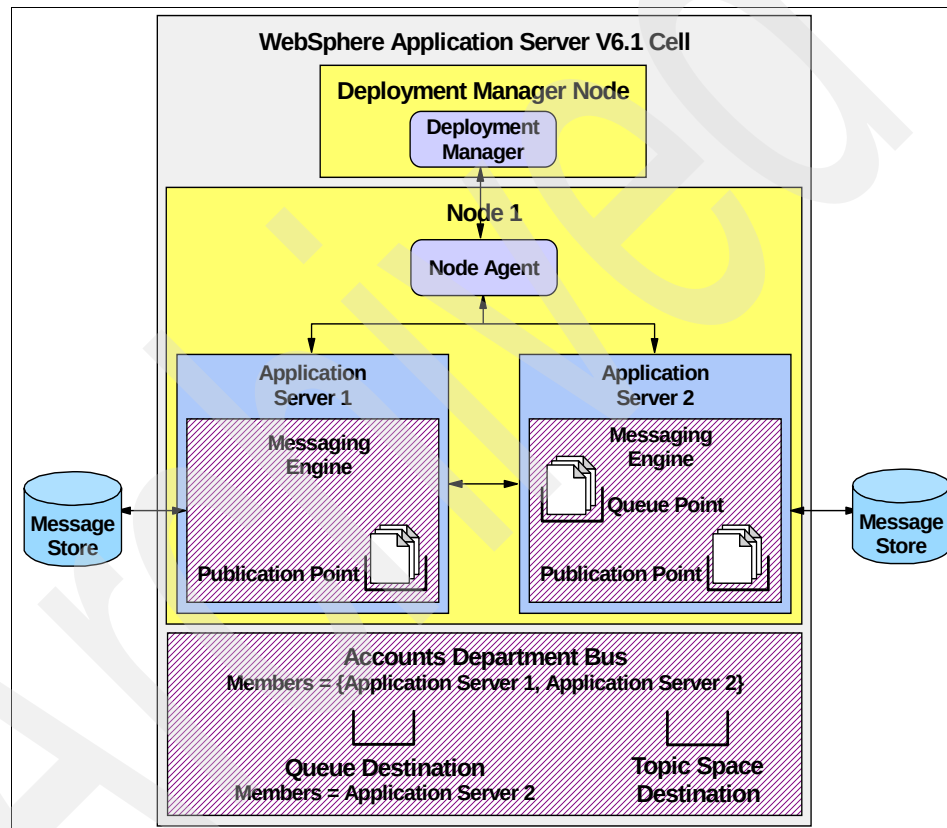


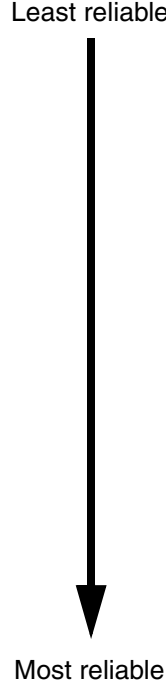
Figure 9-8 Queue and publication points in the bus

Reliability

It is on a destination that an administrator specifies the default quality of service levels that will be applied when a message producer or message consumer interacts with the destination. An administrator is able to configure a default reliability and a maximum reliability for each bus destination. There are five levels

of reliability that can be specified for these properties. These are described in Table 9-1.

Table 9-1 Service integration bus destination reliabilities



Reliability	Description
Best Effort nonpersistent	Messages that are sent to this destination are discarded when the messaging engine with which it associated is stopped, or if it fails. Messages can also be discarded if the connection used to send them becomes unavailable or as a result of constrained system resources. Messages delivered asynchronously to non-transactional MessageListeners or message-driven beans will not be redelivered if an exception is thrown.
Express nonpersistent	Messages that are sent to this destination are discarded when the messaging engine with which it is associated is stopped or if it fails. Messages can also be discarded if the connection used to send them becomes unavailable.
Reliable nonpersistent	Messages that are sent to this destination are discarded when the messaging engine with which it is associated is stopped or if it fails.
Reliable persistent	Messages that are sent to this destination can be discarded when the messaging engine with which it is associated fails, but are persisted if the messaging engine is stopped normally.
Assured persistent	Messages that are sent to this destination are never discarded.

Note: Reliability settings should be chosen according to your messaging needs. More reliable qualities of service might not perform as well as less reliable qualities of service.

Administrators can also allow message producers to override the default reliability that is specified on a destination. The mechanism that is used to achieve this depends on the type of the message producer. For instance, a JMS message producer can use the quality of service properties on the default messaging JMS provider connection factory to map the JMS PERSISTENT and NON_PERSISTENT delivery modes onto the required bus reliabilities. This is discussed in more detail in “Quality of service properties” on page 466.

Note: The reliability specified by a message producer can never exceed the maximum reliability specified on a bus destination. In the case of a JMS message producer, attempting to do this will cause a JMS exception to be thrown to the client application.

Strict message ordering

New in V6.1: Destinations on a bus can now be configured to be much more strict in the delivery of messages in the order they were produced. When the setting is enabled, certain automatic restrictions are placed on the use of the destination, such as disallowing concurrent consumption of messages by multiple applications, which may disrupt message ordering.

A destination can be configured so that the order of messages produced and consumed is preserved in a much more rigorous fashion than in normal circumstances. This setting is found on the destination configuration page.

Generally, messages going from a single producer to a single consumer will be seen to arrive in the same order in which they were produced. However, the order of messages may change due to certain events, such as a system failure of some kind. If a destination is configured to try and enforce message ordering, there are a number of automatic restrictions that come into play at run time. These are listed below:

1. Concurrent consumers are prevented from attaching to an ordered destination.

Only a single consumer can attach to an ordered destination at any given time. This is like an exclusive lock that prevents other consumers from attaching and potentially consuming messages out of order.

2. Partially consumed messages prevent subsequent messages from being consumed.

Destinations without strict message ordering will allow consumers to skip over messages that have been “partially” consumed. An example of this is a message that has a lock on it due to an uncommitted transaction. For a destination with strict message ordering, this would result in the destination being blocked until the partially consumed message is fully removed or replaced (committed or rolled back).

3. Concurrent message driven beans (MDBs) are restricted for an ordered destination.

To prevent race conditions and ensure ordered processing of MDBs from the destination, the maximum concurrent endpoints and maximum batch size

settings of any MDB deployed to an ordered destination are overridden and set to one.

4. Concurrent mediations are restricted for an ordered destination.

The concurrent mediation setting is set to false, ensuring an ordered mediation of messages.

However, there are other issues that should be understood, but cannot be automatically detected at run time. The main ones are listed below:

- ▶ If there is an exception destination configured, this may cause messages under error conditions to be directed away from the consumer, thus disrupting the message order. We recommend that for ordered destinations that no exception destination be defined.
- ▶ Topology changes to the bus, such as deleting and recreating an ordered destination, or introducing or removing mediation, could affect message ordering.
- ▶ Mediations or application code can be designed to disrupt message ordering. For example, mediations may divert messages to other destinations.
- ▶ Alias or foreign destinations do not have a message ordering option. In each case, only the underlying destination can be ordered.
- ▶ If a queue type destination is deployed to a cluster bus member with more than one messaging engine, this results in a destination with more than one queue point or mediation point. Message ordering cannot be maintained across such a destination.
- ▶ Only messages with a reliability of “assured persistent” should be used with an ordered destination. Any other reliability levels may result in lost or duplicated messages.
- ▶ Multiple producers can send messages to an ordered destination, but messages are presented in the order in which they were committed by the sending transaction. This may be different from the order in which they were actually written to the queue.
- ▶ Messages of different reliabilities can overtake one another. We recommend that messages to the ordered destination be of the same reliability level.
- ▶ Messages of different priorities can overtake one another. We recommend that messages to the ordered destination be of the same priority.

9.1.6 Mediations

A *mediation* processes in-flight messages between the production of a message by one application, and the consumption of a message by another application. Mediations enable the messaging behavior of a bus to be customized. Examples of the processing that can be performed by a mediation are:

- ▶ Transforming a message from one format into another
- ▶ Routing messages to one or more target destinations that were not specified by the sending application
- ▶ Augmenting messages by adding data from a data source
- ▶ Distributing messages to multiple target destinations
- ▶ Discarding messages

A mediation is defined within a specific bus. This mediation can then be associated with a destination on the bus. A corresponding *mediation point* is automatically created and associated with the destination as a result of this process. A mediation point is a specialized type of message point. A destination with which the mediation is associated is referred to as a *mediated destination*.

9.1.7 Foreign buses

A bus can be configured to connect to, and exchange messages with, other messaging networks. In order to do this, a *foreign bus* must be configured.

A foreign bus encapsulates information related to the remote messaging network, such as the type of the foreign bus and whether messaging applications are allowed to send messages to the foreign bus. A foreign bus can represent:

- ▶ A bus in the same cell as the local bus
- ▶ A bus in a different cell from the local bus
- ▶ A WebSphere MQ network

The ability of a bus to be able to communicate with other messaging networks provides several benefits, examples of which are:

- ▶ It enables the separation of resources for different messaging applications that only need to communicate with each other infrequently. This simplifies the administration of the resources for each individual messaging application.
- ▶ It enables a bus to be integrated with preexisting messaging networks.

When buses are interconnected, applications can send messages to destinations that are defined on other buses. Published messages can also span multiple buses, if the links between the buses are configured to allow it.

Note: Care must be taken to avoid creating circular link dependencies (Bus A → Bus B → Bus C → Bus A), when configuring foreign buses within complex topologies. Circular links are not supported by the bus.

Routing definition types

During foreign bus configuration, an administrator defines a routing definition that specifies the type of the foreign bus. This information is used at run time to determine the protocol that will be used to communicate with the foreign bus. The three types of routing definition that can be defined are:

- Direct, service integration bus link

This routing definition type indicates that the local bus will connect directly to another bus. This is shown in Figure 9-9, where the Accounts Department Bus is linked to the HR Department Bus within its own cell and the Payroll Department Bus within another cell.

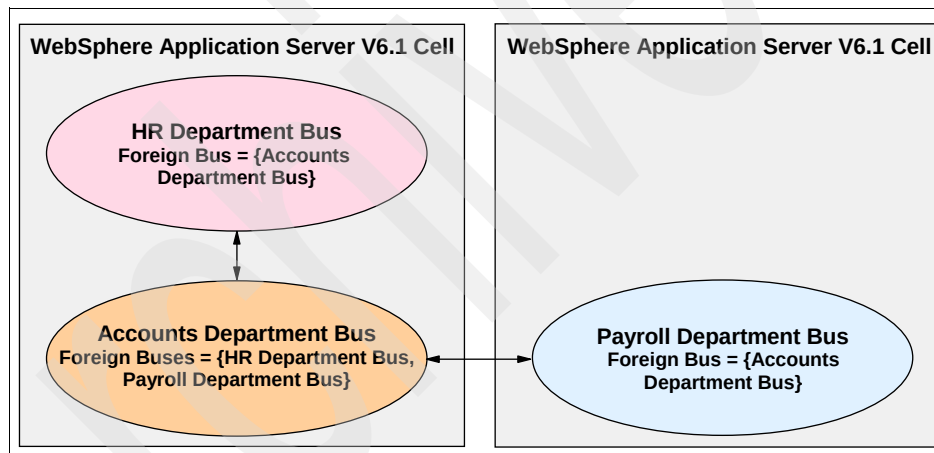


Figure 9-9 Direct, service integration bus links

- Direct, WebSphere MQ link

This routing definition type indicates that the local bus will connect directly to a WebSphere MQ gateway queue manager. This WebSphere MQ queue manager might itself be connected to several other queue managers in a WebSphere MQ network. This is shown in Figure 9-10.

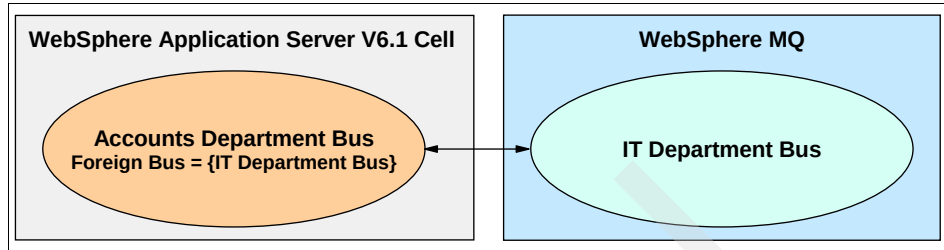


Figure 9-10 Direct, WebSphere MQ link

Note: Connections to WebSphere MQ on z/OS have a special connection type called WebSphere MQ Server that offers advantages over a foreign bus defined as a direct WebSphere MQ link. For more information, see 9.2.7, “WebSphere MQ Servers” on page 592.

► Indirect

The indirect routing definition type indicates that the foreign bus being configured is not directly connected to the local bus. In this situation, the administrator specifies the name of the next bus in the route. This bus can be another bus or a WebSphere MQ network, but it must already be defined in order to configure an indirect routing definition. Ultimately, a message could travel through several intermediate buses before it reaches its destination.

This is shown in Figure 9-11, where the Accounts Department Bus is linked indirectly to the Payroll Department Bus via the HR Department Bus.

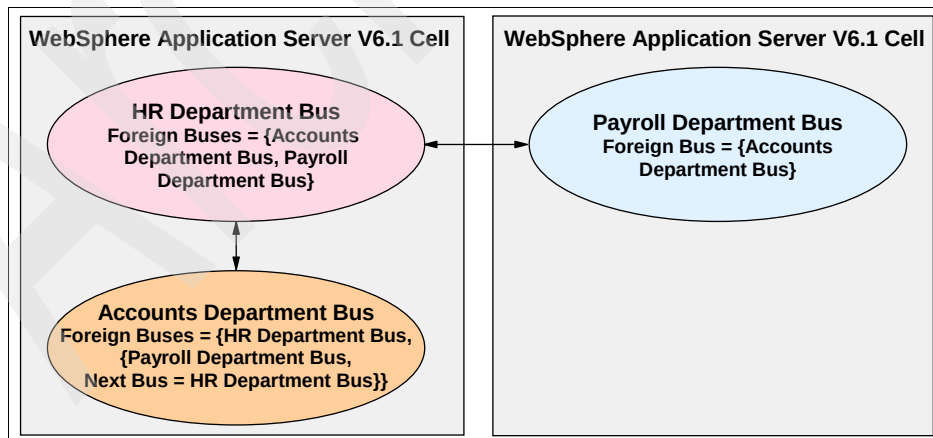


Figure 9-11 Indirect foreign bus link

Foreign bus links

Recall that a service integration bus is simply an architectural concept within a cell. Similarly, when a foreign bus is configured on a bus, it simply describes a link between the two buses at an architectural level.

In order for the two buses to be able to communicate with each other at run time, links must be configured between a specific messaging engine within the local bus and a specific messaging engine, or queue manager, within the foreign bus. When configuring a direct service integration bus link, these links must be configured in both directions in order for the two buses to be able to communicate. At run time, messages that are routed to a foreign bus will flow across the corresponding link. This is shown in Figure 9-12.

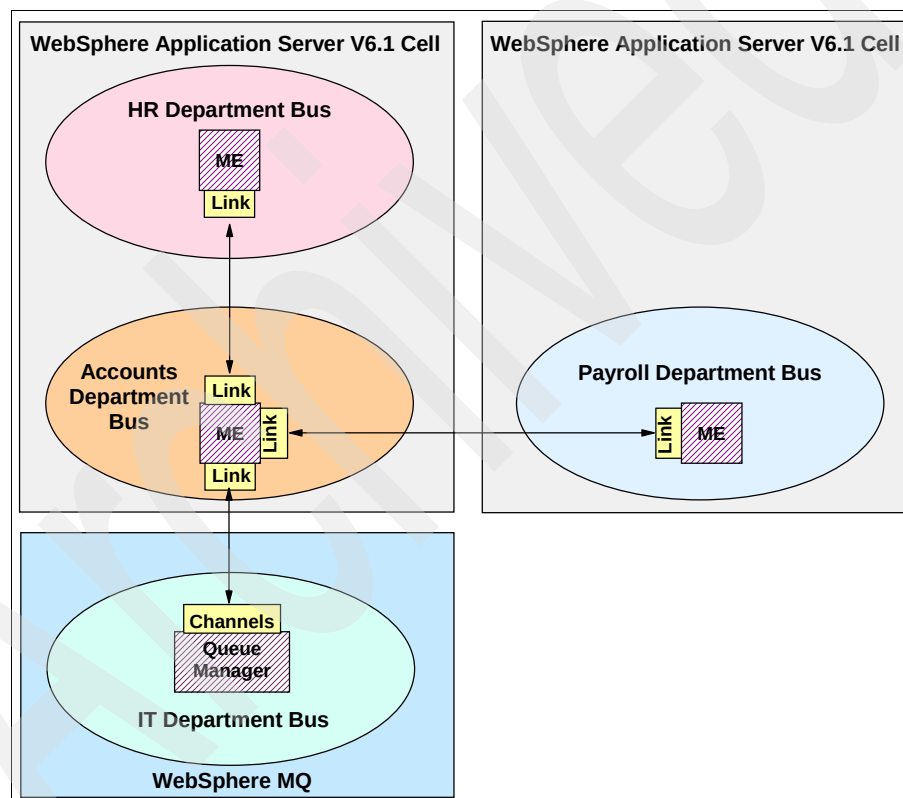


Figure 9-12 Run time view of foreign buses

Note: It is not possible to define multiple links between the local bus and a specific foreign bus.

Foreign buses and point-to-point messaging

Messaging applications that make use of the Point-to-Point messaging model, with destinations that are defined on a local bus, are able to act as both message producers and message consumers. This is shown in Figure 9-13.

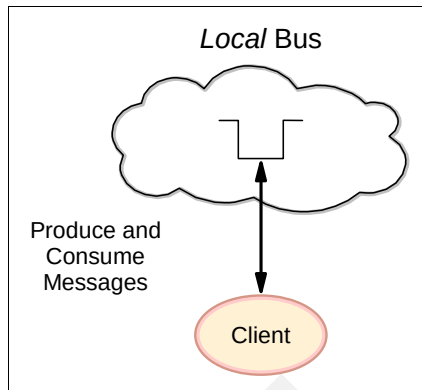


Figure 9-13 Point-to-point messaging on the local bus

However, when a messaging application is making use of the Point-to-Point messaging model with destinations that are defined on a foreign bus, it is only able to act as a message producer. This is shown in Figure 9-14.

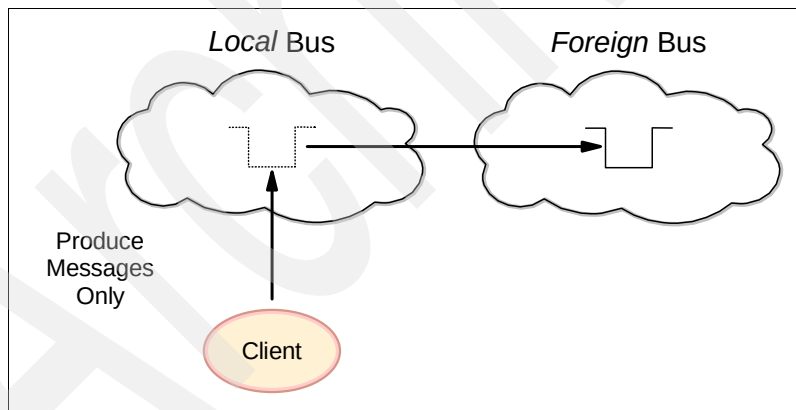


Figure 9-14 Point-to-point message producer for a foreign bus

If a messaging application is required to consume messages from a destination that is defined on a foreign bus, the messaging application must connect directly to the foreign bus. This is shown in Figure 9-15 on page 560.

This is similar to the restrictions placed on WebSphere MQ messaging clients, where a client application is only able to consume messages from a queue by connecting directly to the WebSphere MQ queue manager on which the queue is defined.

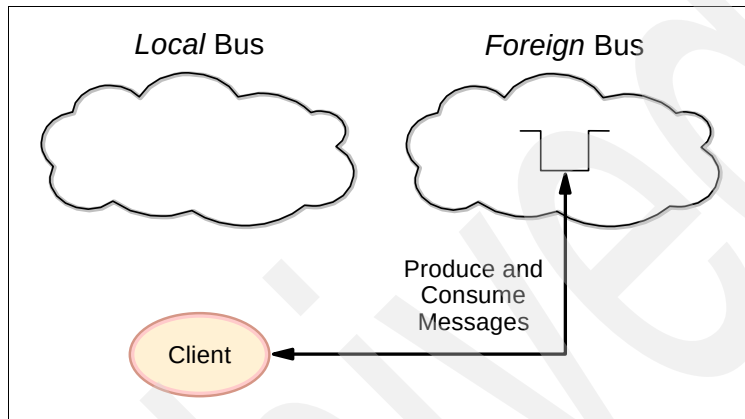


Figure 9-15 Point-to-point messaging on a foreign bus

If the messaging application is unable to connect directly to the foreign bus, then the destinations on the foreign bus must be configured to forward messages to destinations on the local bus. The messaging application is then able to connect to the local bus to consume the messages. This is shown in Figure 9-16.

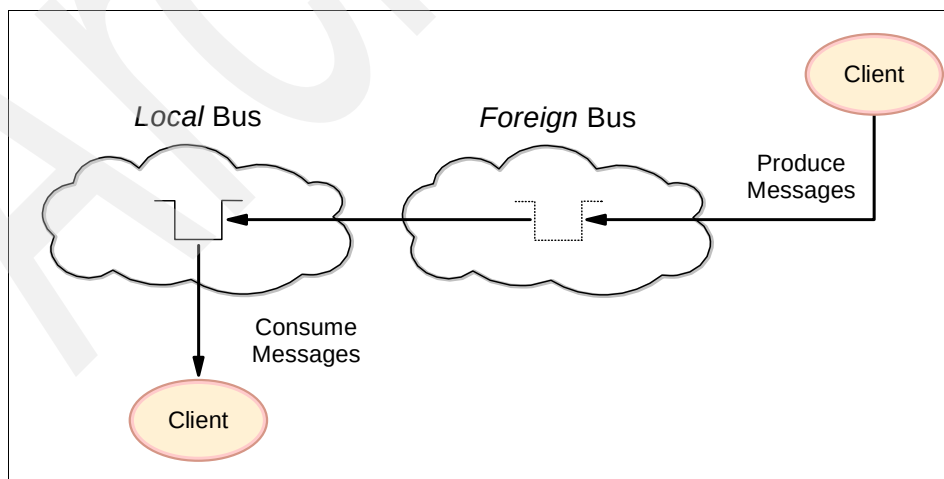


Figure 9-16 Forwarding messages for consumption from the local bus

Foreign buses and Publish/Subscribe messaging

By default, foreign bus links will not flow messages that are produced by messaging applications using the Publish/Subscribe messaging model. It is possible to configure a foreign bus link such that messages published to topic spaces on the local bus will be published on the foreign bus.

9.2 Run time components

At run time, a bus is comprised of a collection of cooperating messaging resources. The sections that follow describe the run time aspects of these messaging resources in more detail.

9.2.1 SIB service

The *SIB service* is a WebSphere Application Server component that is responsible for managing all of the messaging resources that have been associated with a particular application server. However, the SIB service is not associated with a specific bus or messaging engine. Its management tasks include:

- ▶ Managing the life cycle of any messaging related transport chains that have defined within the application server
- ▶ Handling inbound connection requests from external messaging applications

Figure 9-17 shows a SIB service within an application server environment.

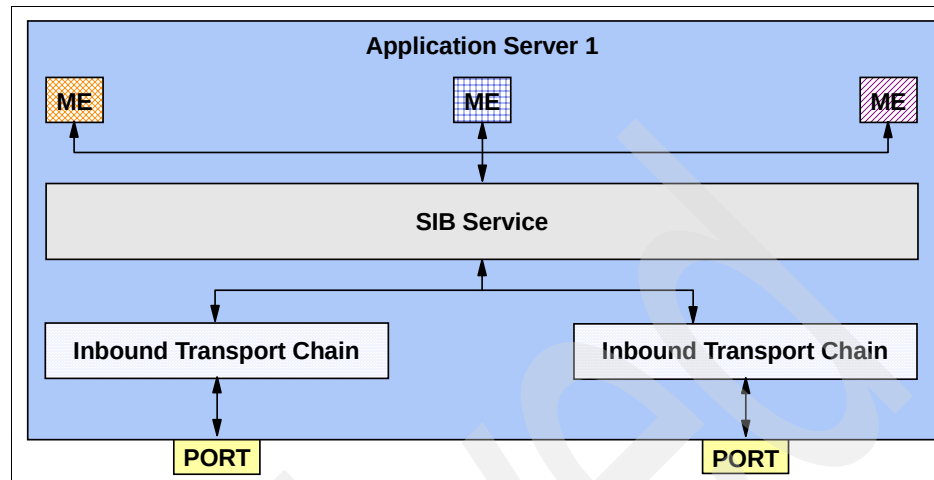


Figure 9-17 SIB service

Every application server has exactly one SIB service. However, by default the SIB service within an application server is disabled. This ensures that the SIB service does not consume resources unnecessarily if the application server is not hosting any messaging resources.

The process of adding an application server as a member of a bus automatically enables its SIB service. This ensures that the SIB service is available to manage the messaging resources that are created as a result of adding the application server as a bus member.

The SIB service can also be manually enabled within an application server that is not a member of a bus. An application server configured in this manner is able to act as a bootstrap server for clients that are running outside of the WebSphere Application Server environment, or for messaging engines that are running in a different cell. Refer to 8.7, “Connecting to a service integration bus” on page 520 for more information regarding bootstrap servers.

Configuration reload

The SIB service also allows certain configuration changes to be applied to a bus, without requiring a restart of the application servers that are hosting components associated with that bus. The configuration changes that can be applied without an application server restart are:

- ▶ Creation, modification, or deletion of a destination
- ▶ Creation, modification, or deletion of a mediation

For example, if a new destination is created on a bus, that destination can be made available for use without needing to restart application servers or messaging engines associated with the bus.

However, the configuration changes that require the affected application servers or messaging engines to be restarted before the changes come into effect include:

- ▶ Creation of a new bus
- ▶ Creation of a new messaging engine
- ▶ Creation of a bus link
- ▶ Creation of a WebSphere MQ link
- ▶ Creation of a WebSphere MQ Server

Note: Each bus that requires this functionality must also be configured to support configuration reload. By default, each bus has configuration reload support enabled. See 9.8.1, “SIB service configuration” on page 613 for more information.

9.2.2 Service integration bus transport chains

The SIB service and any messaging engines running within an application server make use of a variety of transport chains in order to communicate with each other and with client applications. The sections that follow describe the inbound and outbound transport chains used by bus components.

Inbound transport chains

When an application server is created using the default template, a number of inbound transport chains are automatically defined. These transport chains enable messaging clients to communicate with a messaging engine. A messaging client can be a client application or another messaging engine. Table 9-2 describes these transport chains.

Table 9-2 Messaging engine inbound transport chains

Transport chain and associated port	Default port	Client Types	Description
InboundBasicMessaging SIB_ENDPOINT_ADDRESS	7276	Remote messaging engines JMS client applications running in the J2EE client container and using the default messaging JMS provider	This chain allows clients of the specified type to communicate with a messaging engine using the TCP protocol.

Transport chain and associated port	Default port	Client Types	Description
InboundSecureMessaging SIB_ENDPOINT_SECURE_ADDRESS	7286	Remote messaging engines JMS client applications running in the J2EE client container and using the default messaging JMS provider	This chain allows clients of the specified type to communicate securely with a messaging engine using the secure sockets layer (SSL) protocol over a TCP connection. The SSL configuration information for this chain is based on the default SSL repertoire for the application server.
InboundBasicMQLink SIB_MQ_ENDPOINT_ADDRESS	5558	WebSphere MQ queue manager sender channels JMS client applications running in the J2EE client container and using the WebSphere MQ JMS provider	This chain allows clients of the specified type to communicate with a messaging engine using the TCP protocol.
InboundSecureMQLink SIB_MQ_ENDPOINT_SECURE_ADDRESS	5578	WebSphere MQ queue manager sender channels JMS client applications running in the J2EE client container and using the WebSphere MQ JMS provider	This chain allows clients of the specified type to communicate securely with a messaging engine using the secure sockets layer (SSL) protocol over a TCP connection. The SSL configuration information for this chain is based on the default SSL repertoire for the application server.

As discussed in 9.2.1, “SIB service” on page 561, the SIB service is responsible for managing the life cycle of the messaging-related inbound transport chains within an application server. Certain transport chains can be started even if the application server is not hosting any messaging engines. When a transport chain starts, it binds to the TCP port to which it has been assigned and listens for network connections. Table 9-3 describes the circumstances under which the inbound transport chains are started by the SIB service.

Table 9-3 Default transport chain initialization during application server startup

Application server configuration	Transport chains	
	InboundBasicMessaging InboundSecureMessaging	InboundBasicMQLink InboundSecureMQLink
SIB service disabled	Not started	Not started
SIB service enabled No WebSphere MQ links No WebSphere MQ client links	Started	Not started
SIB service enabled WebSphere MQ links or WebSphere MQ client links defined	Started	Started

Figure 9-18 shows the InboundBasicMessaging and InboundSecureMessaging transport chains, and the corresponding ports that they are bound to, within an application server.

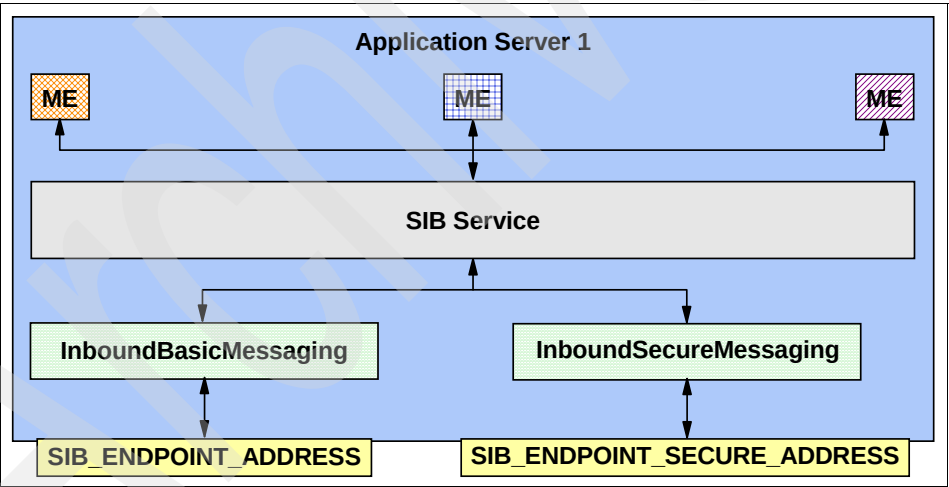


Figure 9-18 Messaging engine inbound transport chains

Outbound transport chains

When you create an application server using the default template, a number of outbound transport chains are automatically defined. These transport chains are also available to JMS client applications running within the J2EE client container. Outbound transport chains are used by messaging clients to establish network

connections to bootstrap servers or to WebSphere MQ queue manager receiver channels. Table 9-4 on page 566 describes these transport chains.

Table 9-4 Default messaging engine outbound transport chains

Transport chain	Description
BootstrapBasicMessaging	This chain is suitable for establishing a bootstrap connection to inbound transport chains within an application server that are configured to use the TCP protocol. An example of such a transport chain is the InboundBasicMessaging chain.
BootstrapSecureMessaging	This chain is suitable for establishing a bootstrap connection to inbound transport chains within an application server that are configured to use SSL over a TCP connection. An example of such a transport chain is the InboundSecureMessaging transport chain. Success in establishing such a connection is dependent on a suitably compatible set of SSL credentials being associated with both this bootstrap outbound transport chain and also the inbound transport chain to which it is connecting. The SSL configuration used is taken from the default SSL repertoire of the application server within which the messaging client is running, or from the relevant configuration file if the messaging client is running within the J2EE client container.
BootstrapTunneledMessaging	This chain can be used to tunnel a bootstrap request through the Hypertext Transfer Protocol (HTTP). Before this transport can be used, a corresponding inbound transport chain must be configured on the bootstrap server.
BootstrapTunneledSecureMessaging	This chain can be used to tunnel a secure bootstrap request through the Hypertext Transfer Protocol (HTTPS). Success in establishing such a connection is dependent on a suitably compatible set of SSL credentials being associated with both this bootstrap outbound transport chain and also the inbound transport chain to which it is connecting. The SSL configuration used is taken from the default SSL repertoire of the application server within which the messaging client is running, or from the relevant configuration file if the messaging client is running within the J2EE client container. Before this transport can be used, a corresponding inbound transport chain must be configured on the bootstrap server.

Transport chain	Description
OutboundBasicMQLink	This chain is suitable for establishing a connection to a WebSphere MQ queue manager receiver channel using the TCP protocol.
OutboundSecureMQLink	This chain is suitable for establishing a secure connection to a WebSphere MQ queue manager receiver channel that has been configured to accept SSL connections. Success in establishing such a connection is dependent on a suitably compatible set of SSL credentials being associated with both this outbound transport chain and also the WebSphere MQ receiver channel to which it is connecting. The SSL configuration for the outbound transport chain is taken from the default SSL repertoire of the application server that is attempting to contact the WebSphere MQ queue manger receiver channel.

When attempting to establish a network connection, a messaging client must use an outbound transport chain suitable for connecting to the corresponding target. For example, the `BootstrapTunneledMessaging` transport chain can only be used to connect to an inbound transport chain that supports bootstrap requests tunneled over the HTTP protocol. Similarly, the `OutboundBasicMQLink` can only be used to connect to a WebSphere MQ queue manager receiver channel. Refer to 8.7, “Connecting to a service integration bus” on page 520 for more information regarding bootstrap servers.

Configuring outbound transport chains within an application server used for bootstrap purposes is considered to be an advanced administrative task. For this reason, these transport chains can only be altered, or new bootstrap transport chains defined, using the `wsadmin` command-line environment.

Outbound transport chains within the J2EE client container environment that are used for bootstrap purpose are not configurable. However, certain attributes of the outbound transport chains that are used to establish SSL connections can be customized.

Secure transport considerations

As discussed previously, additional considerations need to be taken into account when using a transport chain that makes use of the SSL protocol to encrypt the traffic that flows over the connection.

Establishing an SSL or HTTPS connection between messaging engines, or between a messaging engine and a JMS application running within the J2EE

client container, requires a set of compatible credentials to be supplied by both the party initiating the connection, and the party accepting the connection.

Within an application server environment, the credentials used by a secure transport chain can be configured by associating the required SSL repertoire with the relevant SSL channel within the chain. For inbound transport chains, this can be performed using the WebSphere administrative console. By default, secure transport chains within an application server environment are associated with the default SSL repertoire for the cell. When configuring secure communications between two messaging engines, the name of the inbound transport chain on both messaging engines must match in order for the connection to be established. These transport chains must also be configured with compatible SSL credentials. This is true when securing both intra-bus messaging engine connections and inter-bus messaging engine connections.

Within the J2EE client container environment, the credentials used by a secure outbound transport chain are specified in the `sib.client.ssl.properties` file. Every WebSphere profile has its own copy of this file, contained in the properties subdirectory of the profile. The properties contained within this file specify, among other things, the location of the key store and trust store to be used by the outbound transport chain, when attempting to establish a secure connection to a messaging engine.

Note: Any messaging engine that is active on an application server can be contacted by any enabled inbound transport chain. By default, all application servers are created with both secure and insecure transport chains. In order to ensure that a messaging engine can only be contacted using a secure transport chain, it is necessary to either disable or delete the insecure transport chains that are defined on the corresponding application server.

9.2.3 Message stores

A messaging engine must have a message store (and only one) as a place to preserve persistent and non-persistent data for normal operation and for recovery should a failure occur. This message store can be implemented as a data store or as a file store. The process of adding an application server as a member of a bus automatically creates a messaging engine on that application server. As part of that process wizard, a choice is presented as to which implementation of a message store is required.

- ▶ A *data store* is a message store implemented as a set of database tables within a relational database, accessed via a JDBC data source.
- ▶ A *file store* is a message store implemented as a set of flat files within a file system that is accessed directly via the native operating system.

Both types of message store and considerations when choosing between them are discussed in the following sections.

File stores

A file store for a messaging engine is hosted directly on a file system as a set of flat files via the underlying operating system. The messaging engine does not need any other resources to be set up in order to access the file store. The file store uses three levels of data storage in separate files and locations. This is described further in the following sections.

File store files

As can be seen in Figure 9-19, there are three different type of files within a file store: the permanent store file, the temporary store file, and the log file.

- Permanent store file

This contains data that is required to survive a restart of the messaging engine. This will include information about the storage and transmission of persistent messages as well as the persistent messages themselves.

- Temporary store file

This contains temporary data that will not survive a message engine restart, such as any non-persistent messages spilled to the file store to release Java heap memory. The temporary store file is emptied when the message engine starts.

- Log store file

This contains transient data that has not been written to the file, such as information about currently active transactions.

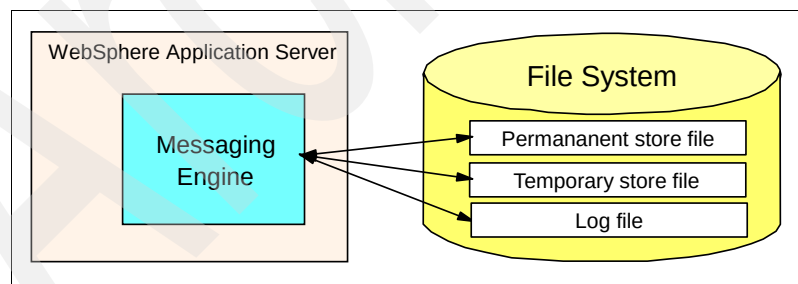


Figure 9-19 Structure of the file store and relationship to the Messaging Engine,

File store location and attributes

The locations of three files that make up the file store can be configured by the administrator, however the default location of the file store will be a subdirectory under:

`${USER_INSTALL_ROOT}/filestores/com.ibm.ws.sib/<me_name>.<me_build>`

The file paths within the subdirectory are store/PermanentStore, store/TemporaryStore, and log/Log.

The log file has a fixed size at run time and does not expand during use. The messaging engine will write data to the log file in a sequential manner, meaning new records are appended to the end. Upon reaching the maximum capacity of the log, the oldest records are overwritten by new records as needed. Any data required to be kept is subsequently written to the permanent and temporary store files as appropriate. Only extremely short-lived data is not moved to a store file. The minimum size for a log file is 10 MB with the default being 100 MB.

Both the permanent and temporary store files have separately configured minimum file sizes of 0 bytes (the default minimum setting is 200 MB). They may also have optional maximum size limits placed on them of at least 50 MB each (the default setting is 500 MB). When created, both the permanent and temporary log files consume file space up to their individual minimum reserve, plus the size of the log file. If this does not meet their maximum allocations, then the store files are free to grow. This growth is unlimited if a maximum allocation has not been set.

Note: For a production system, maximum and minimum limits should be applied and be set to the same value so that the file sizes are stable. This would prevent unlimited growth from filling up the file system, and allow the messaging engine to continue to operate unaffected should the file system fill up due to external causes.

The default settings and configuration for a file store is designed to be adequate for a typical messaging workload without the need for any administration. However, it is up to the administrator to make sure that enough space is allocated to the file store components for predictable and smooth operation of the messaging engine. To improve the performance and availability of the log or store files, the file store attributes can be modified to affect sizing and placement of the files. This can be done at creation of the filestore, or later on.

Note: Optimal operation of a messaging engine cannot be guaranteed where the file store is subject to a compressing file system, such as Windows NT® with the Compress this directory option active. In a production system, the use of file system compression should be avoided.

File store access and high availability considerations

A messaging engine has exclusive access of its own file store, and a file store can only be used by the messaging engine that created it. Each file store contains uniquely identifying information about its messaging engine. An instance of a messaging engine will open its file store with an exclusive lock to prevent other instances of the same messaging engine from trying to use the file store at the same time. This situation might arise if there was an accidental activation of a messaging engine on multiple servers within a cluster. When the instance of the messaging engine stops for any reason (either controlled or server failure), the file store's files are closed, allowing another instance to open the file store.

The major consideration for high availability of a file store is the file system it is placed in. The recommendation is to use hardware or software based facilities to maximize the availability of the file systems themselves, such as the use of Storage Area Networks (SAN).

WebSphere Application Server V6.1 supports either cluster-managed or networked file systems. Cluster-managed file systems use clustering and failover of shared disks to ensure high availability of files and directories. Networked file systems use remote servers to store and access files as though it were a local server. Make sure that the file system in use supports access locking to ensure integrity of the file store components, particularly the log file by the use of exclusive locks.

Note: Neither the WebSphere administrative console or the messaging engine can check that the file store configuration is correct. Errors will only surface at run time, so we recommend that the administrator conduct a check and thorough failover testing. In particular, ensure that all members of a cluster have universal access to the directories containing the file store components.

Deleting a file store

When a messaging engine is removed, the file store files are not automatically removed with it and must be located and deleted manually in order to reclaim the files space. The default file store directory names contain the *Universal Unique Identifier (UUID)* of the messaging engine. It is possible to destroy and recreate a messaging engine of the same name without having to manually remove the

old file store as the UUID (and so the file store directory names) will have changed. Delete the file store files by using the facilities of the operating system.

Backing up and restoring a file store

A file store is made up of simple flat files. As such, backing up and restoring these files can be done using a backup tool or facilities of the operating system.

Note: It is important that the permanent store file, temporary store file, and log file of a file store be backed up and restored as one unit and not individually. Also, please make sure that the messaging engine has been stopped before performing a backup or restore. To do otherwise might result in significant data corruption.

Reduction of file store sizes

While it is possible to reduce the file size settings of the file store components in the configuration, it is not possible for the files to actively shrink or compress their contents. When the configuration has been changed and the messaging engine restarted, the messaging engine will attempt to apply the new settings. If the files are still too big due to their contents, a message is written to SystemOut.log and the existing settings are kept. The messaging engine will attempt to apply the new settings each time it is started.

Note: As stated previously, messaging engine problems may occur if the file store file sizes are too small. Care must be taken to make sure the sizes are adequate for the expected messaging workload.

Failover of messaging engine between V6 and V6.1

As WebSphere Application Server V6.0 does not support file stores, it is not possible to fail over a messaging engine with a file store to a V6.0 server. To prevent this, the cluster should be divided into sets of servers at different versions, and the high availability policy of the messaging engine restricted to the servers at V6.1.

Data stores

A data store can be used for a messaging engine, hosted within an embedded Cloudscape database. A JDBC data source to access this database is also defined on the server that has been added to the bus. These defaults allow the messaging engine to run without any further configuration.

However, while adding a bus member, it is possible to specify the JNDI name of a different data source for use by the messaging engine. The sections that follow

describe the issues that must be considered when deciding which RDBMS to use as a data store.

Data store location

The data store can be located on the same host as the messaging engine with which it is associated, or it can be located on a remote host. The decision of where to locate the data store might depend on the capabilities of the RDBMS that host the data store. For example, the embedded Cloudscape database must run within the same application server process on which the messaging engine runs.

Note: Check with your database administrator to ensure that your RDBMS supports remote access from JDBC client applications.

The location chosen for the data store can have an impact on the overall performance, reliability, or availability characteristics of the bus components. For example, a data store located on the same host as the messaging engine with which it is associated can provide higher persistent message throughput by avoiding flowing data over the network to the data store. However, such a configuration might not provide the availability required, because failure of the host would mean that both the messaging engine and its data store would become unavailable.

Figure 9-20 shows the various options available when deciding where to locate a data store. The messaging engine in application server 1 uses the default Cloudscape data store, running in the same process as the application server. The messaging engine in application server 2 uses a data store hosted by a DB2 instance running on the same host as node 1. The messaging engine in application server 3 uses a data store hosted by a DB2 instance running on a remote host.

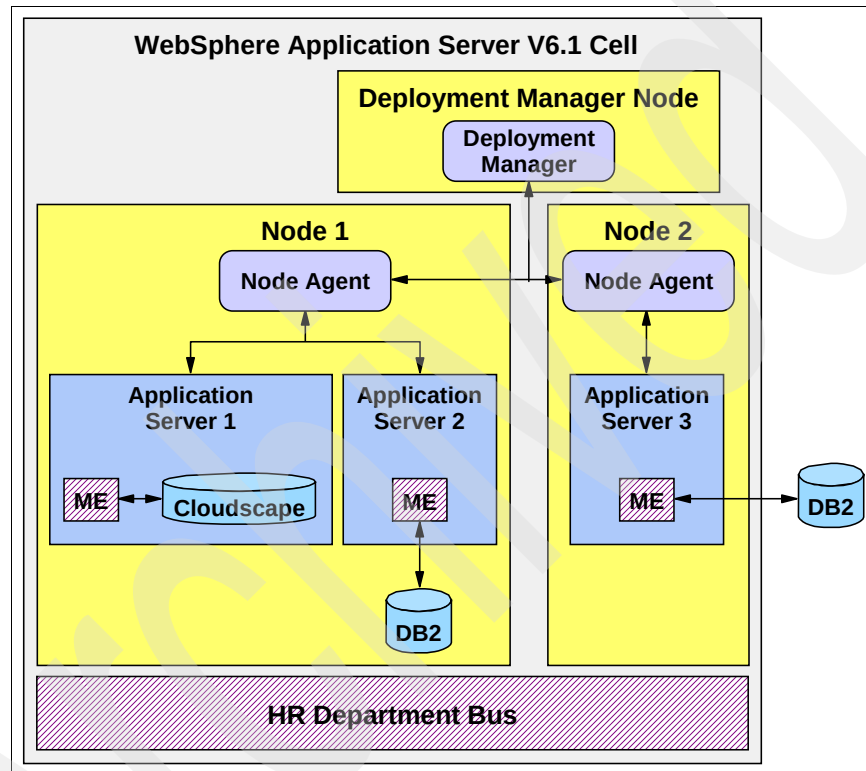


Figure 9-20 Data store locations relative to the associated messaging engine

Data store access

Each messaging engine must have exclusive access to the tables defined within its data store. This can be achieved, either by using a separate database as the data store for each messaging engine, or by partitioning a single, shared, database into multiple data stores using unique schema names for each data store.

Deciding which of these mechanisms to use depends on the capabilities of the RDBMS that will host the data store. For example, the embedded Cloudscape database does not support concurrent access by multiple processes.

Note: Check with your database administrator to ensure that your RDBMS supports shared access from JDBC client applications and that it allows schema names to be specified on a JDBC connection. DB2 and Network Cloudscape support this functionality.

For databases that do not allow a schema name to be specified on a JDBC connection, multiple messaging engines share database access by each messaging engine using a different user ID when connecting to the database.

Archived

Figure 9-21 on page 576 shows the options available when deciding whether to use exclusive access or shared access to a data store. The messaging engine in application server 1 has exclusive access to the database hosting its data store. The messaging engines in application servers 2 and 3 have shared access to the database hosting their data stores. This shared database has been partitioned into separate schemas, with each messaging engine accessing the data store tables within a different schema.

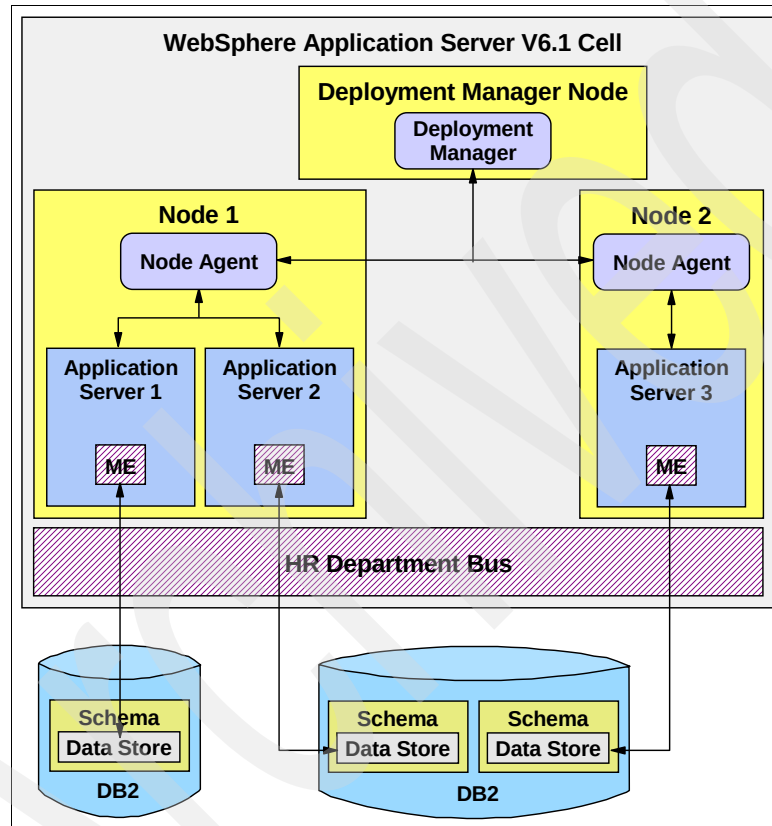


Figure 9-21 Exclusive and shared access to data stores

Data store tables

The messaging engine expects its data store to contain a set of specific tables, each of which has a specific table definition. Each messaging engine can be configured to create the tables within its data store, if they are not already present. During initialization, a messaging engine connects to its data store and checks for the required tables. If the messaging engine has the functionality to create tables, and they do not exist, it attempts to create the tables.

Some organizations allow a database administrator to perform only certain tasks on a database, such as creating tables. In this situation, the database administrator can use the **sibDDLGenerator** command to generate the DDL statements required to create these tables. The **sibDDLGenerator** command is located in the \bin\ subdirectory of the WebSphere installation directory. Refer to the WebSphere Information Center for a full description of the **sibDDLGenerator** command.

Note: In order for the messaging engine to be able to create the required tables within its data store, the user ID for the database must have sufficient privileges. Please refer to the WebSphere Information Center for a full description of the database privileges required in order for the messaging engine to access the data store.

Table 9-5 describes the tables defined within the data store for a messaging engine.

Table 9-5 Messaging engine data store tables

Table name	Description
SIBOWNER	Ensures exclusive access to the data store by an active messaging engine.
SIBCLASSMAP	Catalogs the different object types in the data store.
SIBLISTING	Catalogs the SIBnnn tables.
SIBXACTS	Maintains the status of active two-phase commit transactions.
SIBKEYS	Assigns unique identifiers to objects in the messaging engine.

Table name	Description
SIBnnn, where nnn is a number	Contains persisted objects such as messages and subscription information. These tables hold both persistent and nonpersistent objects, using separate tables for the different types of data, according to the following convention: ▶ SIB000 Use this name for the table that contains information about the structure of the data in the other two tables. ▶ SIB001 Use this name for the table that contains persistent objects. ▶ SIB002 Use this name for the table that contains non-persistent objects saved to the data store to reduce the messaging engine memory requirement.

Note: When you remove a messaging engine, WebSphere Application Server does not automatically delete the tables in its data store. To reuse this data store with another messaging engine, delete the tables within the data store manually.

Considerations when choosing the message store type

A file store has several advantages over a data store:

- ▶ Better performance
A file store can often achieve higher throughput than a data store due to smaller overhead of the file system as compared to that of a relational database.
- ▶ Lower administration requirements
There are little or no administration requirements with the use of a file store. A data store may require ongoing database administration depending on the messaging workload profile to maintain optimum performance.
- ▶ Lower deployment costs
Costs associated with database server licensing and the services of a database administrator do not apply to a file store as there is no database.

However, if an organization already has existing database resources and skills, it may be preferable to use a data store in order to utilize those skills. This would apply more to larger companies with a strong team of database administrators.

From a technical standpoint, applications may share the messaging engine's JDBC connection to a data store to improve performance using a one-phase commit optimization. This is not possible with a file store.

Security for both types of message store can be achieved utilizing the facilities of the underlying infrastructure. For example, file stores can use a secure, possibly encrypted network attached drive to achieve both electronic and physical security. Data stores can use be secured using the available database security facilities.

9.2.4 Exception destinations

If a messaging client encounters a problem when attempting to consume a message from a bus destination, message delivery has failed. The message can be placed back on the bus destination for redelivery. Use the `maximumFailedDeliveries` property on a bus destination to determine the number of times a message can fail delivery. The default value of this property is five.

An *exception destination* handles undeliverable messages. Both queue and topic space destinations can define an exception destination. If a message cannot be delivered to its intended bus destination, it is rerouted to the specified exception destination. This mechanism prevents the loss of messages that cannot be delivered.

Note: Messages can also be placed on an exception destination for a variety of other reasons, examples of which include:

- ▶ When a destination is deleted, any messages on the destination are placed on the exception destination, unless the bus has been configured to discard them.
- ▶ When a message is received from a foreign bus, the message is placed on the exception destination if the target destination has reached its high message threshold.

Each messaging engine has a default exception destination of `_SYSTEM.Exception.Destination.<Messaging_engine_Name>`. By default, all bus destinations that have message points on a messaging engine use the default exception destination for that messaging engine when rerouting undeliverable messages. This enables administrators to access all of the undeliverable messages for a messaging engine in one place.

However, an administrator can also configure a bus destination to use a nondefault exception destination. This enables administrators to access all of the undeliverable messages for a specific destination in one place, allowing for more fine-grained management of undeliverable messages.

When configuring a destination to use a non-default exception destination, the exception destination specified can be a local or a remote bus destination. We also recommend that this destination is a queue destination and that it exists prior to the creation of the bus destination with which it is associated. If the exception destination specified has been deleted when a destination attempts to reroute an undeliverable message, the undeliverable message is rerouted to the default exception destination for the message engine.

Note: It is not possible to delete a default exception destination from a bus. This ensures that there is always a default exception destination available on each messaging engine within the bus.

Note: Errors might occur as a message traverses the bus to its target destination. In this situation, the messaging engine handling the message attempts to redeliver the message. However, if the messaging engine determines that the target destination is unreachable, it can place the message on its default exception destination. For this reason, all exception destinations on the bus must be monitored to ensure that problem messages are processed appropriately.

When message order is important, it might be necessary to configure a bus destination not to use an exception destination. In this case, any messages that cannot be delivered to the target destination are not rerouted, and will be redelivered repeatedly. This has the effect of blocking the delivery of subsequent messages to the bus destination in question. For this reason, such a configuration should be used with caution.

Note: Publication messages arriving at a topic space destination for which there are no subscribers are not considered to be undeliverable. Such messages are discarded.

9.2.5 Service integration bus links

As discussed in 9.1.7, “Foreign buses” on page 555, defining a foreign bus on a bus simply defines a link between the two buses at an architectural level. When the foreign bus in question represents another bus, the link is implemented at run time by establishing a connection between a messaging engine from each of the

buses. This link is configured on a messaging engine by defining a *service integration bus link*. A service integration bus link encapsulates the information required to communicate with a specific messaging engine, within a specific foreign bus.

When configuring a service integration bus link, it must be associated with the target foreign bus definition. The foreign bus definition with which it is associated enables the service integration bus link to determine the name of the target bus. This is shown in Figure 9-22.

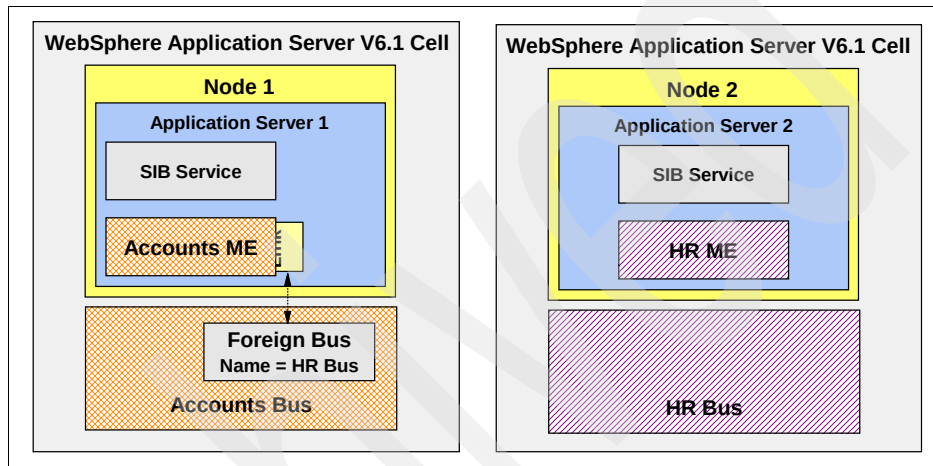


Figure 9-22 Association between a service integration bus link and a foreign bus

This requirement also determines the order in which these objects must be defined. The foreign bus must be defined within a bus before a corresponding service integration bus link can be configured on a messaging engine.

Note: The name specified for the foreign bus must exactly match the real name of the target bus.

The names of each of the buses involved in the link must also be unique. For this reason, if two buses within separate cells need to be linked, care must be taken when naming each of the buses.

When attempting to establish the connection, the messaging engine within the local bus always attempts to connect to the foreign bus as though it were a remote client, even if the foreign bus is defined within the same cell. For this reason, a list of provider endpoints must also be specified when configuring the service integration bus link. These provider endpoints are used by the messaging engine in the local bus to connect to a bootstrap server in the foreign bus. For

more information about the bootstrap process, refer to 8.7, “Connecting to a service integration bus” on page 520.

The service integration bus link is also required to specify the name of the messaging engine on the target bus with which to connect. The messaging engine in the local bus uses the bootstrap server to locate the target messaging engine in the foreign bus. Figure 9-23 shows this process.

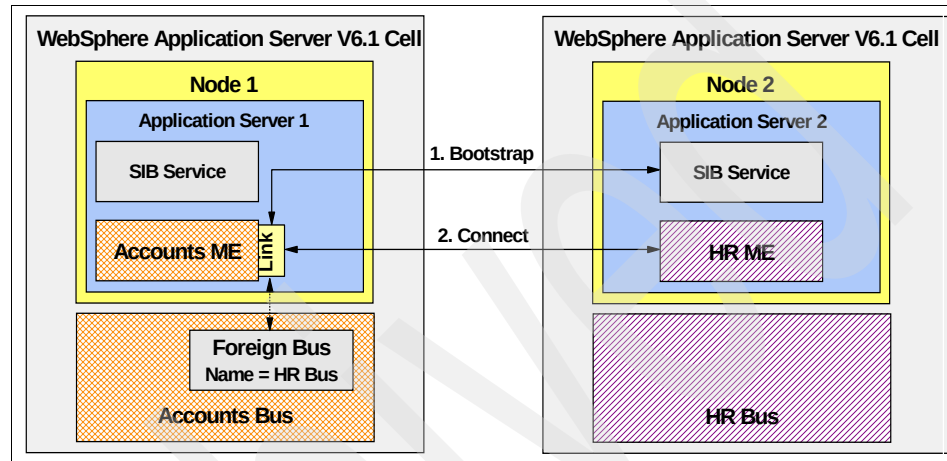


Figure 9-23 Bootstrapping during service integration bus link initialization

Once again, this requirement imposes an order in which the various configuration tasks must be performed. Each of the buses involved in the link must have at least one bus member defined before a service integration bus link can be configured.

The final requirement when configuring a service integration bus link is that the link must be configured in both directions in order for the two buses to communicate at run time. This is shown in Figure 9-24 on page 583.

Note: The name specified for the service integration bus link within both buses must be the same.

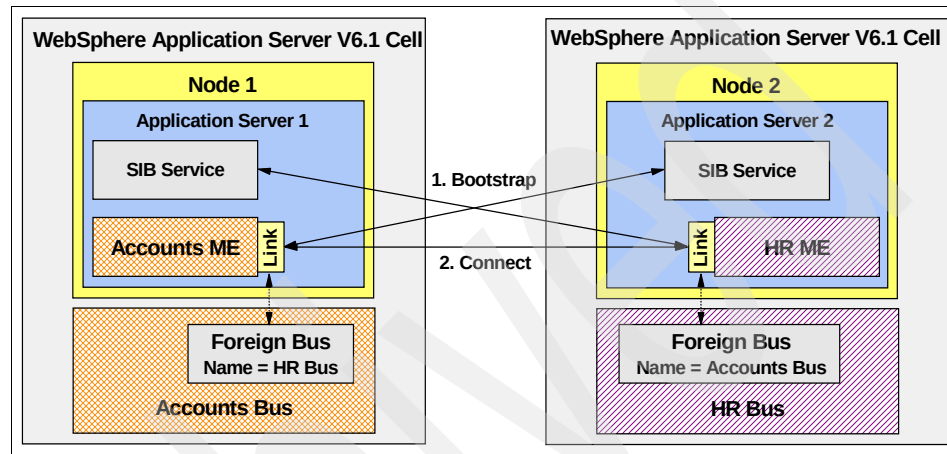


Figure 9-24 Defining a service integration bus link in both directions

Note: If the transport chain used by the service integration bus link encrypts its traffic using SSL, the names of the target inbound transport chain on each link must be the same. The transport chain specified must also be configured identically on each bus to ensure that compatible SSL credentials are used when establishing the link.

Topic space mappings

By default, a service integration bus link only flows messages across the link that are addressed to a queue destination on the foreign bus. In order to flow publication messages across the service integration bus link, topic space mappings need to be configured on the foreign bus definition.

These mappings define the topic space destination within the local bus for which publication messages are passed over the link. They also define the topic space destination on the foreign bus to which these publication messages are addressed. Refer to the WebSphere Information Center for more information regarding the definition of topic space mappings.

9.2.6 WebSphere MQ links

Defining a foreign bus on a bus simply defines a link between the two buses at an architectural level. When the foreign bus in question represents a WebSphere MQ network, the link is implemented at run time by establishing sender and receiver channels between a specific messaging engine and a WebSphere MQ queue manager. These channels are configured on a messaging engine by defining a *WebSphere MQ link*.

To a messaging engine configured with a WebSphere MQ link, the WebSphere MQ queue manager appears to be a foreign bus. To the WebSphere MQ queue manager, the messaging engine appears to be another WebSphere MQ queue manager. When configuring a WebSphere MQ link, an administrator must specify a virtual queue manager name. This is the queue manager name by which the messaging engine will be known to the remote WebSphere MQ queue manager. The WebSphere MQ queue manager is completely unaware that it is communicating with a messaging engine.

When you configure a WebSphere MQ link, you must associate it with the target foreign bus definition. The name specified for the foreign bus does not need to match the name of the target WebSphere MQ queue manager. However, specifying a name for the foreign bus that matches the target WebSphere MQ queue manager simplifies the routing of messages across the link.

Figure 9-25 shows a high level view of a WebSphere MQ link. Notice that the name of the foreign bus with which the WebSphere MQ link is associated matches the name of the target WebSphere MQ queue manager.

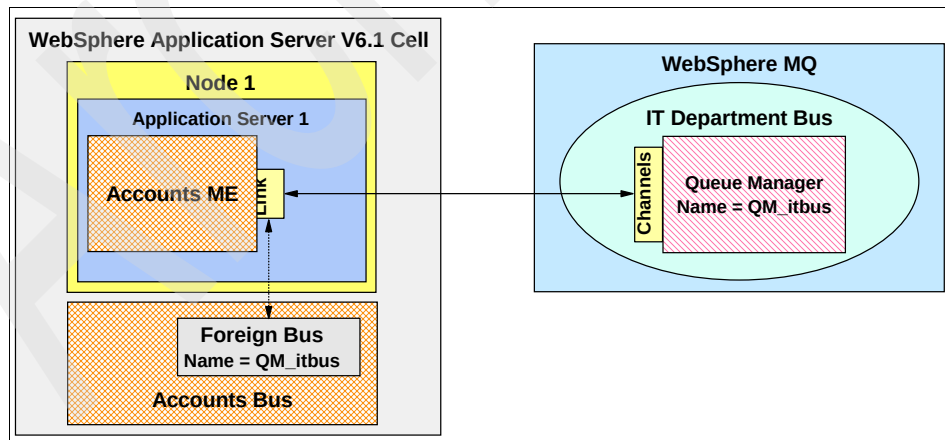


Figure 9-25 Overview of a WebSphere MQ link

WebSphere MQ link sender channel

The WebSphere MQ link sender channel establishes a connection to a receiver channel on the target WebSphere MQ queue manager. It converts messages from the format used within the bus, to the format used by WebSphere MQ, and then sends these messages to the receiver channel on the target WebSphere MQ queue manager. For a full description of how messages are converted as they traverse the WebSphere MQ link, refer to the WebSphere Information Center. The WebSphere MQ link sender channel emulates the behavior of a sender channel in WebSphere MQ. This is shown in Figure 9-26.

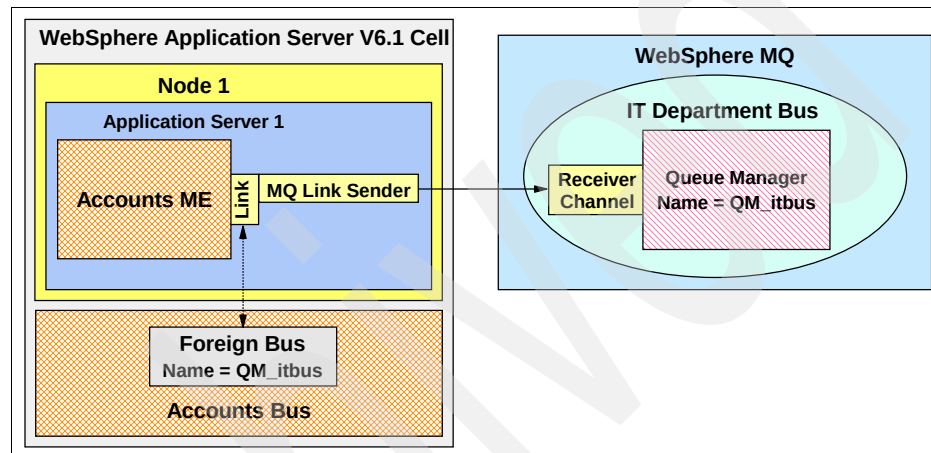


Figure 9-26 WebSphere MQ link sender channel

Note: It is only necessary to define a WebSphere MQ link sender channel if messages are required to be sent from the bus to the WebSphere MQ network.

When you configure a WebSphere MQ link sender channel, you are required to specify the following information:

- ▶ A name for the channel, which must exactly match, including case, the name of the receiver channel defined on the target WebSphere MQ queue manager
- ▶ The host name or IP address of the machine hosting the target WebSphere MQ queue manager
- ▶ The port number on which the target WebSphere MQ queue manager is listening for inbound communication requests
- ▶ An outbound transport chain

Note: If the receiver channel on the target WebSphere MQ queue manager accepts only SSL connections, you must associate the transport chain with a suitably compatible set of SSL credentials.

WebSphere MQ link receiver channel

The WebSphere MQ link receiver channel allows a sender channel within a WebSphere MQ queue manager to establish a connection to a messaging engine within the bus. It converts messages from the format used within WebSphere MQ, to the format used by the bus. For a full description of how messages are converted as they traverse the WebSphere MQ link, refer to the WebSphere Information Center. The WebSphere MQ link receiver channel emulates the behavior of a receiver channel in WebSphere MQ. This is shown in Figure 9-27.

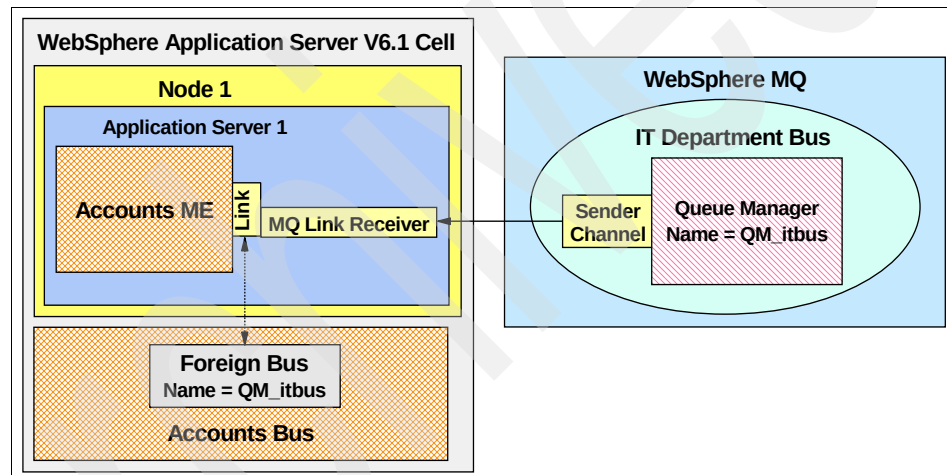


Figure 9-27 WebSphere MQ link receiver channel

Note: It is only necessary to define a WebSphere MQ link receiver channel if messages are required to be sent from the WebSphere MQ network to the bus.

When configuring a WebSphere MQ link receiver channel, the following information is required: a Name for the channel, which must exactly match, including case, the name of the sender channel defined on the target WebSphere MQ queue manager

The inbound transport chain with which the sender channel on the WebSphere MQ queue manager communicates is dependent on the configuration of the

WebSphere MQ sender channel. The WebSphere MQ administrator should be consulted to ensure that the sender channel is configured appropriately. As discussed in “Inbound transport chains” on page 563, the InboundBasicMQLink transport chain defaults to listening on port 5558 for connections from WebSphere MQ, and the InboundSecureMQLink transport chain defaults to listening on port 5578 for connections from WebSphere MQ.

MQ Publish/Subscribe broker profile

By default, a WebSphere MQ link only flows messages across the link that are addressed to a queue destination on the WebSphere MQ network. To flow publication messages across the WebSphere MQ link, configure a *publish/subscribe broker profile* for the WebSphere MQ link. A Publish/Subscribe broker profile allows *topic mappings* to be defined. These topic mappings define the topic names for which publication messages will be flowed across the WebSphere MQ link. Please refer to the WebSphere Information Center for more information about the definition of topic mappings within a publish/subscribe broker profile.

Addressing destinations across the WebSphere MQ link

There are several issues that must be considered when addressing a message to a destination that will flow across a WebSphere MQ link. These issues exist because of the differences in naming structure between the bus and WebSphere MQ.

WebSphere MQ has a two-level addressing structure, as follows:

- ▶ Queue manager name
- ▶ Queue name

Each of these elements within WebSphere MQ is limited in length to 48 characters. Within the bus, a destination can be uniquely identified using the following elements:

- ▶ Service integration bus name
- ▶ Destination name

The bus places no length restrictions on these elements.

The difference in the allowable lengths of the various naming elements causes problems when a messaging application running in one environment attempts to address a message to a destination defined in the other environment, across the WebSphere MQ link. These issues are discussed in the sections that follow.

WebSphere MQ to service integration bus addressing

Messages that are sent from a WebSphere MQ application to a bus destination which has a name greater than 48 characters in length must have some means of using the shorter name used in WebSphere MQ to address the long name used in the bus.

The bus uses an alias destination to map between the shorter name used by WebSphere MQ, and the longer name used by the bus. A WebSphere MQ client application can address a message to an alias destination within a bus that is defined with a short name of less than 48 characters. The alias destination then maps this message onto the destination defined with a long name of greater than 48 characters.

Service integration bus to WebSphere MQ addressing

Another problem can happen when a messaging client is required to address a message to a queue defined on an arbitrary queue manager within the WebSphere MQ network. For example, when defining JMS destinations for use by JMS client applications, it is only possible to specify the name of the bus on which the target destination is defined, and the name of the destination. If the destination exists within the WebSphere MQ network, the name of the foreign bus is specified as the bus name. However, if the target queue is not defined on the queue manager to which the WebSphere MQ link connects, additional information is required in order to address messages to the correct queue.

To solve this problem, when defining a JMS queue or an alias destination that represents a queue on a WebSphere MQ network, use a special format for the target queue name, of the form: *<queue>@<queue manager>*. These destination names are only parsed by the WebSphere MQ link, which uses the information to determine which values to place in the target queue and queue manager fields of the message header.

In the most simple case, the name specified for the foreign bus matches the name of the queue manager on which the target queue is defined. When this is the case, only the name of the target queue needs to be specified. If no queue manager name is applied as a suffix, then the foreign bus name will be added as the queue manager name by default. This is shown in Figure 9-28 on page 589.

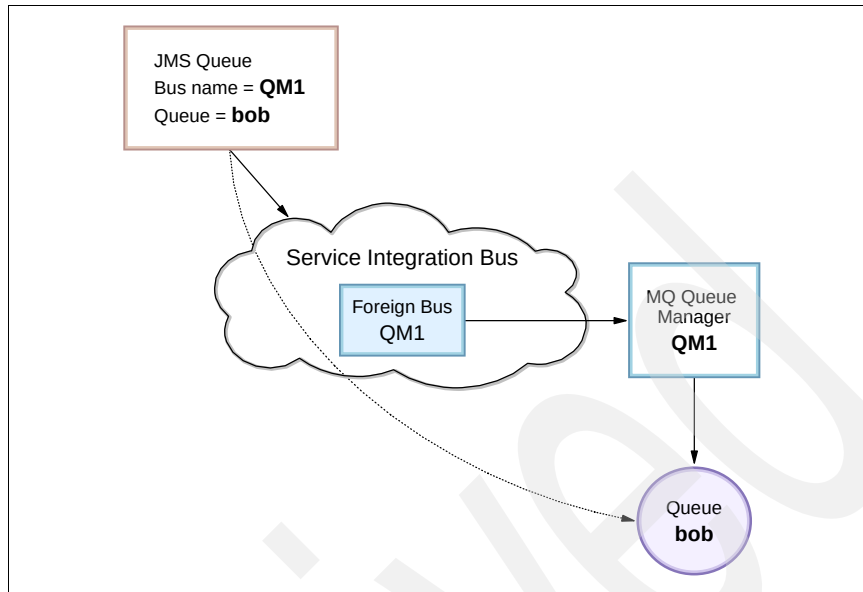


Figure 9-28 Simple WebSphere MQ addressing

This is still the case, even if the WebSphere MQ queue manager on which the target queue is defined, is not the same queue manager to which the WebSphere MQ link connects. This is shown in Figure 9-29.

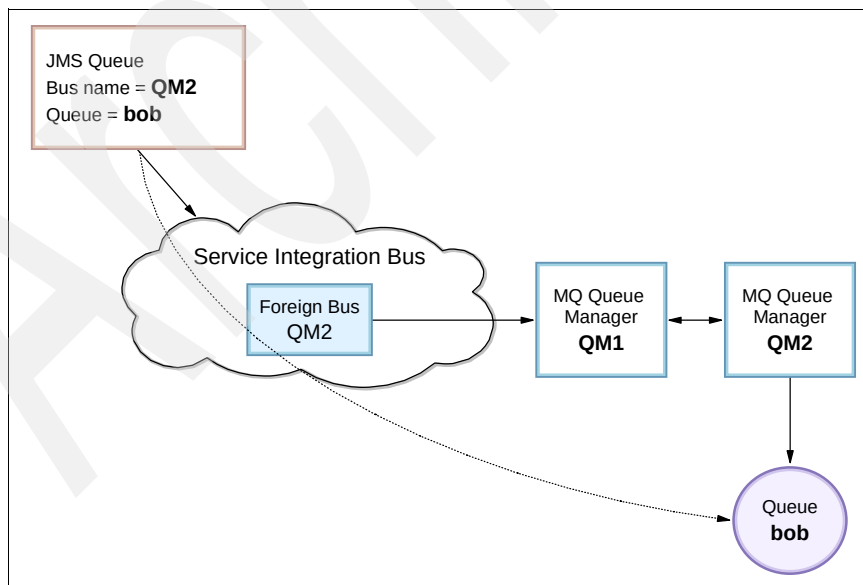


Figure 9-29 Simple WebSphere MQ addressing

When the name specified for the foreign bus does not match the name of the queue manager on which target queue is defined, the queue manager name must be included as part of the queue name using the format described previously. This allows the message to be appropriately routed by WebSphere MQ once the message has left the bus. This is shown in Figure 9-30.

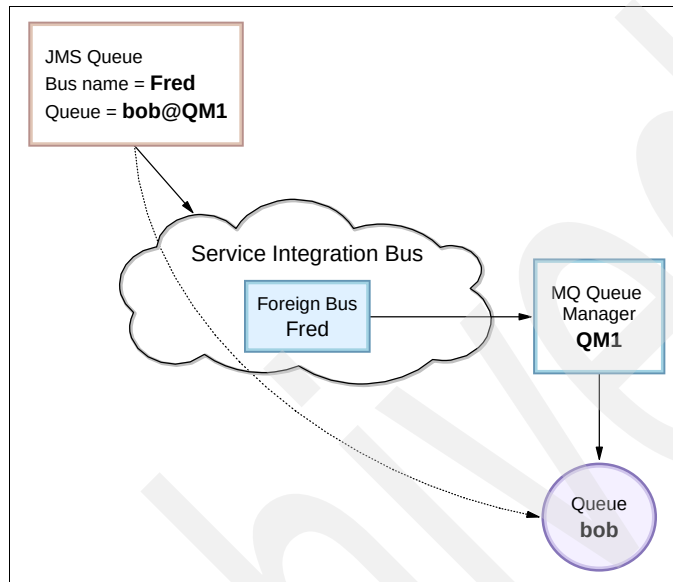


Figure 9-30 Advanced WebSphere MQ addressing

This mechanism enables a messaging client to address a message to a queue that is defined on any queue manager within the WebSphere MQ network. This is shown in Figure 9-31 on page 591.

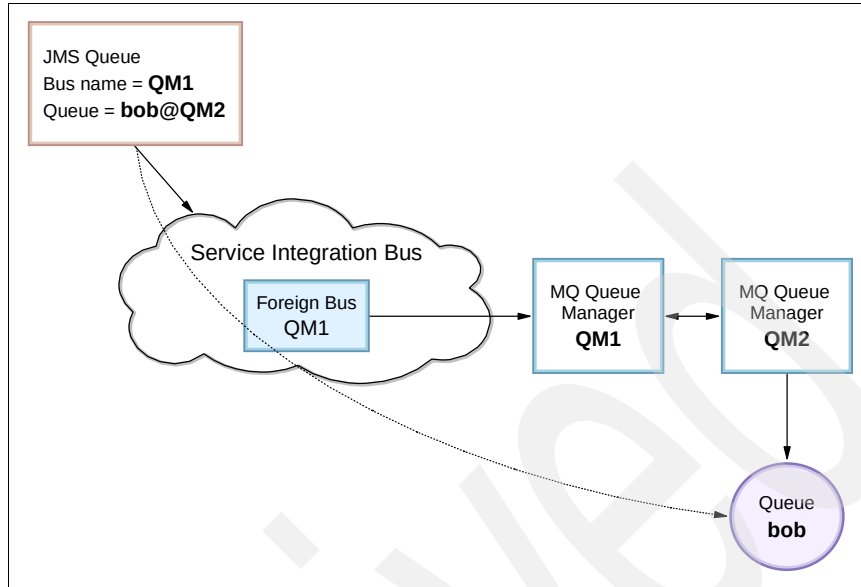


Figure 9-31 Advanced WebSphere MQ addressing

Note: The naming mechanism described within this section can only be used to address messages to destinations defined within WebSphere MQ. It must not be used to attempt to address messages to destinations defined on another bus. An *indirect* foreign bus must be used for that purpose.

WebSphere MQ client links

A WebSphere MQ client link enables a messaging engine to act as a WebSphere Application Server V5.x embedded JMS Server. This function is provided as an aid to the migration of V5.x to V6 and should not be used for any other purpose.

A WebSphere MQ client link enables any applications that are installed and configured on V5.x, using V5.x JMS resources, to continue to function as normal after the V5.x JMS server has been migrated to V6.

The process of migrating a V5.x node that contains an embedded JMS server will remove that JMS server and create a bus with a WebSphere MQ client link. Queues previously defined on the V5.x embedded JMS server will be created automatically on the bus.

See the Information Center topic *Migrating from version 5 embedded messaging* for more information.

You should not need to create a WebSphere MQ client link manually. Use the one created automatically for you by the migration process.

Important: We recommend that you replace all V5.x JMS resources with v6.0 default messaging provider JMS resources as soon as possible. Once all resources have been changed, it is possible to delete the WebSphere MQ client link, as all applications will be using the V6.0 default messaging provider directly.

9.2.7 WebSphere MQ Servers

New in V6.1: For those of you who wish to access WebSphere MQ on a z/OS platform, there is a new mechanism called WebSphere MQ Server that allows applications to take advantage of the high availability and load balancing features of the MQ queue sharing groups that the z/OS implementation of MQ provides.

An alternative to using an WebSphere MQ link when connecting to WebSphere MQ V6 on a z/OS platform is the use of a new type of server called a WebSphere MQ Server. This is a special type of sever that can be added to a bus and used in place of an MQ link to take advantage of the advanced load balancing and high availability features of the z/OS based MQ shared queue groups.

An MQ shared queue group is a collection of queues that can be accessed by one or more queue managers. Each queue manager that is a member of the shared queue group has access to any of the shared queues. This has the advantages of high availability and workload balancing, as queue managers can fail over to one another as they become too busy or unavailable. For more information about MQ shared queue groups, please refer to *WebSphere MQ in a z/OS Parallel Sysplex Environment*, SG24-6864.

The other advantage of the WebSphere MQ Server over an MQ link is that the server does not depend on any one designated messaging engine. This type of connectivity to MQ can tolerate the failure of any given message engine as long as another is available in the bus, increasing robustness and availability.

Restriction: This type of connectivity will only work when communicating with z/OS installations of WebSphere MQ. Any attempt to use a WebSphere MQ Server with a non z/OS based WebSphere MQ installation will fail and will require a WebSphere MQ link in order to work.

A high level overview of a WebSphere MQ server can be seen in Figure 9-32 on page 593. It shows the high level of failure tolerance built in to this connectivity mechanism. An application can use any messaging engine within a bus to connect to the WebSphere MQ Server, so if one fails, another can be used. The WebSphere MQ Server itself can connect to a single MQ queue manager, or one of a shared group to access the queues. When connecting to a shared group, if one queue manager fails, another can be used to access the same queues.

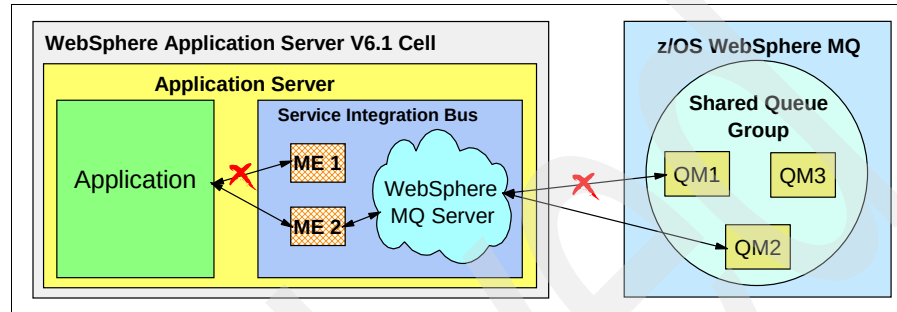


Figure 9-32 Overview of a WebSphere MQ Server

For each queue manager or shared queue group that needs to be accessed, a separate WebSphere MQ Server definition is required to be created. The process of creating a server definition allows the connection details and any security information to be defined for the target queue manager or shared group.

The defined server is added to a bus as a bus member. Queue destinations can then be created for the server definition and the queue points assigned to individual MQ queues. The destinations can be internally or externally mediated (MQ link does not support this). Where this is external, a separate process to the bus is used, but does the same job.

To the WebSphere MQ Server, the MQ queue manager or shared queue group is regarded as a mechanism to queue messages for the bus. The WebSphere MQ Server is regarded by the WebSphere MQ network as just another MQ client attaching to the queue manager or shared queue group.

One major difference between WebSphere MQ Server and WebSphere MQ link is that messages are not stored within the messaging engine with WebSphere MQ Server. Messaging applications directly send and receive messages from the WebSphere MQ queues. This is the reason that MQ server is tolerant of a message engine failure. The message engines are stateless in this regard.

This allows message beans to be configured to immediately process messages as they arrive on an MQ queue. Similarly, any bus mediations take place immediately upon a message appearing on an MQ queue.

In other respects, the use of the WebSphere MQ Server is similar to that of a WebSphere MQ link, and we recommend that you consult the IBM Information Center and “Addressing destinations across the WebSphere MQ link” on page 587 for further information about the use of WebSphere MQ destinations.

9.3 High availability and workload management

Note: This section introduces you to the high availability and workload management capabilities when using the bus. Before configuring your system, consult the following:

- ▶ *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392.
- ▶ *WebSphere Application Server V6: High Availability Solutions*, REDP-3971

High availability and workload management can be achieved using clusters as bus members. It is worth noting, however, that messaging engines do not follow the same clustering model that J2EE applications do in clusters.

9.3.1 Cluster bus members for high availability

When you add a cluster to a bus, a single messaging engine is created. The messaging engine is active on only one server within the cluster. In the event of an application server or messaging engine failure, the messaging engine becomes active on another server in the cluster if one is available.

By default, the messaging engine starts on the first available server in a cluster. If you want to ensure that the messaging engine runs on a particular server, for example, if you have one primary server and one backup server, or if you want the messaging engine to only run on a small group of servers within the cluster, then you must specifically configure this. See 9.8.10, “Setting up preferred servers” on page 643 for details on configuring preferred servers.

9.3.2 Cluster bus members for workload management

Because a single messaging engine for the cluster is active, there is no workload management by default. To achieve greater throughput of messages, it is beneficial to spread messaging load across multiple servers and, optionally, across multiple hosts. You can achieve this, while maintaining a simple destination model, by creating additional messaging engines for the cluster, each of which has a preference to run on a separate server in the cluster.

You can configure these messaging engines with a preference to run on particular servers within the cluster. This enables a messaging engine to run in every server in the cluster, thus providing every application in the cluster with a messaging engine for local access to the bus. Local access to the bus is always better for messaging performance, especially in the case of queues where the queue is assigned to the bus member from which it is being accessed.

When a queue is assigned to a cluster bus member, the queue will be partitioned across all messaging engines in the cluster.

9.3.3 Partitioned queues

A queue is partitioned automatically for you when a queue destination is assigned to a cluster bus member. Every messaging engine within the cluster owns a partition of that queue and is responsible for managing messages assigned to the partition. Every message sent to the queue is assigned to exactly one of the partitions.

Local partitions

When a JMS client attempting to access a partitioned queue is connected to a messaging engine hosting one of those partitions (a messaging engine in the cluster), then the client is able to access only that local partition of the queue for both consuming and producing messages.

Note: The only instance where messages are not sent to the local partition is when that local partition is full and other partitions of the queue are not. In this case, messages are routed to an available remote partition.

Clients attempt to consume only from the local partition, even if there are no messages on the local partition and there are messages available on other partitions.

Remote partitions

If the JMS client connects to a messaging engine not hosting a destination partition, a messaging engine in the same bus but not in the cluster, then each client-created consumer connects to one remote partition to consume messages. Each session created is workload managed with respect to which remote partition it connects for consuming messages.

Messages sent to a remote partitioned destination are workload-managed across the individual partitions on an individual message basis, regardless of the session.

Important: Cluster bus members and partitioned queues alone do not give better message throughput. The applications producing and consuming the messages must be configured to use the bus.

- ▶ Message producers must be configured to ensure that their messages will be workload-managed onto the different partitions of a partitioned queue. The following are examples of workload management:
 - Message producers, JMS clients, connect directly to the cluster. This has some restrictions in Version 6.0. See 9.3.4, “JMS clients connecting into a cluster of messaging engines” on page 596. We anticipate removing these restrictions in the near future with a Fix Pack.
 - Message producers connect to messaging engines that are not part of the cluster. This requires servers outside of the cluster to be available and added to the bus, and for the message producers to make their JMS connections to those messaging engines. Once a messaging engine outside of the cluster accepts a message, the engine becomes responsible for routing the message through the bus to a queue point for the destination. Workload management selects a particular queue point of the partitioned destination so messages are spread evenly across all partitions of the queue.
 - An EJB or servlet in a cluster produces messages. Because the calls to the EJB or servlet are workload-managed across the cluster, and assuming that messages are produced to a local queue partition, it follows that the messages produced will be workload managed across the partitions of the queue.
- ▶ Message consumers must be configured to connect to each partition of a partitioned queue to consume messages. If any partitions do not have consumers, then the messages sent to that partition might never be consumed.

The simplest and recommended way of configuring consumers to every partition of a partitioned queue is by installing a message-driven bean on the cluster.

9.3.4 JMS clients connecting into a cluster of messaging engines

JMS clients outside of a cluster can connect directly into a workload-managed cluster of messaging engines. In this case, *workload-managed* means the cluster is a bus member and one messaging engine has been added for every server in the cluster. Each messaging engine has been configured to prefer a different server in the cluster. JMS clients connect to the messaging engines using the connection rules described in 8.7, “Connecting to a service integration bus” on page 520.

In this scenario, there is an undesirable side effect of the rules when the servers in the cluster are used as the provider endpoints for the connection factory. Consider the following example:

A JMS client connects into a cluster of servers A, B, and C. The connection factory is configured with provider endpoints of A, B, and C. This allows the client to bootstrap to any of the three servers in the cluster. Following the connection rules, the connection factory bootstraps to the first server in the provider endpoints list, A. Server A has a local messaging engine; therefore, the messaging engine on Server A is chosen as the preferred connection point for the client.

Because the connection always tries the first entry in the provider endpoints list first, every client connecting directly into the cluster connects to the messaging engine in server A. All messages produced for a destination partitioned across the cluster are assigned to the partition of the destination associated with the messaging engine. This is obviously not very good for workload management of messages. There are two methods that can overcome this:

- ▶ Enable a SIB service on a server outside of the cluster. Configure the provider endpoints on the connection factory to point to this SIB service. If there is no messaging engine local to this SIB service, then the client connections will be workload-managed around all of the messaging engines in the bus.

If you only have messaging engines in the cluster, no further configuration is required. If there are other non-cluster bus members, and you only want the clients to connect directly to the messaging engines in the cluster, then you must configure a target group on your connection factory. See “Target groups” on page 531.

- ▶ Provide different clients with differently configured connection factories, each of which has a different provider endpoint in the first position in the list.

9.3.5 Preferred servers and core group policies

To configure a messaging engine to prefer a server or group of servers, you must configure a core group policy. A core group policy is used to identify server components, and define how they will behave within a cell or cluster. This section discusses these components.

Policy type

For messaging engines, use a policy type of One of N. This means that, while the messaging engine can be defined on every server in the cluster, WebSphere’s HA Manager ensures that it is only active on one of the servers in the group, and will always be active on one of the servers, if one is available.

Match criteria

The match criteria of a core group policy enables the HA Manager to decide what server components match the policy and so should be managed according to the policy. There are two match criteria that you must use to match a messaging engine:

- ▶ `type=WSAF_SIB`
This criterion matches any messaging engine.
- ▶ `WSAF_SIB_MESSAGING_ENGINE=<messaging_engine_name>`
This criterion matches the messaging engine of the name provided.

Preferred servers

The preferred servers defined in a policy allow you to list a group of servers on which the messaging engine will prefer to run. The higher up in the list of preferred servers a particular server is, the more preferred it is. For a messaging engine that is part of a cluster bus member, select only preferred servers that are part of the cluster. The messaging engines are defined only in the cluster and cannot be run on any servers outside of the cluster.

Fail back and preferred servers only

These two options have a large effect on how a particular policy will make a messaging engine behave in a cluster.

If you select **Fail back**, when a more preferred server becomes available, then the messaging engine will be deactivated where it currently runs and activated on the more preferred server. Enabling fail back ensures that a messaging engine always runs on the most preferred server that is available. This is usually desirable, as there should be a good reason for configuring a preferred server in the first place. If you do not enable fail back, then once a messaging engine has started it will not move to a more preferred server if one becomes available.

If you select **Preferred servers only**, then the messaging engine will only be allowed to be active on servers in the policy's preferred servers list. If you do not select **Preferred servers only**, all servers in the cluster that are not in the list will be able to have the messaging engine active on them, but they will be selected only if none of the preferred servers are available.

Be very careful when selecting preferred servers only because it is possible to reduce or remove the high availability of a messaging engine and of the queue partitions that the messaging engine owns.

If none of the preferred servers are available, then the messaging engine will not be active anywhere. This means any queue partitions owned by that messaging engine will also be unavailable. Any messages currently on those partitions will

be trapped and cannot be consumed until one of the preferred servers has become available and the messaging engine has been activated.

Large clusters

If you have a medium or large cluster of servers (five or more, configured with messaging engines), then we recommend a slightly special configuration of preferred servers.

With a large number of messaging engines defined on a cluster, it would be undesirable to have all of the messaging engines starting up on the first server in the cluster to start. We suggest the following configuration.

Configure each messaging engine with a group of preferred servers consisting of a subset of the cluster with fail back and preferred servers only enabled. The set of preferred servers should be large enough to support your availability requirements by providing sufficient failover capabilities for the messaging engine. For example, you might decide that the messaging engine must be able to run on two or three servers. Configure each messaging engine with a different subset of servers, with each messaging engine having a unique, most-preferred server, as in Figure 9-33 on page 600.

In Figure 9-33 on page 600, the shading indicates the preference order of the servers.

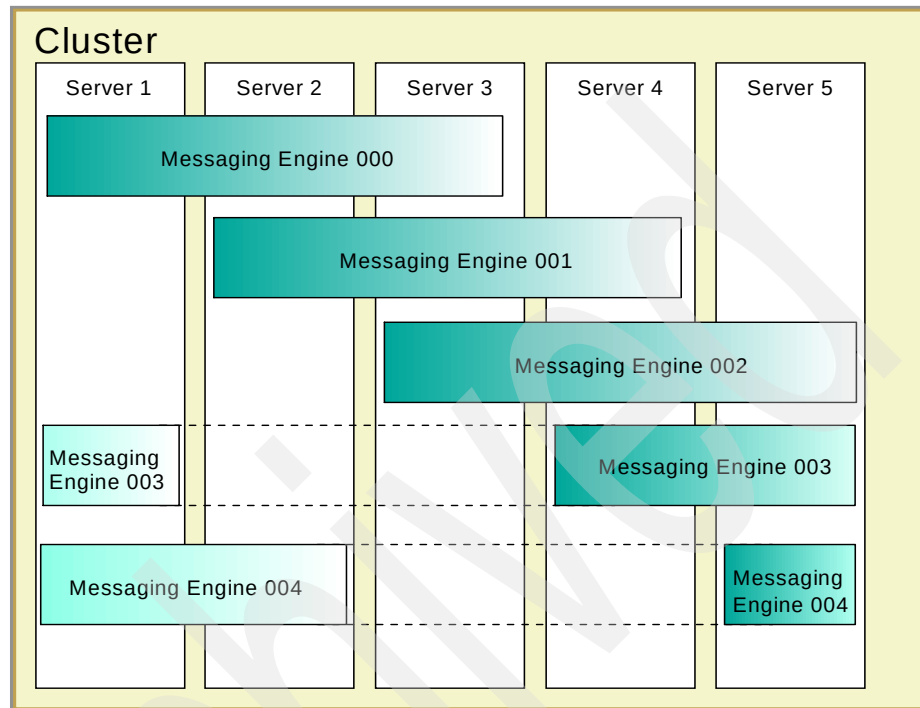


Figure 9-33 Configuring large clusters of messaging engines

9.3.6 Best practices

For the greatest throughput of messages, do the following:

1. Create a cluster bus member with messaging engines running on every server in the cluster.
2. Define the queue or queues being used on the cluster bus member.
3. Ensure that message production to the queue is workload-managed across the cluster:
 - Install an EJB or servlet application on the cluster and have that application produce the messages. Workload management of the client calls to the application workload manages the message production across the cluster.

- Produce messages from clients connected to messaging engines outside of the cluster. The bus can then workload manage the messages across the cluster.
4. Install an MDB application on the cluster to consume the queue messages.

9.4 Service integration bus topologies

This section discusses briefly some messaging topologies, working up from the simplest to more complex configurations.

9.4.1 One server in the cell is a member of one bus

In this topology, there is only one bus. There might be multiple application servers in the cell, but only one is a member of the bus. This is roughly equivalent to the typical V5.x JMS server topology.

The pros of this topology are:

- ▶ It is very simple to set up and manage.
- ▶ It can be expanded later by adding more servers to the bus.

The cons are:

- ▶ Clients running on other application servers in the cell have to connect remotely to the bus rather than connecting locally. This can affect messaging performance.
- ▶ Clients running outside application servers have to connect to the bus member to do messaging. The connection factory you use needs to have provider endpoints configured with the details of the bus member server.

If the SIB service is enabled on other application servers in the cell, then connection factories can be configured with provider endpoints that point to a list of bootstrap servers. See 8.7, “Connecting to a service integration bus” on page 520 for more information about using a bootstrap server and defining a list of provider endpoints.

In either case, all messaging connections go to the bus member server and might affect messaging performance.

- ▶ Message consumers might not be on the same server as the queue points they are consuming from. This could have a performance impact.
- ▶ This topology cannot be upgraded easily to support high availability or workload management. High availability and workload management require clustering application servers. You can create a new cluster and include the bus member as the first application server in the cluster. However, this will not

automatically give you the messaging high availability features that are normally associated with adding a cluster as a bus member.

- Using the bus member server as the template for a cluster server is *not* equivalent to adding a cluster to the bus. No bus information is copied as part of the template process. The SIB service will be enabled on the new cluster server as a server property, not part of any particular bus.
- Using the bus member as the first server in cluster server is *not* equivalent to adding a cluster to the bus. Only the original server is part of the bus.

It is possible to add a cluster to the bus, delete all of the queues you want to be highly available or workload-managed, and recreate queues of the same name that have their queue points located on the new cluster bus member. Any messages on the queues are lost when they are deleted.

9.4.2 Every server in the cell is a member of the same bus

In this topology, there are multiple application servers, but no clusters. There is one bus and each application server is a member of that bus.

The pros of this topology are:

- ▶ Clients in application servers can connect locally to the bus, improving performance. If only some servers in the cell are members of the bus, then install any messaging applications on those servers.
- ▶ Clients running outside application servers can connect to any cell server to perform messaging, providing some degree of high availability for those clients.
- ▶ It is possible to have queue points in the same servers as applications that consume messages from them, improving performance.

The cons are:

- ▶ This topology is not easily upgradeable to support high availability or workload management. See the list of cons in 9.4.1, “One server in the cell is a member of one bus” on page 601.

9.4.3 A single cluster bus member and one messaging engine

This scenario assumes that all the application servers in the cell belong to one cluster and that cluster is a member of the bus.

The benefit of this scenario is:

- ▶ The messaging engine is highly available. If the messaging engine or the server on which it runs fails, then the messaging engine starts up on another

available server in the cluster. See “High availability and workload management” on page 594 for more details.

The drawback of this scenario is:

- ▶ If you want to ensure that the messaging engine runs on one preferred server, for example, if you have one primary server and one backup server, then you must specifically configure this. See “Setting up preferred servers” on page 643.

Note: Be aware that some configurations of preferred servers for a messaging engine can make that messaging engine not highly available.

If preferred servers are set up for the messaging engine with the preferred servers only option, then it is possible, if none of the preferred servers are available, that the messaging engine will not have another server on which to start even if other servers are available in the cluster.

9.4.4 A cluster bus member with multiple messaging engines

This scenario assumes that all the application servers in the cell belong to one cluster. Multiple messaging engines have been defined for the cluster.

The pros of this topology are:

- ▶ The messaging engines in the cluster are highly available.
- ▶ The cluster bus member is capable of messaging workload management. A queue point assigned to the cluster bus member is partitioned onto every messaging engine in the cluster and messages delivered into the cluster are distributed between the partitions.

The drawback is that there are some restrictions on the workload management of client connections directly into a cluster. See 9.3.4, “JMS clients connecting into a cluster of messaging engines” on page 596 for details.

9.4.5 Mixture of cluster and server bus members

The cell has some application server clusters and other non-clustered servers. Both non-clustered servers and server clusters have been added as bus members. Complex configurations such as these can be completely tailored to best suit your application and server topologies.

The benefits of this topology are:

- ▶ Cluster bus members can be configured with partitioned destinations to support workload-managed, message-consuming applications, such as message-driven beans.
- ▶ Cluster bus members can be used to make system-critical destinations highly available.
- ▶ To overcome the workload management restrictions of clients connecting to a cluster, clients outside the cell can connect to server bus members. Clients can then put messages to destinations with partitioned queue points. Messages are workload-managed between the partitions.
- ▶ To overcome the workload management restrictions of clients connecting to a cluster (see 9.3.4, “JMS clients connecting into a cluster of messaging engines” on page 596), clients outside the cell can connect to server bus members outside of the cluster. Clients can then put messages to partitioned destinations and the messages will be workload-managed across the partitions.
- ▶ Clients bootstrapping to servers (with a SIB service) outside the cluster can get workload management of their connections to the messaging engines within the cluster bus member.

The drawback is that these more complex messaging topologies take a little more planning and configuration than simpler topologies.

9.4.6 Multiple buses in a cell

It is possible to have many buses within a cell. This topology can be desirable under in the following situations:

- ▶ Separation of concerns
Applications that do not need to share messages can be isolated from each other by using their own bus.
- ▶ Test configuration
A test configuration with identical destination names can be created on a separate bus that is not used by the production system. Changing the name of the bus in the connection factories can then redirect the test application to the production bus without changing any other configuration.

9.5 Service integration bus and message-driven beans

Message-driven beans (MDBs) attached to destinations in the bus are attached by means of the SIB JMS Resource Adapter, an activation specification, and a JMS destination. The resource adapter is responsible for connecting to the bus and delivering messages to the MDB.

Note: For performance reasons, we recommend that MDBs are always installed on a server that has an active local messaging engine and a queue point on that local messaging engine.

9.5.1 Message-driven beans connecting to the bus

The resource adapter always attempts to connect a message-driven bean to a messaging engine in the same server, if one is defined there. If there is no messaging engine in the same server, then a messaging engine is selected from the bus using the standard connection selection process, see 8.7, “Connecting to a service integration bus” on page 520.

There are three scenarios where an MDB will start but not connect to the destination for which it is configured to listen. The resource adapter will allow the MDB application to start under these circumstances and will attempt to connect the MDB to its configured destination as soon as possible.

Local messaging engine defined but unavailable

If a messaging engine is defined locally, but is unavailable when the MDB application starts, the MDB application starts successfully and the resource adapter connects it to the messaging engine when it activates. Situations when this happens include:

- ▶ If the messaging engine has not started by the time, the MDB application is started.
- ▶ The MDB is installed on a cluster bus member that has been configured for high availability, and is on a server other than the one with the active messaging engine.

When an MDB application is started, but the locally defined messaging engine is unavailable, the warning message in Example 9-1 will appear in SystemOut.log.

Example 9-1 Message: local messaging engine not available

CWSIV0759W: During activation of a message-driven bean, no suitable active messaging engines were found in the local server on the bus MyBus

When the messaging engine activates, the message in Example 9-2 is displayed when the MDB is connected to its destination.

Example 9-2 MDB connected to messaging engine

CWSIV0764I: A consumer has been created for a message-driven bean against destination MyQueue on bus MyBus following the activation of messaging engine cluster1.000-MyBus.

Note: Messaging engines are frequently the last component of an application server to complete their startup, often even after the open for e-business message is issued for the server. As a result, it is not unusual for MDB applications to cause the above warning message.

Remote destination unavailable

If there is an active locally defined messaging engine, but the MDB is configured to listen to a queue currently unavailable (for example, if the messaging engine that hosts the queue point is not active), then the warning message in Example 9-3 is displayed.

Example 9-3 Message: remote destination unavailable

CWSIV0769W: The creation of a consumer for remote destination MyQueue on bus MyBus for endpoint activation ...<section removed>... failed with exception javax.resource.ResourceException: CWSIP0517E: Cannot attach to queue message point for destination MyQueue.

The resource adapter tries to connect the MDB to the configured destination every 30 seconds until it succeeds. Each failure to connect results in the message shown in Example 9-3.

Remote messaging engine unavailable

If there is no locally defined messaging engine, then a messaging engine is selected from the bus. If there are no currently available messaging engines in the bus, then the resource adapter allows the MDB application to start anyway and attempt to connect the MDB to a messaging engine every 30 seconds. The message in Example 9-4 appears on the first failed attempt to connect to a messaging engine. Subsequent failures are silent.

Example 9-4 Message: remote messaging engine unavailable

CWSIV0775W: The creation of a connection for destination MyQueue on bus MyBus for endpoint activation ...<section removed>... failed with exception com.ibm.websphere.sib.exception.SIResourceException: CWSIT0019E: No suitable messaging engine is available in bus MyBus.

No messages are delivered to the MDB until the resource adapter has been able to start a connection to an active messaging engine. The message in Example 9-5 is displayed with a connection is made.

Example 9-5 Message: connection made to remote messaging engine

```
CWSIV0777I: A connection to remote messaging engine myNode.server1-MyBus for
destination MyQueue on bus MyBus for endpoint activation ...<section
removed>... is successfully created.
```

9.5.2 MDBs and clusters

The behavior of message-driven beans installed on clusters that use the bus is directly related to the bus configuration.

Clusters that are not part of a bus

When an MDB is installed on a cluster that is not part of a bus, the MDBs on each server connect independently to the bus to consume messages.

Note: You should not configure an MDB on a cluster with no local messaging engine to listen to a partitioned queue in another cluster. There is no guarantee that every partition of the queue in the other cluster will have at least one MDB listening to it. This could lead to a partition without any consumers.

Clusters configured for highly available messaging

When a cluster is configured for highly available messaging, a messaging engine is active on one of the servers in the cluster. An MDB application installed on that cluster will start on all servers in the cluster, but only the MDB on the server with the active messaging engine will receive messages. Should the active messaging engine fail, or the server on which it is active fails or is stopped, then the messaging engine will start on another server in the cluster. The MDB on that server will be connected to the messaging engine and start receiving messages.

In this scenario, the bus has been configured to have one active messaging engine in the cluster, and, effectively, the MDB mirrors that configuration.

Clusters configured for messaging workload management

When a cluster is configured for messaging workload management, a messaging engine will most likely be active on each server in the cluster.

For a MDB installed on the cluster and listening to a topic with a non-durable subscription, each message on the topic will be received once on each server with an active messaging engine. If more than one messaging engine is active on a server, a publish topic message will still be received only once by the MDB on that server.

If the MDB installed on the cluster is listening to a topic with a shared, durable subscription, then one MDB in the cluster receives each message published on the topic only once.

If the MDB installed on the cluster is listening to a queue partitioned on the cluster, then the MDB is attached to each partition active on the server. Should more than one messaging engine be active on a server, then the MDB will receive messages from each messaging engine's partition of the queue.

For a MDB installed on the cluster and listening to a queue with its queue point on a messaging engine outside of the cluster, the MDB on each server is attached to the queue. An MDB on a server with more than one active messaging engine will not receive a greater proportion of the messages than an MDB on a server with only a single active messaging engine.

9.6 Service integration bus security

New in V6.1: The bus security in WebSphere Application Server V6.1 has been enhanced. Bus security is enabled independently of application and administrative security, though administrative security must be enabled to enable bus security. New features include the requirement of trusted transport chains, client authentication, and an authorization policy that requires users and groups to be granted access to the bus and its resources.

Bus security can be turned on or off at the time of bus creation, or afterward. For the bus security to be activated, administrative security must be enabled.

Every bus has an optional inter-engine authentication alias that can be specified. If this property is left unset, then it will default to **none** and be ignored. However, if an alias is specified and security enabled, then the ID will be checked when each messaging engine starts communicating with other messaging engines in the bus. This provides additional security to prevent hackers pretending to be another messaging engine in the bus.

A list of permitted transport chains can be defined that may be used to access a secured bus. There are three modes: allow all defined transport chains, allow only SSL enabled transport chains, and allow only those transport chains in a list defined by the administrator.

The mediations authentication alias is used to authorize any mediation processes trying to access the secured bus.

Each secured bus now has a *bus connector role*. Any external client that needs to access the bus needs to be added to the bus connector role. By default, if the client has not been added, they will be denied access, even if they have valid credentials. However, there are options to allow only servers that are members of the bus to connect to the bus, all authenticated users to connect to the bus, or everyone (including unauthenticated users) to connect to the bus.

When security is enabled on WebSphere Application Server, certain steps must be taken for JMS applications using the bus to authenticate themselves to the bus, allowing them to continue to use the messaging resources.

- ▶ All JMS connection factory connections must be authenticated. This can be done in two ways:
 - The connection factory can have a valid authentication alias defined on it.
 - The JMS application can pass a valid user name and password on the call to `ConnectionFactory.createConnection()`. An ID passed in this way overrides any ID specified in an authentication alias on the connection factory.
- ▶ All activation specifications must have a valid authentication alias defined on them.

Note: If a connection factory is looked up in the server JNDI from outside of the server environment (for example, from the client container), any authentication alias defined on the connection factory will be unavailable. This prevents unauthorized use of an authenticated connection factory.

JMS clients outside of the server can provide a user name and password on the call to create a connection. If the client is a J2EE client application running in the WebSphere application client environment, it is possible to define an authenticated connection factory resource in the .ear file.

Details on WebSphere security can be found in *WebSphere Application Server V6.1 Security Handbook*, SG24-6316.

9.7 Problem determination

The following information is presented to help you become familiar with successful messaging engine startup, and some common problems.

No problems

Example 9-6 shows an example of what you can expect to see in systemOut.log on server start up for a messaging engine that starts successfully.

Example 9-6 Successful messaging engine start

```
...
CWSID0016I: Messaging engine Node1.server1-ITS0Bus is in state Joined.
...
CWSID0016I: Messaging engine Node1.server1-ITS0Bus is in state Starting.
...
CWSID0016I: Messaging engine Node1.server1-ITS0Bus is in state Started.
...
```

Note: When you start a server that is part of a cluster bus member, then the messaging engine will not always be started. Only one server in the cluster will have a specific messaging engine activated on it and this messaging engine might already be started.

If this is the case, then you will see the messaging engine in the state Joined, but not Starting or Started. This is perfectly normal and means that the messaging engine is in a stand-by state, waiting to be activated should the currently active instance of the messaging engine become unavailable.

When you have more than one messaging engine in a bus, you will also see the messaging engines communicate with each other. Every messaging engine in the bus connects to every other messaging engine in the bus, as shown in Example 9-7.

Example 9-7 Messaging engine connections

```
...
CWSIT0028I: The connection for messaging engine Node1.server1-ITS0Bus in bus
ITS0Bus to messaging engine Node2.server2-ITS0Bus started.
...
CWSIP0382I: messaging engine B68588EF698F4527 responded to subscription
request, Publish Subscribe topology now consistent.
...
```

CWSIS1535E: Messaging engine's unique ID does not match...

If you see the error shown in Example 9-8, the database that the messaging engine points to contains the unique ID of a different messaging engine. The most likely cause of this is when you create a bus, add a server to that bus using the default Cloudscape database and start the server. Later, you delete and recreate a bus of the same name. The newly created messaging engine will use a default data source that points to the same database used by the old messaging engine, and this database will contain the ID of the old messaging engine.

This error can also be caused by configuring any messaging engine with the same message store as another messaging engine.

Example 9-8 Messaging engine unique ID does not match when using a data store

```
CWSIS9999E: Attempting to obtain an exclusive lock on the data store.  
CWSIS1535E: The messaging engine's unique id does not match that found in the  
data store. ME_UUID=1C80283E64EAB2CA, ME_UUID(DB)=B1C40F1182B0A045  
WSIS1519E: Messaging engine Node1.server1-ITS0Bus cannot obtain the lock on its  
data store, which ensures it has exclusive access to the data.  
CWSID0027I: Messaging engine Node1.server1-ITS0Bus cannot be restarted because  
a serious error has been reported.  
CWSID0016I: Messaging engine Node1.server1-ITS0Bus is in state Stopped.
```

For a data store, the simplest solution is to drop the tables in the database, or delete and recreate the database and then restart the server. Another solution is to change the messaging engine's data store by changing the schema, user, and database configured for the messaging engine. For a file store, delete the files, or the directory paths. See "Adding the bus member" on page 631 for more details

CWSIT0019E: No suitable Messaging Engine...

This exception shown in Example 9-9 can be thrown to a JMS client on a `createConnection` call. Causes of this exception include:

- ▶ The JMS connection factory cannot contact an SIB service, for out of cell JMS clients only. Check that the provider endpoints listed in the connection factory match the host and port for the SIB services on the servers. Ensure that the SIB services are enabled and the servers are started.
- ▶ The bus name defined in the JMS connection factory does not match the name of a bus defined in WebSphere.
- ▶ No messaging engines on the named bus are active.

```
javax.jms.JMSEException: CWSIA0241E: An exception was received during the call  
to the method JmsManagedConnectionFactoryImpl.createConnection:  
com.ibm.websphere.sib.exception.SIResourceException: CWSIT0019E: No suitable  
messaging engine is available in bus ITS0Bus.
```

9.8 Configuration and management

This section discusses how to set up and configure a bus using the administrative console.

The following specific activities are described:

- ▶ SIB service configuration
- ▶ Creating a bus
- ▶ Configuring bus properties
- ▶ Enabling bus security
- ▶ Adding a bus member
- ▶ Creating a queue destination
- ▶ Creating a topic space destination
- ▶ Creating an alias destination
- ▶ Adding messaging engines to a cluster
- ▶ Setting up preferred servers
- ▶ Setting up a foreign bus link to a service integration bus
- ▶ Setting up a foreign bus link to an MQ queue manager
- ▶ Creating a foreign destination

When configuring the bus for use with the default messaging provider, the minimum tasks that apply are:

1. Creation and configuration of a bus (optionally including security)
2. The addition of at least one bus member
3. The definition of destinations of one variety or another

When configuring the bus to communicate with WebSphere MQ, you can set up a WebSphere MQ link through a foreign bus, or, where MQ is on a z/OS platform, a WebSphere MQ Server. The minimum tasks for both appear below:

To use a WebSphere MQ link:

1. Create and configure a bus.
2. Add at least one bus member and any required destinations.
3. Set up a foreign bus link to an MQ queue manager.

4. Add alias destinations that points to the MQ queues via the MQ link foreign bus.

To use a WebSphere MQ Server

1. Create and configure a bus.
2. Create a WebSphere MQ Server definition and add it to the bus as a member.
3. Create one or more queue destinations that correspond to the MQ queues.
4. Add at least one other bus member and any other destinations.

Note: In the following instructions, we frequently suggest saving the changes. You do not have to do this and can make several changes before saving.

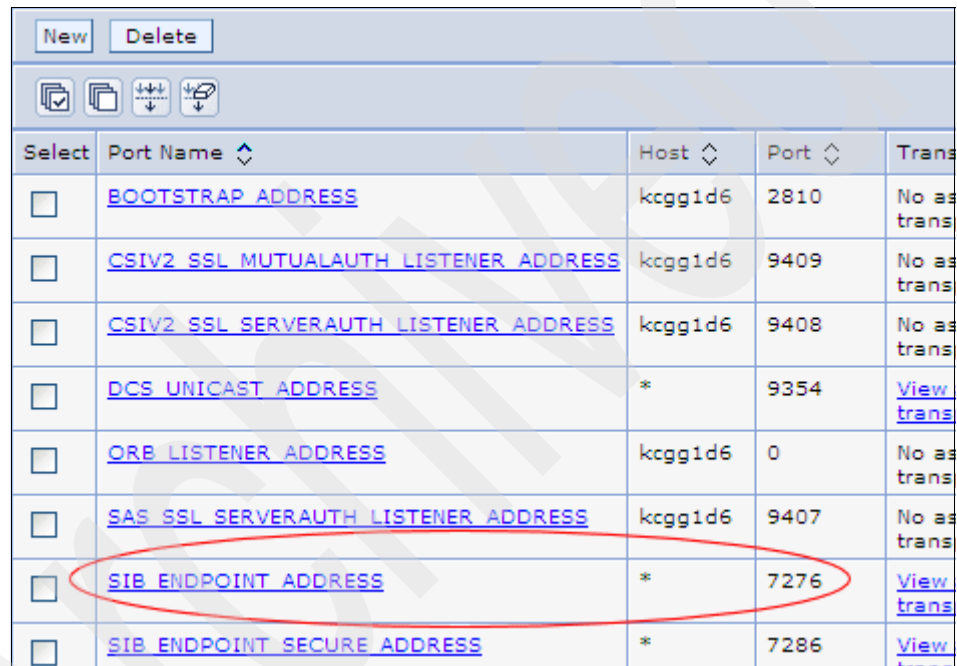
9.8.1 SIB service configuration

SIB service is an application server service enabling the server for service integration activities. When a server is added to a bus, it automatically has its SIB service enabled. Having the SIB service allows an application server to have active messaging engines and to be used as a provider endpoint for default messaging connection factories. The port on which the SIB service listens can be looked up on the servers configuration window.

1. Select **Servers** → **Application Servers**.
2. Select the application server.

- Under **Communications**, expand the **Ports** heading.
SIB_ENDPOINT_ADDRESS is the port used by SIB Service for that server.
See Figure 9-34.

Note: SIB service listens on a number of ports, not just the port for SIB_ENDPOINT_ADDRESS. SIB_ENDPOINT_SECURE_ADDRESS is also available, and is used for secure communications. Tunnelled and tunnelled secure endpoints are also provided: jfap/http/tcp and jfap/http/ssl/tcp. Refer to the Information Center for more details.



Select	Port Name	Host	Port	Trans
☐	BOOTSTRAP ADDRESS	kcgg1d6	2810	No as trans
☐	CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS	kcgg1d6	9409	No as trans
☐	CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS	kcgg1d6	9408	No as trans
☐	DCS_UNICAST_ADDRESS	*	9354	View. trans
☐	ORB_LISTENER_ADDRESS	kcgg1d6	0	No as trans
☐	SAS_SSL_SERVERAUTH_LISTENER_ADDRESS	kcgg1d6	9407	No as trans
☐	SIB_ENDPOINT_ADDRESS	*	7276	View. trans
☐	SIB_ENDPOINT_SECURE_ADDRESS	*	7286	View. trans

Figure 9-34 Port numbers used by a server

The settings for the SIB service of an application server can be found on the administrative console:

- Select **Servers** → **Application Servers**.
- Select the application server.
- Under Server messaging, select **SIB service**. See Figure 9-35 on page 615.

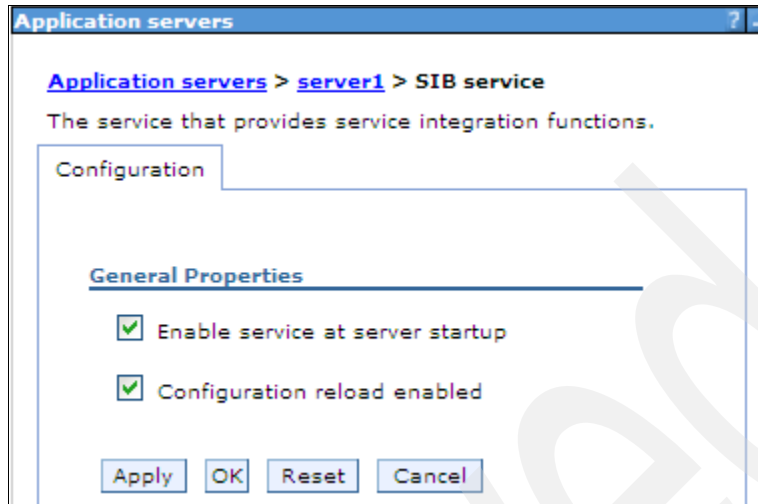


Figure 9-35 SIB Service window

The window for SIB service has two options.

- Enable service at server startup

This option is not enabled on a server by default. However, it is automatically enabled if you add a server to a bus. If you disable the SIB service, then any messaging engines defined on the server will not be started.

- Configuration reload enabled

This option allows the SIB service to activate dynamically certain changes to a bus configuration during run time. Creation, deletion, or modification of a destination or mediation takes effect almost immediately on a running system. If a new destination is created, it becomes available for use without having to restart servers or messaging engines. Some configuration changes do require the affected server or messaging engine to be restarted before the changes become effective, such as the creation of a new bus, messaging engine, foreign bus link, or MQ link.

A matching flag must also be enabled on each bus on which you want to enable configuration reload. This flag is enabled by default on every bus, but can be disabled if you want. To modify the flag either way, do the following:

- i. Select **Service integration** → **Buses**.
- ii. Select a bus.
- iii. Modify the **Configuration reload enabled** flag as appropriate.

- iv. Save the changes.

9.8.2 Creating a bus

No buses are defined by default. To create a bus, do the following:

1. Select **Service integration** → **Buses**.
2. Click **New**. See Figure 9-36.

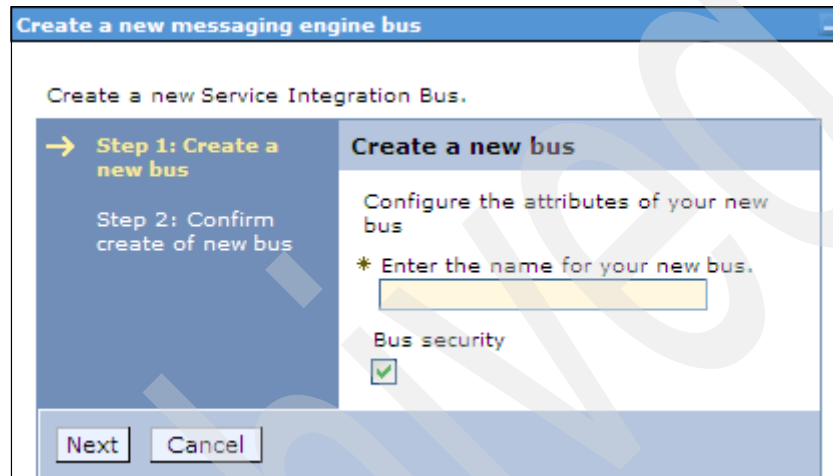


Figure 9-36 First window of the bus creation wizard

This window is the only opportunity to provide the name of the new bus. You cannot change the name of a bus after it has been created, but you can create any number of buses in a cell and delete old ones. Make your bus name unique and meaningful. It is a required field.

The Bus security check box allows security to be enabled on the bus. If administrative security is enabled, then the check box is selected by default. If security is selected, then bus security is enabled and only SSL enabled transport chains are allowed. Disabling bus security will allow any transport chain.

3. Click **Next**.
4. Click **Finish** and save your changes.

9.8.3 Configuring bus properties

1. Select **Service integration** → **Buses**.
2. Select the bus that you want to configure. The bus configuration window is displayed. See Figure 9-37.

The screenshot shows a configuration window with two tabs: 'Configuration' and 'Local Topology'. The 'Configuration' tab is active. It is divided into two main sections: 'General Properties' on the left and a list of links on the right.

General Properties:

- Name:** ITSONodeSamplesBus
- UUID:** 3BE978DFB065C35D
- Description:** A text area with up and down arrows.
- Inter-engine transport chain:** A text field.
- ☐ Discard messages
- ☒ Configuration reload enabled
- High message threshold:** 50000 messages

Right-hand links:

- Topology:**
 - Bus members
 - Messaging engines
 - Foreign buses
- Destination resources:**
 - Destinations
 - Mediations
- Services:**
 - Inbound Services
 - Outbound Services
 - WS-Notification services
- Additional Properties:**
 - Custom properties
 - Security
 - Web service gateway instances

At the bottom of the 'General Properties' section are four buttons: 'Apply', 'OK', 'Reset', and 'Cancel'.

Figure 9-37 Bus configuration window

The following properties can be set:

- Description

This field is an optional description for the bus, for administrative purposes.

- Inter-engine transport chain

The transport chain used for communication between messaging engines in this bus. It must correspond to one of the transport chains defined in the Messaging engine inbound transports settings for the server. When you specify the name of a transport chain, that chain must be defined to all servers hosting messaging engines in the bus. Otherwise, some messaging engines might not be able to communicate with their peers in the bus. The default transport chain is InboundBasicMessaging.

- Discard messages

Use this field to specify whether messages on a deleted message point should be retained at a system exception destination or can be discarded.

- Configuration reload enabled

Select this option to enable certain changes to the bus configuration to be applied without requiring the messaging engines to be restarted. If you select this option, make sure the matching flag on the SIB service is also enabled. See 9.8.1, “SIB service configuration” on page 613.

- High message threshold

Enter a threshold above which the messaging system will take action to limit the addition of more messages to a message point. When a messaging engine is created on this bus, the value of this property sets the default high message threshold for the messaging engine.

3. Click **Apply** or **OK** and save your changes.

9.8.4 Enabling bus security

Bus security can be enabled or disabled and further configured in this window. If administrative security is disabled, then bus security cannot be enabled.

1. Select **Service integration** → **Buses**.
2. Select the bus that you want to configure. The bus configuration window is displayed.
3. Select **Security** in the Additional Properties section. The bus security configuration window is displayed. See Figure 9-38 on page 619

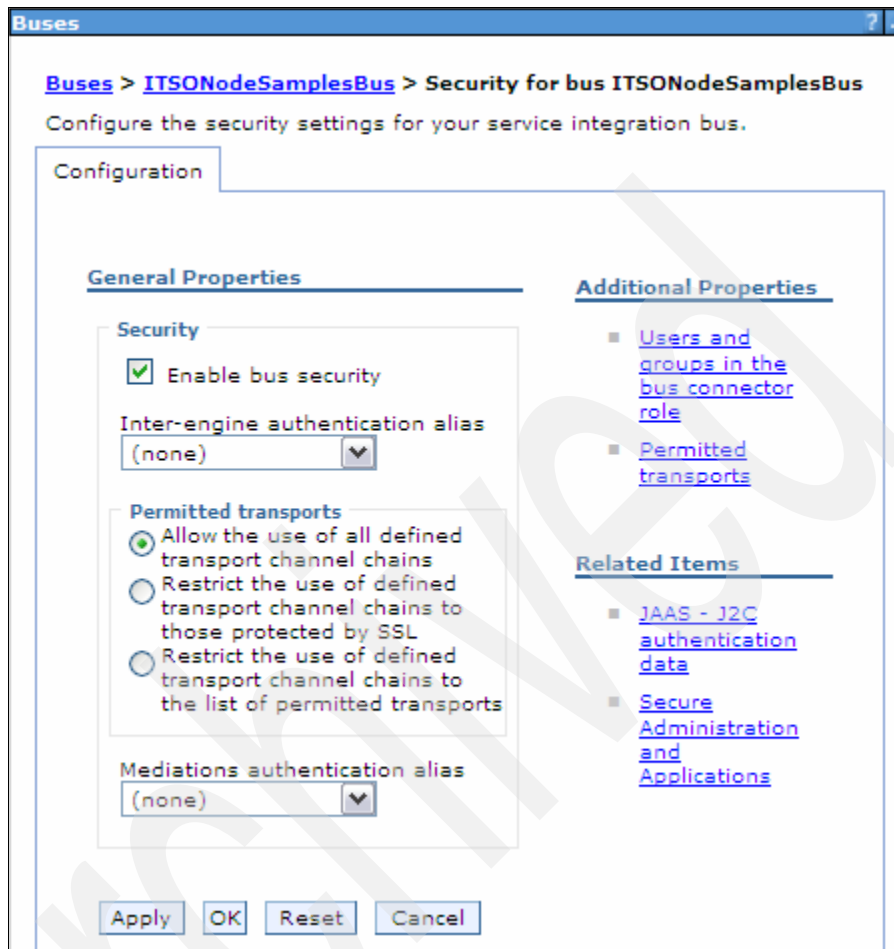


Figure 9-38 Bus security configuration window

The following properties can be set:

- Enable bus security

Select this option to if you want to enable bus Security. If this option is enabled, access to the bus itself and to all destinations must be authorized. Bus security cannot be enabled if administrative security is not also enabled.

- Inter-engine authentication alias

This field contains the name of the authentication alias used to authorize communication between messaging engines on the bus. This field is optional. If a value is specified, and bus security is enabled, incoming

connections to the bus are controlled to prevent unauthorized clients or messaging engines from establishing a connection.

- Permitted transports

There are three policies that may be selected to dictate which message transport chains may be used when bus security is enabled. The first one allows the use of any transport defined to any bus member. The second one allows the use of only those transports that are protected by SSL encryption. The third option restricts allowed transports to those appearing on an administrator maintained list. This is may accessed by selecting **Permitted Transports** in the Additional Properties section.

- Mediations authentication alias

Enter the name of the authentication alias used to authorize mediations to access the bus. This field is optional and will be ignored if no value is set, or bus security is disabled.

4. Click **Apply** or **OK** and save your changes.

Authorizing users or groups to bus security

If bus security is enabled, individual users or groups must be connected to the bus connector role to connect to the bus. Even if the external party is properly authenticated, they will be denied access to the bus if they do not have this role.

To add, remove, or list users in this role, do the following:

1. Select **Service integration** → **Buses**.
2. Select the bus that you want to configure. The bus configuration window is displayed.
3. Select **Security** in the Additional Properties section. The bus security configuration window is displayed.
4. Select **Users and groups in the bus connector role** in the Additional Properties section. Here you may review the current list, add, or remove entries. For this example, we will add an entry.
5. Select **New**. The bus security connector role entry configuration window is displayed. See Figure 9-39 on page 621.

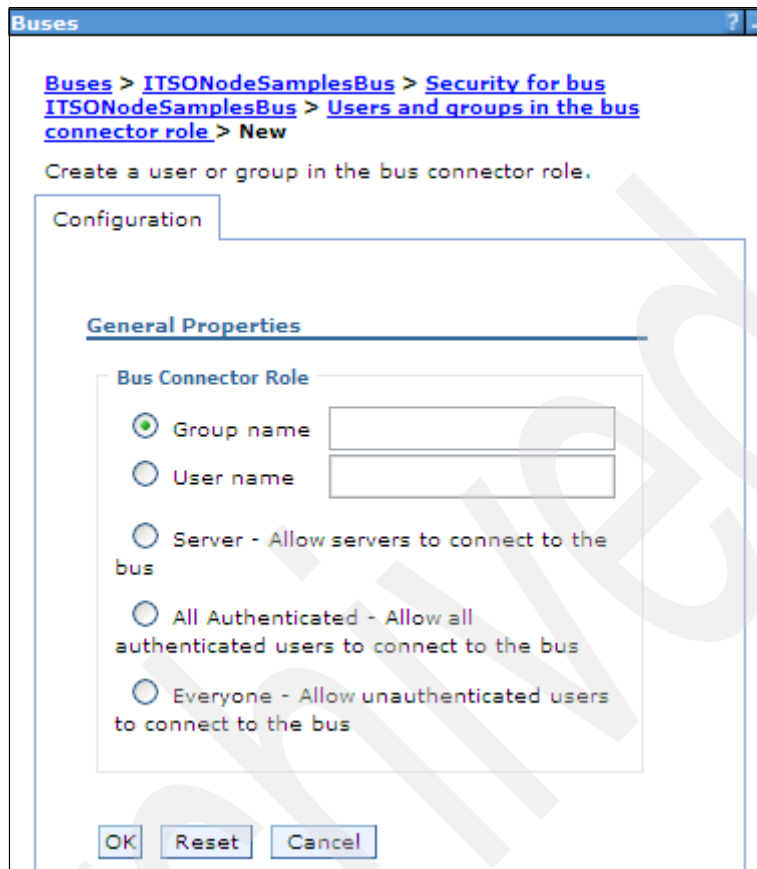


Figure 9-39 Bus security connector role entry configuration window

Only one of the following properties may be set (chosen by the radio button):

- Group name
Gives a specified group the bus connector role.
- User name
Gives a specified user the bus connector role.
- Server

This is the identity of a WebSphere Application Server. It allows Message Driven Beans (MDBs) to connect without specifying an authentication alias

- All Authenticated

This allows all users that have been authenticated to the bus, but not part of the bus connector role, to connect to the bus. This adds the role to the group AllAuthenticated.

- Everyone

This allows all users, authenticated or not, to connect to the bus without being part of the bus connector role. All users are treated as anonymous.

6. Click **OK** and save your changes.

9.8.5 Adding a bus member

A member of a bus can be an application server, a cluster, or a WebSphere MQ Server. For a cluster or application server, a messaging engine is automatically created within the bus. The messaging engine requires a message store for persistent and temporary storage. This message store can be implemented as flat files (file store), or as tables in a database (data store).

A wizard is used to add a member to a bus. Application servers and clusters use the same windows and procedure to be added. WebSphere MQ Servers do not have a messaging engine created, and so do not have to specify a message store. This will be addressed in “Creating and using a WebSphere MQ Server” on page 633.

To add a member to the bus:

1. Select **Service integration** → **Buses**.
2. Select the bus to which you want to add a member.
3. Select **Bus members** in the Additional Properties section.
4. On the Bus members window, click **Add**. See Figure 9-40 on page 623.
5. Select the type of member you wish to add.
6. Select **Next**.

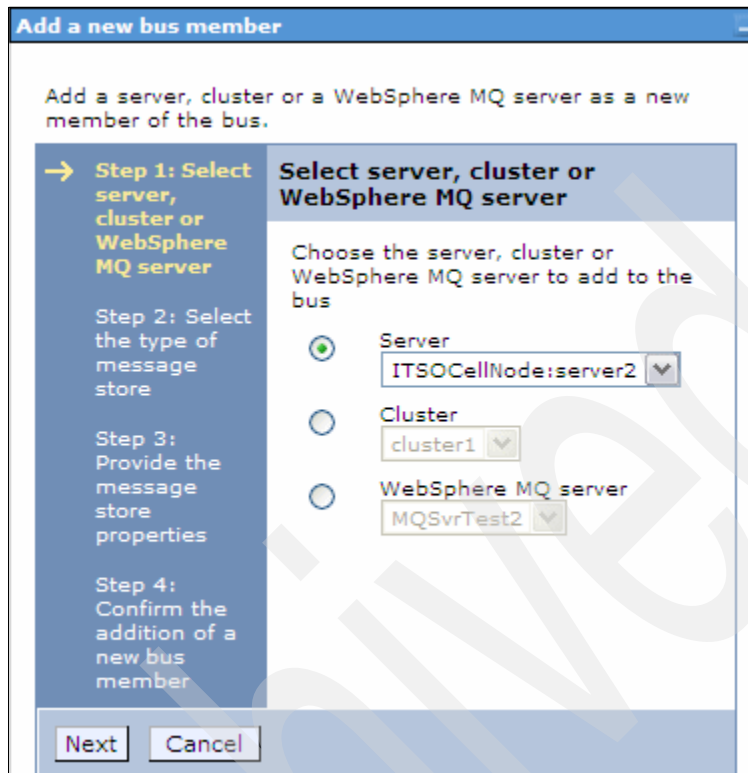


Figure 9-40 First window of Add bus member wizard

Click **Next**.

7. Every messaging engine has a message store associated with it. This window allows you to select the type. See Figure 9-41.

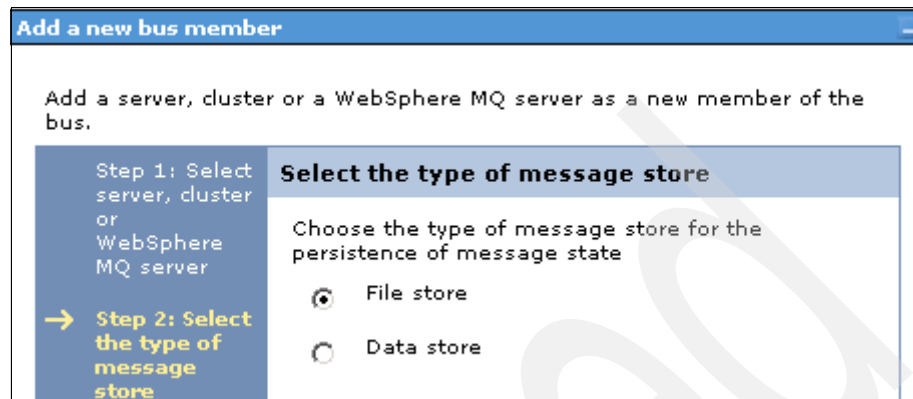


Figure 9-41 Select the message store type

What you select here will determine how you proceed through the wizard.

Adding a server or cluster to a bus using a file store

If you want to use a file store with a cluster or application server, do the following from the second window of the wizard that adds bus members.

1. Select **File Store**.

2. Select **Next**. The file store configuration window will appear. See Figure 9-42 on page 625.

Add a new bus member

Add a server, cluster or a WebSphere MQ server as a new member of the bus.

Step 1: Select server, cluster or WebSphere MQ server

Step 2: Select the type of message store

→ **Step 3: Provide the message store properties**

Step 4: Confirm the addition of a new bus member

Provide the message store properties

Provide properties for the file store

Log

* Log size: 100 MB

☒ Default log directory path

☐ Log directory path:

Store

☒ Same settings for permanent and temporary stores

Permanent and temporary stores

* Minimum permanent store size: 200 MB

☐ Unlimited permanent store size

* Maximum permanent store size: 500 MB

☒ Default permanent store directory path

☐ Permanent store directory path:

Previous Next Cancel

Figure 9-42 File store configuration window

The following properties can be set:

- Log size

The size of the log file. The minimum value is 10 MB. The default is 100 MB. This file does not grow and so does not have minimum and maximum file sizes.

- Default log directory path

Select this radio button to accept the default system generated path for the log file. The file name will be **Log**. The directory path will be `${USER_INSTALL_ROOT}/filestores/com.ibm.ws.sib/<me_name>.<me_build>/log/`.

Restriction: For a cluster, the file store does not have the option of default directory paths. The administrator must specify the actual directory paths to be used by all file store files.

- Log directory path

Select this radio button and supply a non-default directory path for the log file. The file name will be Log.

- Same settings for permanent and temporary stores

The permanent and temporary store files can have identical settings. If you select this option, only one set of store file settings will appear in the window below this option (marked Permanent and Temporary stores). If this option is not selected, there will be separate sets for the Permanent store and the Temporary store displayed in the window (in that order).

- Minimum permanent store size

The minimum size of the permanent store file. The minimum value is 0 MB. The default is 200 MB.

- Unlimited permanent store size

Select this check box to remove any maximum size restrictions on the permanent store file.

- Maximum permanent store size

This setting will be ignored if the permanent store size is set to be unlimited. The minimum value is 50 MB. The default is 500 MB.

- Default permanent store directory path

Select this radio button to accept the default system generated path for the permanent store file. This directory path will be `${USER_INSTALL_ROOT}/filestores/com.ibm.ws.sib/<me_name>.<me_build>/store/`.

- Permanent store directory path

Select this radio button and supply a non-default directory path for the permanent store file. The file name for the permanent store file will be PermanentStore. The file name for the temporary store file will be TemporaryStore. If you choose to have the same settings for the permanent and temporary store files, these files will be co-located in the same directory with the indicated filenames.

3. Select **Next**.
4. Select **Finish** and save your changes.

For more information about file stores, refer to the IBM Information Center and “File stores” on page 569.

Adding a server or cluster to a bus using a default data store

Every messaging engine has a message store associated with it. If you elect to use the default data store, a Cloudscape database will be created automatically and initialized with the messaging engine tables. To create a bus member that automatically creates a messaging engine and uses the default Cloudscape database, do the following from the second window of the Add bus member wizard. See 9.8.5, “Adding a bus member” on page 622.

1. Select **Data Store**.

2. Select **Next**. The data store configuration window will appear. See Figure 9-43

Add a server, cluster or a WebSphere MQ server as a new member of the bus.

Step 1: Select server, cluster or WebSphere MQ server

Step 2: Select the type of message store

→ Step 3: Provide the message store properties

Step 4: Confirm the addition of a new bus member

Provide the message store properties

Select properties for the data store

☒ Create default data source with generated JNDI name

☐ Use existing data source

Data source JNDI name *

Schema name IBMWSSIB

Authentication alias (none)

☒ Create tables

Previous Next Cancel

Figure 9-43 Data source window with default settings option checked.

3. Select **Next** and then **Finish** and save your changes.

Adding a bus member with a different data store

This section discusses the steps required to create a bus member using a different data source from the default. In this section, we use DB2 as an example.

Creating a database

The first step is to create the new database and define the user IDs allowed to access the database. The privileges required are outlined in the Information Center. Refer to the *Data Stores* topic under the service integration bus administration topics for further information.

For example, The user ID for a DB2 database must have the following privileges:

- ▶ SELECT, INSERT, UPDATE, and DELETE privileges on the tables
- ▶ CREATETAB authority on the database
- ▶ USE privilege on the tablespace

- **CREATEIN** privilege on the schema

Use the **sibDDLGenerator** command to generate the DDL statements needed to create the data store for the messaging engine, including the proper privileges. For information about using this command, see the *sibDDLGenerator command* topic in the Information Center.

Creating a J2C authentication alias

To define access to the new database, define a J2C authentication alias containing the user ID and password defined in “Creating a database” on page 628.

1. Select **Security** → **Secure administration, applications and infrastructure**.
2. Under **Authentication**, expand the **Java Authentication and Authorization Service** section and select **J2C Authentication data**.
3. Click **New**.
 - a. Provide a name for this Alias. The alias name will be used later to identify this name as the one to access the database.
 - b. Provide a User ID and Password that have permission to access the resource you will be using.
 - c. Click **Apply** or **OK** and save your changes.

Creating a JDBC provider and data source

With this step, you define the database to the application server. First, a JDBC provider is defined to tell the application server how to find the libraries required to access the database.

1. Select **Resources** → **JDBC** → **JDBC Providers**.
2. Select the appropriate scope for the JDBC Provider. If you are adding a cluster as a bus member, then select that cluster as the scope. If you are adding a server as a bus member, then select the server as the scope.
3. Click **New**.
 - a. Select a database type. In this example, we use **DB2**.
 - b. Select the provider type. This is dependent on the database type. For a DB2 database, select **DB2 Universal JDBC Driver Provider**.
 - c. Select the implementation type. For DB2, use **Connection pool data source**.
4. Click **Next**.
5. Supply the absolute directory paths of the JDBC driver files according to the on-screen instructions.

6. Click **Next** and then click **Finish**.
7. The default values for the DB2 provider use a variable to designate the directory path where the JDBC drivers are found. Ensure that the DB2UNIVERSAL_JDBC_DRIVER_PATH environment variable is correctly set as per step 5 above:
 - a. Select **Environment** → **WebSphere Variables**.
 - b. Select an appropriate scope for the variable, usually node.
 - c. Set the value for the DB2UNIVERSAL_JDBC_DRIVER_PATH variable to be the path to the Java folder in the DB2 installation on the host appropriate to the scope selected.
 - d. Save your changes, if any.

Note: When the data source is being created at cluster scope, each node that has a server in the cluster must have the DB2 JAR files available on it. The DB2UNIVERSAL_JDBC_DRIVER_PATH variable must be set appropriately for every node.

8. Create a data source for the bus member. Select **Resources** → **JDBC** → **Data sources**.
9. Set the scope for the new data source.
10. Click **New** to create a new data source.
 - a. Provide a unique and meaningful **Data source name**.
 - b. Provide a **JNDI Name** for the data source. Remember this name because you will need to provide it when adding your cluster or server to the bus.
 - c. Provide a J2C authentication alias. This will be the credentials to connect to the database successfully.
 - d. Click **Next**.
 - e. Select the existing **DB2 Universal JDBC Driver Provider**.
 - f. Click **Next**.
 - g. Provide the Database name, Driver type and, optionally, the Server name. Get this information from your database administrator.
 - i. The database name must be the name of an existing DB2 Database.
 - ii. The driver type is 2 if the DB2 database exists locally or is catalogued locally. If the database is only available on a remote host, then the driver type is 4 and you must enter the Server name.

Note: There is no need to provide a component-managed authentication alias at this stage. That will be specified later in the data store of the messaging engine. Specifying the alias in either location is supported, but for tighter security control, we recommend that you specify it in the messaging engine's data store.

h. Click **Next** and **Finish** and save your changes.

Adding the bus member

Once the database and supporting definitions are in place, the bus member can be added. To add the bus member, do the following:

1. Select **Service integration** → **Buses**. Select the bus you want.
2. Select **Bus members** in the Additional Properties section.
3. Click **Add**.
4. To add a server to the bus, do the following:
 - a. Select **Server** on the radio button.
 - b. Select the server you want to add from the drop-down list.To add a cluster to the bus, do the following:
 - a. Select **Cluster** on the radio button.
 - b. Select the cluster you want to add from the drop down list.
5. Click **Next**.
6. Select **Data store** and click **Next**.

7. Select **Use existing data source**. See Figure 9-44 on page 632

Add a server, cluster or a WebSphere MQ server as a new member of the bus.

Provide the message store properties

Select properties for the data store

☐ Create default data source with generated JNDI name

☒ **Use existing data source**

Data source JNDI name
* jdbc/testds

Schema name
IBMWSSIE

Authentication alias
(none) ▼

☒ Create tables

Previous Next Cancel

Figure 9-44 Data source window with existing settings option checked

8. Supply the required **Data source JNDI name** of the JDBC data source you have created. This is the only required field.
9. The Schema name will be the default, and you may alter this if necessary. This is only required if you are using the same database instance to contain multiple data stores, each with its own schema.
10. Select the appropriate **Authentication alias** to connect to the database. This should be the same one that you selected when you configured the data source.
11. Ensure that the **Create tables** box is checked. The messaging engine will create all of the tables it needs in the database when it starts for the first time.
12. Click **Next** and then click **Finish**.

Important: The user ID in the authentication alias must have sufficient authority to be able to create tables in the database. Check with your database administrator.

If you do not want the data store to use an ID with the authority to create and drop tables, then your database administrator must create the tables for you before you start the messaging engine. See the Information Center section on *Enabling your database administrator to create the data store tables*.

Creating and using a WebSphere MQ Server

In this section, we will show you how to create a WebSphere MQ Server and add it as a member of a bus.

Creating a WebSphere MQ Server

To create a WebSphere MQ Server, do the following:

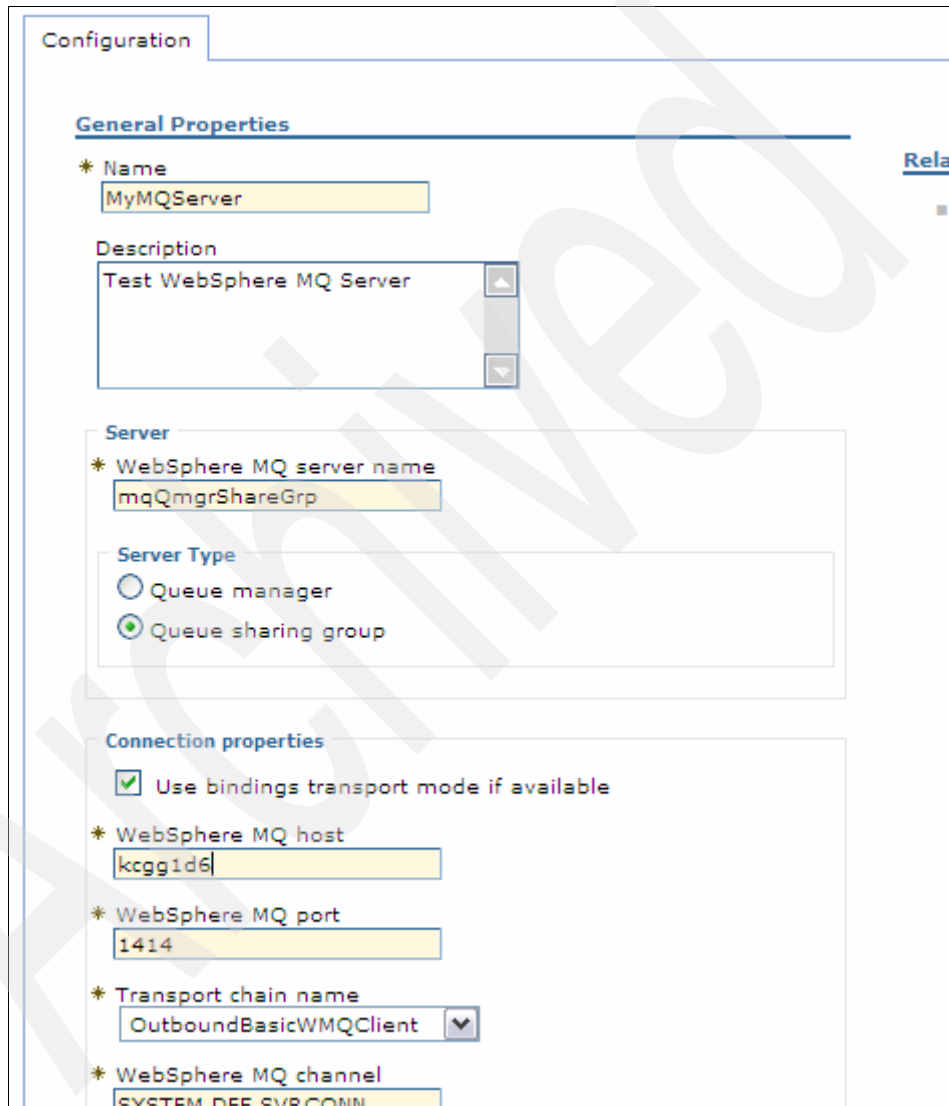
1. Select **Servers** → **WebSphere MQ Servers**.
2. Click **New**. You will see the WebSphere MQ Server configuration window. See Figure 9-46 on page 636.

The following properties need to be set:

- Name
Enter a meaningful name for the WebSphere MQ Server.
- Server
This is the name (as defined in WebSphere MQ) of the MQ queue manager, or the queue sharing group.
- Server type
Here you define the type of the server you want to connect to, a queue manager or a queue sharing group.
- Use bindings transport mode if available
If this is selected, bindings transport mode will always be used in preference to client transport mode. Otherwise, client transport mode will be used.
- WebSphere MQ host
The DNS host name or IP address of the machine that is hosting the MQ Queue manager.
- WebSphere MQ port
The TCP/IP port number (default 1414) used to connect to the WebSphere MQ queue manager.

– Transport chain name

Select the appropriate transport chain from the drop down list. This is used to establish an outbound network connection to the WebSphere MQ Server. See 9.2.2, “Service integration bus transport chains” on page 563 for further information:



The image shows a screenshot of the 'Configuration' window for a WebSphere MQ Server. The window is titled 'Configuration' and has a tab labeled 'General Properties'. Below the tab, there are several sections: 'Name' with a text field containing 'MyMQServer', 'Description' with a text area containing 'Test WebSphere MQ Server', 'Server' with a section for 'WebSphere MQ server name' containing 'mqQmgrShareGrp' and 'Server Type' with radio buttons for 'Queue manager' and 'Queue sharing group' (selected), 'Connection properties' with a checked box for 'Use bindings transport mode if available', 'WebSphere MQ host' containing 'kcg1d6', 'WebSphere MQ port' containing '1414', 'Transport chain name' with a dropdown menu showing 'OutboundBasicWMQClient', and 'WebSphere MQ channel' containing 'SYSTEM.DEF.SVRCONN'. A large, faint watermark 'AS/400' is visible across the center of the image.

Figure 9-45 WebSphere MQ Server configuration window

- WebSphere MQ Channel

This is the name of the connection channel, as defined in WebSphere MQ.

Other properties on the window will be the defaults, and information about these can be found in the IBM Information Center.

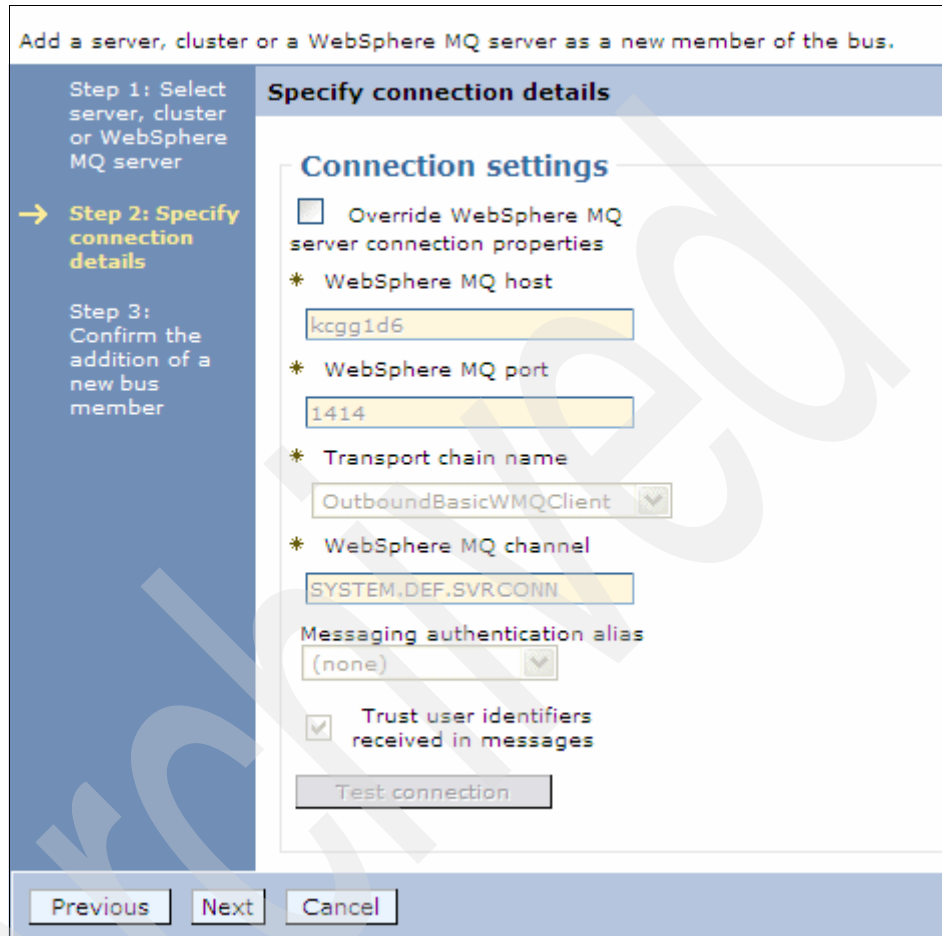
3. Click **OK** and save your changes.

Adding a bus member

To add a WebSphere MQ Server as a bus member, do the following:

1. Select **Service integration** → **Buses**. Select the bus you want.
2. Select **Bus members** in the Additional Properties section.
3. Click **Add**.
4. Select **WebSphere MQ server** on the radio button and select the server from the drop-down list.

5. Click **Next**. You will see the connection settings window. See Figure 9-46 on page 636.



The screenshot shows a window titled "Add a server, cluster or a WebSphere MQ server as a new member of the bus." with a sidebar on the left containing three steps: "Step 1: Select server, cluster or WebSphere MQ server", "Step 2: Specify connection details" (highlighted with a yellow arrow), and "Step 3: Confirm the addition of a new bus member". The main area is titled "Specify connection details" and contains a "Connection settings" section. This section includes an unchecked checkbox for "Override WebSphere MQ server connection properties". Below this are five required fields marked with an asterisk: "WebSphere MQ host" (text box with "kcg1d6"), "WebSphere MQ port" (text box with "1414"), "Transport chain name" (dropdown menu showing "OutboundBasicWMQClient"), "WebSphere MQ channel" (text box with "SYSTEM.DEF.SVRCONN"), and "Messaging authentication alias" (dropdown menu showing "(none)"). There is also a checked checkbox for "Trust user identifiers received in messages" and a "Test connection" button. At the bottom of the window are three buttons: "Previous", "Next", and "Cancel".

Figure 9-46 WebSphere MQ Server connection settings window

This window gives the opportunity to review and override some of the WebSphere MQ Server connection properties. This may be useful in a multiple bus topology where you may need bus-specific settings for the server.

If you wish to override the inherited connection settings, select the **Override WebSphere MQ server connection properties** check box and alter the connection settings as desired.

6. Click **Next** and **Finish** and save your changes.

9.8.6 Creating a queue destination

Queue destinations are destinations that you can configure for point-to-point messaging.

1. Select **Service integration** → **Buses**.
2. Select the bus on which you want to create a queue.
3. Select **Destinations** in the Destination resources section. See Figure 9-47.



Figure 9-47 Default destinations

The Destinations window shows two destinations that are created automatically for you. The `Default.Topic.Space` is a default topic space that can be used for publish/subscribe messaging. It can be deleted. The `_SYSTEM.Exception.Destination` is a built-in queue that cannot be deleted.

4. Click **New**. See Figure 9-48 on page 638.

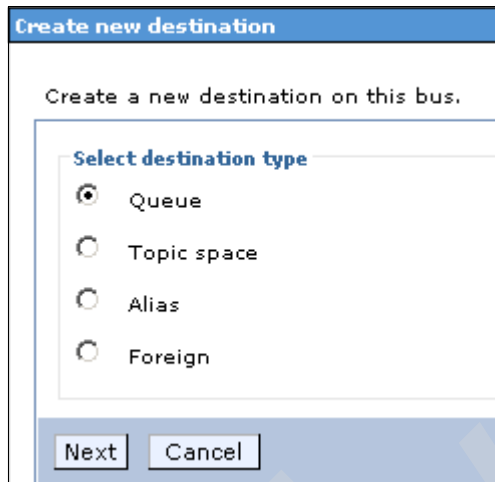


Figure 9-48 Options when creating a new destination

Select **Queue** from the radio button list and click **Next**. See Figure 9-49.

5. Provide an identifier and optional description for the queue.

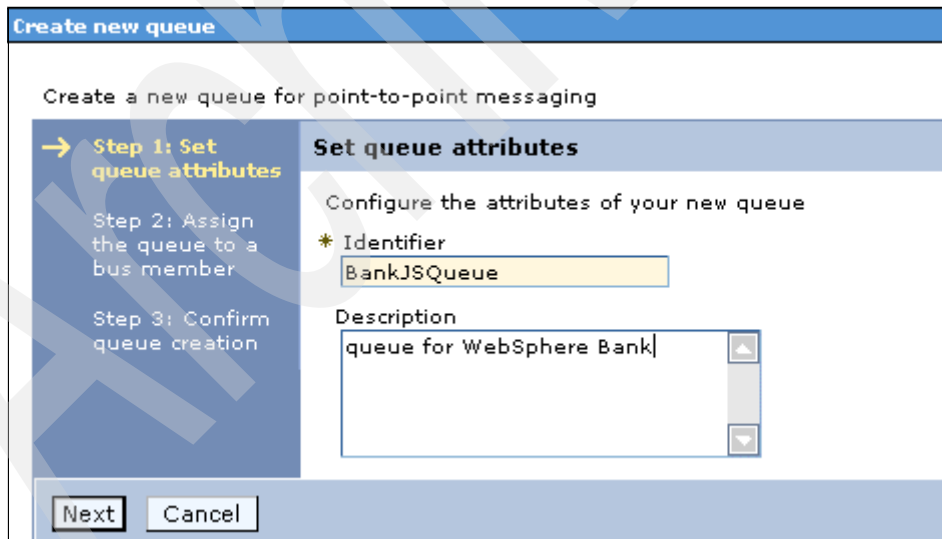


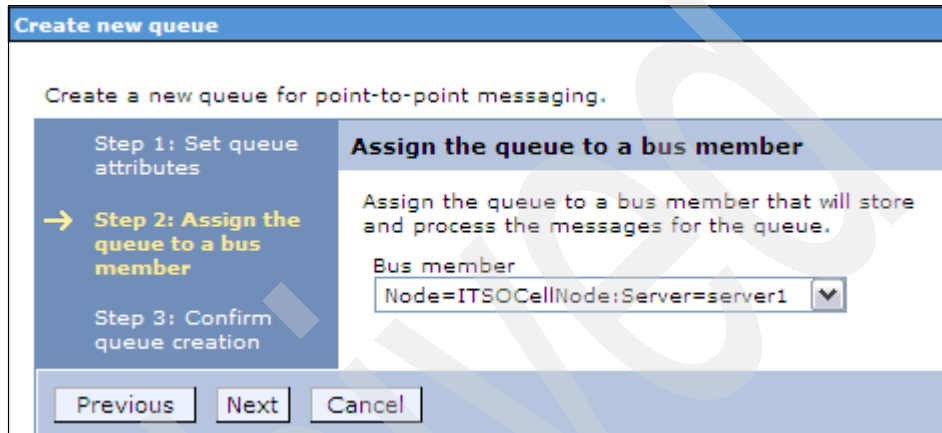
Figure 9-49 Provide an identifier for your destination

If your application uses the JMS interface, it is not sufficient to create a destination on the bus. A JMS destination referencing the bus destination

must also be created. The identifier value specified here must match the Queue name property of the JMS queue definition (see “JMS queue configuration” on page 476).

Click **Next**. See Figure 9-50 on page 639.

6. Select a bus member for the queue point for this queue from the list for the queue. Click **Next**.



The screenshot shows a wizard window titled "Create new queue". The main heading is "Create a new queue for point-to-point messaging." On the left, a vertical pane lists three steps: "Step 1: Set queue attributes", "→ Step 2: Assign the queue to a bus member" (which is highlighted), and "Step 3: Confirm queue creation". The main area of the wizard is titled "Assign the queue to a bus member" and contains the instruction "Assign the queue to a bus member that will store and process the messages for the queue." Below this, there is a label "Bus member" and a dropdown menu showing the selected value "Node=ITSOCellNode:Server=server1". At the bottom of the wizard, there are three buttons: "Previous", "Next", and "Cancel".

Figure 9-50 Select a bus member for the queue

7. Click **Next** and **Finish** and then save your changes.

9.8.7 Creating a topic space destination

Topic space destinations are destinations that can be configured for publish/subscribe messaging.

1. Select **Service integration** → **Buses**.
2. Select the bus on which you want to create a topic space on.
3. Select **Destinations** in the Destination resources section.
4. Click **New**.
5. Select **Topic space** from the list and click **Next**.
6. Provide an identifier and optional description for your topic space.
7. Click **Next**.
8. Click **Finished**.
9. Save your changes.

9.8.8 Creating an alias destination

Alias destinations refer to another destination, potentially on a foreign bus, providing an extra level of indirection for messaging applications. An alias destination can also be used to override some of the values specified on the target destination, such as default reliability and maximum reliability. Foreign buses are discussed in 9.1.7, “Foreign buses” on page 555.

1. Select **Service integration** → **Buses**.
2. Select the bus on which you want to create a topic space.
3. Select **Destinations** in the Destination resources section.
4. Click **New**.
5. Select **Alias** from the list and click **Next**. See Figure 9-51 on page 641.

Configure the attributes of your new alias destination

* Identifier

Bus

description

* Target identifier

Target bus

Quality of Service

Enable producers to override default reliability

Default reliability

Maximum reliability

Send allowed

Receive allowed

Default forward routing path

☒ Delegate authorization check to target destination

Figure 9-51 Alias destination properties

The properties to note are:

- Identifier

This field is the destination name as known by the applications.

- Bus

Enter the name of the bus used by applications when referring to the alias destination.

If the destination that clients will attempt to access is known to them to be on a foreign bus, then select that bus from the menu. An example of this is if a foreign destination is configured in the JMS layer and you want to redirect client requests for that destination.

If the bus does not appear in the list, select **Other**, specify from the list, and enter the name of the bus in the text box.

If you leave the Bus field empty, the alias destination is created on the local bus.

- Target identifier

Enter the identifier of the target destination to which you want this alias destination to route messages. If the alias destination is targeting a queue provided by WebSphere MQ, type the value as a concatenation of the queue name and the queue manager name, for example, queue_name@qmanager_name; for example: Queue1@Qmgr2.

- Target bus

Enter the name of the bus or foreign bus hosting the target destination. This can be the name of a foreign bus representing a WebSphere MQ network. The default is the name specified for the Bus property.

Override any of the other values on the window that you want to override for the destination.

6. Click **Next**.
7. Click **Finished**.
8. Save your changes.

9.8.9 Adding messaging engines to a cluster

When you add a cluster to a bus, you get one messaging engine. To define additional messaging engines, do the following:

1. Ensure that you have defined a data source that the new messaging engine will use for its data store before starting this section (see “Creating a JDBC provider and data source” on page 629).
2. Select **Service integration** → **Buses**. Select the bus you want to use.
3. Select **Bus members** in the Additional Properties section.

4. Select the cluster bus member to which you want to add an additional messaging engine. This will display the list of messaging engines that are defined for the cluster bus member. See Figure 9-52 on page 643.

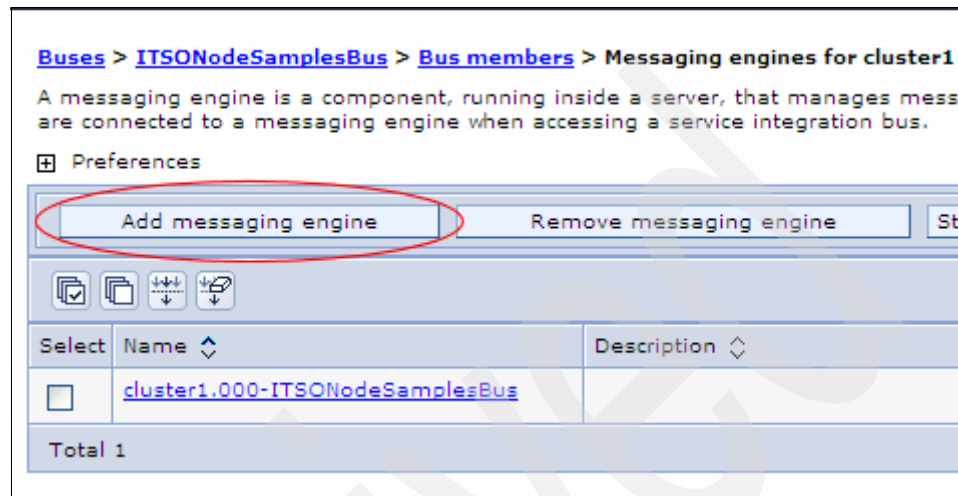


Figure 9-52 Messaging engines as part of a cluster bus member

5. Click **Add messaging engine**.
6. Select the type of message store and click **Next**.
7. Enter the required information for the message store. For information about using multiple message stores, see 9.2.3, "Message stores" on page 568.
8. Click **Next** and **Finish** and save your changes.

9.8.10 Setting up preferred servers

Configure a messaging engine that you prefer to run on one server or a group of servers in a cluster using a core groups policy. The use of policies is required if you want to workload-manage your messaging with the bus.

Note: Before attempting to configure a system for workload management and high availability, consult the following:

- ▶ 9.3, “High availability and workload management” on page 594
- ▶ The *Configuring high availability and workload sharing of service integration* topic in the Information Center
- ▶ *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392
- ▶ *WebSphere Application Server Network Deployment V6: High Availability Solutions*, SG24-6688

Setting up a policy with the appropriate values can give many different behaviors, including the following:

- ▶ A messaging engine will have an affinity for one particular server in the cluster. If that server fails, then the messaging engine will run on other servers, but will move back to the preferred server as soon as it becomes available. This is set up by having a One-of-N Policy defined with one preferred server configured, Preferred servers only set to false, and Fail back set to true.
- ▶ A messaging engine will run on only one server inside the cluster. This means that the messaging engine cannot fail over to another server in the cluster and will only ever run on the defined server. This can be set up by having a One-of-N Policy with one preferred server and Preferred servers only set to true.

Important: If you have more than one messaging engine defined on a cluster bus member and do not define additional core group policies to set up preferred servers, then all messaging engines will start and run on the first server to become available.

To create a core group policy for a messaging engine, do the following:

1. Select **Servers** → **Core groups** → **Core group settings**.

2. Select the **DefaultCoreGroup**. This will show the properties for the default core group. See Figure 9-53 on page 645.

Core groups

[Core groups](#) > DefaultCoreGroup

Use this page to specify the settings for a core group.

Runtime Configuration

General Properties

* Name
DefaultCoreGroup

Description
Default Core Group. The default core group cannot be deleted.

* Number of coordinators
1

Transport type
☒ Channel framework

Channel chain name

Additional Properties

- Core group servers
- Custom properties
- Policies**
- Preferred coordinator servers

Related Items

- Core group bridge settings

Figure 9-53 Default Core Group

3. Select **Policies** in the Additional Properties section. This will show you the list of policies defined for the core group. Two policies are created by default. Do not delete or modify these policies. See Figure 9-54 on page 646.



Core groups				
Core groups > DefaultCoreGroup > Policies				
Use this page to view and manage the policies associated with a core group. Coordinators use these policies to determine on which servers the core group members are activated or deactivated.				
⊞ Preferences				
New Delete				
Select	Name	Description	Policy type	Match criteria
☐	Clustered TM Policy	TM One-Of-N Policy	One of N policy	type=WAS_TRANSACTIONS
☐	Default SIBus Policy	SIBus One-Of-N Policy	One of N policy	type=WSAF_SIB
Total 2				

Figure 9-54 Predefined core group policies in the default core group

4. Click **New**.
5. From the drop-down list, select the **One of N Policy**. Click **Next**. See Figure 9-55 on page 647.
6. Enter a name for the new policy. It might be helpful if the name includes the name of the messaging engine for which you are creating this policy.
Enable **Fail back** and **Preferred servers only** as desired. These settings can be changed later.
Click **Apply**.

Messages

The policy must have at least one match criteria defined.

Changes have been made to your local configuration. You can:

- Save directly to the master configuration.
- Review changes before saving or discarding.

An option to synchronize the configuration across multiple nodes after saving can be enabled in Preferences.

The server may need to be restarted for these changes to take effect.

Core groups > DefaultCoreGroup > Policies > New > cluster1.000-ITSOBUSPolicy

Use this page to define a One of N policy. This type of policy keeps one group member active at all times.

Configuration

General Properties

* Name

cluster1.000-ITSOBUSPolicy

* Policy type

One of N policy

Description

* Is alive timer

0 seconds

Additional Properties

Custom properties

Match criteria

Preferred servers

Figure 9-55 Defining a new policy

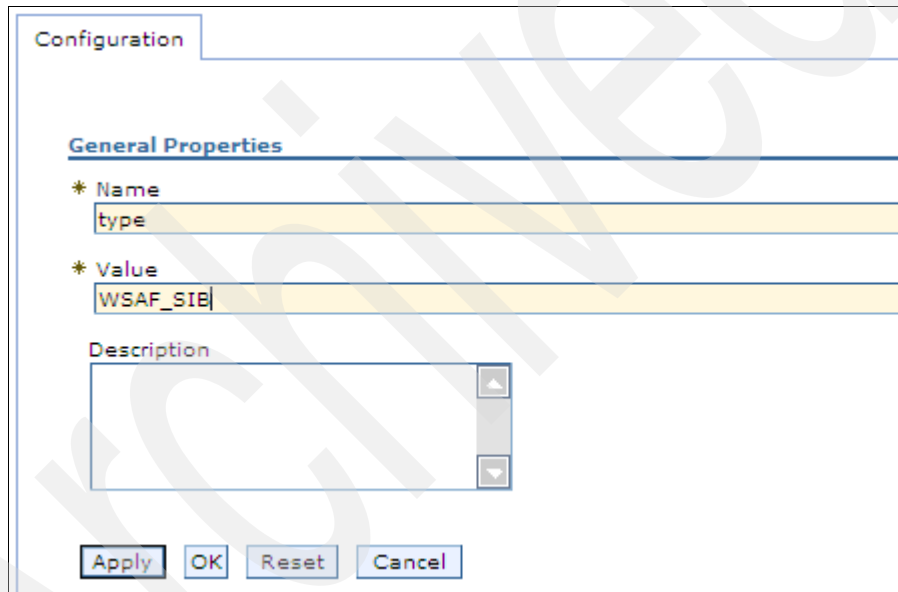
A warning will show that you must define at least one match criteria. Match criteria are name and value pairs used to match server components, such as messaging engines.

Chapter 9. Default messaging provider 647

Important: Be aware that if you set **preferred servers only** that this can prevent the messaging engine from being highly available. If the messaging engine or the server it runs on fails or stops and no other servers that are preferred are available, then the messaging engine cannot be started on other servers that are available in the cluster. They are not preferred and only preferred servers can be used.

7. Select **Match criteria** in the Additional Properties section.
8. Click **New**. See Figure 9-56.

Enter type for the name and WSAF_SIB for the value. This match criteria will match any messaging engine.



The image shows a 'Configuration' dialog box with a 'General Properties' section. Under 'Name', the text 'type' is entered. Under 'Value', the text 'WSAF_SIB' is entered. There is a 'Description' field which is currently empty. At the bottom of the dialog, there are four buttons: 'Apply', 'OK', 'Reset', and 'Cancel'.

Figure 9-56 Defining match criteria for any messaging engine

Click **OK**.

9. Click **New** to define another set of match criteria.
10. Enter WSAF_SIB_MESSAGING_ENGINE for the name and the messaging engine name for the value. Click **OK**.
11. Return to your policy by clicking the policy name in the navigation trail. See Figure 9-57 on page 649.

12. Click **Preferred** so that it is selected.

12. Click **Preferred** so that it is selected.

12. Click **Preferred** so that it is selected.

13. Select the servers you want to configure as preferred and click **Add**.

You can select as many preferred servers as you want. All preferred servers must be servers that are in the cluster on which the messaging engine is defined. Do not select a node agent or deployment manager. See Figure 9-58 on page 650.

Preferred servers have an order of preference. The higher up the list of preferred servers, the more preferred the server will be. To move a server up or down the list, select the server and click **Move up** or **Move down**. If **Fail back** is enabled, then a messaging engine will fail over to the highest available server in the list.

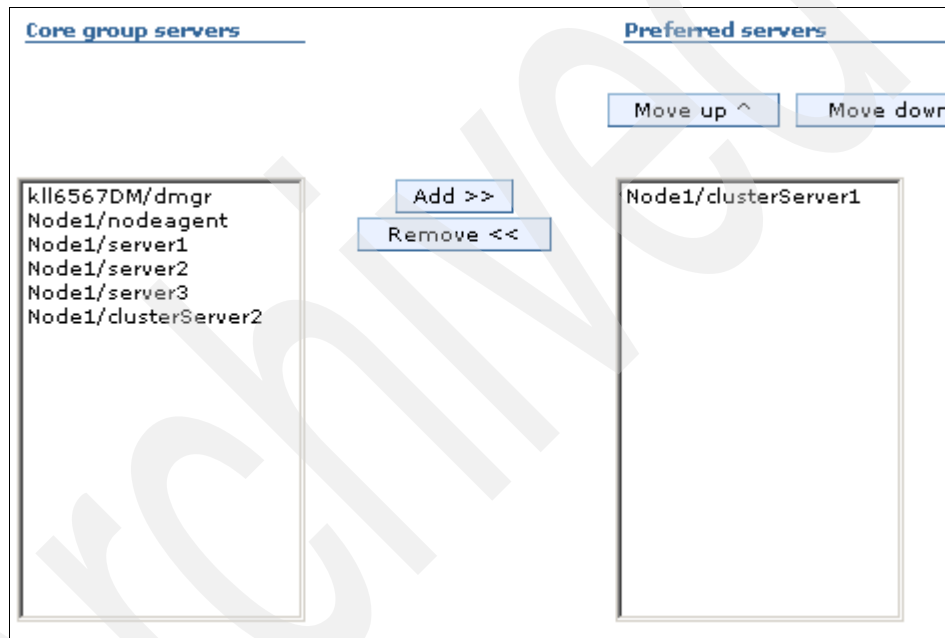


Figure 9-58 Selecting preferred servers for a core group policy

14. Click **OK** and save your changes.

9.8.11 Setting up a foreign bus link to a service integration bus

To define a foreign bus to the bus from which you want to access it, do the following:

1. Select **Service integration** → **Buses**.
2. Select the bus from which you want to access the foreign bus.
3. Select **Foreign buses** in the Topology section.

4. Click **New**. See Figure 9-59 on page 651.
5. Provide the **Name** of the foreign bus.

Important: When your foreign bus is a bus, then this name must match exactly the name of that bus.

Create foreign bus routing definition

Create the routing definition for the foreign bus, to define the routing type and properties.

→ Step 1: Foreign bus properties

Step 2: Routing definition type

Step 3: Routing definition properties

Step 4: Summary

Foreign bus properties

* Name
OtherBus

Description
My other service integration bus

☒ Send allowed

Next Cancel

Figure 9-59 Creating a new foreign bus

Checking the **Send allowed** box allows this bus to send messages to destinations on the foreign bus. This is the default.

You can change this setting at any time. This can be useful if you want to disable a foreign bus for a short time, for example, while configuration changes are being made.

Click **Next**. See Figure 9-60.

6. Select the appropriate value for the routing type.

Create foreign bus routing definition

Create the routing definition for the foreign bus, to define the routing type and properties.

Step 1: Foreign bus properties
→ **Step 2: Routing definition type**
Step 3: Routing definition properties
Step 4: Summary

Routing definition type

Select the type of routing to the foreign bus, a direct service integration bus link, a direct WebSphere MQ link, or an indirect route via another bus.

Routing type
Direct, service integration bus link
Direct, service integration bus link
Direct, WebSphere MQ link
Indirect

Previous Next Cancel

Figure 9-60 Selecting the type of foreign bus

To define another bus, select **Direct, service integration bus link** from the menu. Click **Next**. See Figure 9-61.

7. Optionally, define outbound and inbound user IDs.

Create foreign bus routing definition

Create the routing definition for the foreign bus, to define the routing type and properties.

Step 1: Foreign bus properties
Step 2: Routing definition type
→ **Step 3: Routing definition properties**
Step 4: Summary

Routing definition properties

Routing type
Direct, service integration bus link

Inbound user ID
[Text input field]

Outbound user ID
[Text input field]

Previous Next Cancel

Figure 9-61 Define inbound and outbound user IDs

The inbound user ID authorizes individual messages arriving from the foreign bus to destinations in this bus. When set, this property replaces the user ID in messages entering this bus from the foreign bus. If this is not a secure bus, this property does not affect messages.

The outbound user ID replaces the user ID that identifies the source of a message in all messages being sent to the foreign bus. When set, this property replaces the user ID in messages leaving this bus for the foreign bus. The foreign bus also uses this ID to authorize the message to its destination if both buses are secure buses and the foreign bus has not overridden the user ID with its own inbound user ID.

8. Click **Next**.
9. Click **Finish** and save your changes.

Define the link to the bus

Now that your bus knows about the foreign bus, you will have to set up the link to that bus. This link will be managed by a particular messaging engine on your bus. A link must be created on each bus and it is important that the link has the same name on each bus.

1. Select **Service integration** → **Buses**. Select the bus you want to use.
2. Select **Messaging engines** in the Topology section and select the messaging engine you want to host the link.
3. Select **Service integration bus links** in the Additional Properties section.

4. Click **New**. See Figure 9-62 on page 654 and fill in the following properties:

The screenshot shows a web browser window titled "Buses". The breadcrumb navigation is: [Buses](#) > [ITSONodeSamplesBus](#) > [Messaging engines](#) > [ITSOCellNode.server1-ITSONodeSamplesBus](#) > [Service integration bus links](#) > **New**. Below the navigation, a description reads: "Links between this messaging engine and messaging engines in foreign service integration buses." The "Configuration" tab is selected. The "General Properties" section contains the following fields: "Name" (ITSOBusToOtherBusLink), "UUID" (AF5EC6743CE33689), "Description" (empty text area), "* Foreign bus name" (OtherBus), "* Remote messaging engine name" (Node1.server2-OtherBus), "Target inbound transport chain" (empty text field), "Bootstrap endpoints" (localhost:7278), and "Authentication alias" (none). A "Related Items" section on the right lists "JAAS - J2C authentication data".

Figure 9-62 Defining a service integration bus link

— Name

Enter a name for the link. It might be helpful if this name includes the names of the buses you are linking.

Important: This link name must be the *exactly the same* as the link name on the other bus.

- Foreign bus name

Enter the name of the messaging engine in the foreign bus to which you are linking. This name must also match exactly the name of the messaging engine in the foreign bus hosting the link and is required to prevent configuration changes on the other bus from causing problems with the link.

- Bootstrap endpoints

Provide bootstrap endpoints to allow your bus to connect to the foreign bus. This field is equivalent to the Provider endpoints field for a default messaging provider connection factory. Both provide a list of endpoints to be used to connect to a SIB service.

See 8.7.1, “JMS client run time environment” on page 521.

- Authentication alias

If the foreign bus is secure, then you need to provide authentication data for the link.

5. Click **Apply** or **OK** and save your changes.

Important: You must configure a corresponding foreign bus and service integration bus link on the other bus to complete the link. Ensure that the name of the link is the same in both buses.

Configuring topic space mappings to a foreign bus

This section discusses the steps required to create a topic space mapping between two buses. Before starting this section, you must have defined a foreign bus that is a service integration bus.

- ▶ Select **Service integration** → **Buses**. Select the local bus.
- ▶ Select **Foreign buses** in the Topology section.
- ▶ Select the foreign bus you want to create a topic mapping.
- ▶ Select **Service integration bus link routing properties** in the Additional Properties section.
- ▶ Select **Topic space mapping** in the Additional Properties section.
- ▶ Optionally, enter a description.

Click **Apply**.

Note: You have to click **Apply** even if you do not enter a description.

- ▶ Select **Topic space map entries** in the Additional Properties section.

- Click **New**.
 - Enter the name of the Local topic space from which you want to receive published messages.
 - Enter the name of the Remote Topic space from which you want to receive published messages.
- Click **Apply** or **OK** and save your changes.

9.8.12 Setting up a foreign bus link to an MQ queue manager

A WebSphere MQ link allows your service integration bus to exchange messages with a WebSphere MQ queue manager.

Note: Before creating these definitions, review the information in 9.2.6, “WebSphere MQ links” on page 584.

First, you must define a foreign bus and define it in your bus. From there, enter information in the following fields.

1. Select **Service integration** → **Buses**. Select the bus you want to use.
2. Select **Foreign buses** in the Topology section.
3. Click **New**. See Figure 9-63. Enter information into the following fields:

Create foreign bus routing definition

Create the routing definition for the foreign bus, to define the routing type and properties.

→ **Step 1: Foreign bus properties**

Step 2: Routing definition type

Step 3: Routing definition properties

Step 4: Summary

Foreign bus properties

* Name
QM_itso

Description
Foreign bus to MQ

☒ Send allowed

Next Cancel

Figure 9-63 Creating a new foreign bus

- Name

Enter the name of the foreign bus.

- Send allowed

Checking the **Send allowed** box allows this bus to send messages to destinations on the foreign bus. This is the default. You can change this setting at any time. This can be useful if you want to disable a foreign bus for a short time, for example, while configuration changes are being made.

Click **Next**. See Figure 9-64.

4. Select **Direct, WebSphere MQ link** from the menu.

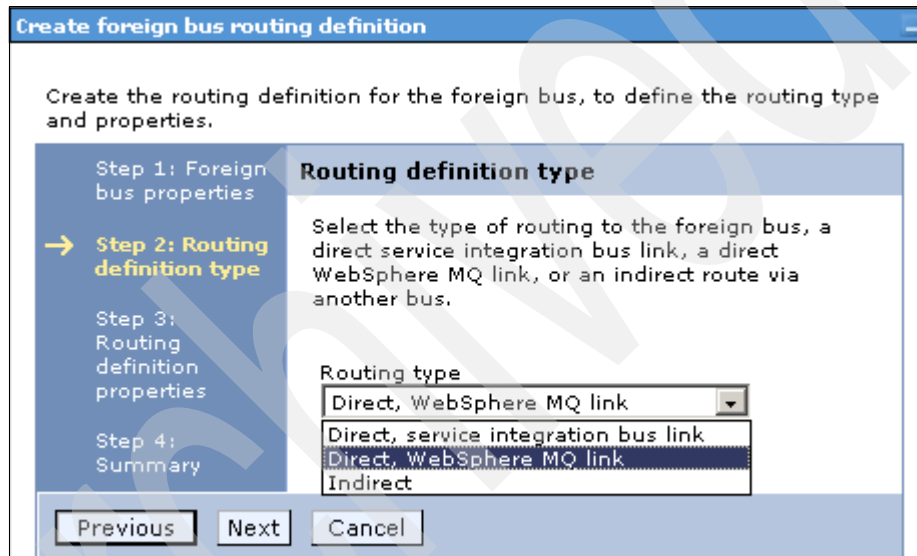


Figure 9-64 Selecting the type of foreign bus

5. Define inbound and outbound user IDs (optional)

The inbound user ID authorizes individual messages arriving from the foreign bus to destinations in this bus. When set, this property replaces the user ID in messages entering this bus from the foreign bus. If this is not a secure bus, this property does not affect messages.

The outbound user ID replaces the user ID that identifies the source of a message in all messages being sent to the foreign bus. When set, this property replaces the user ID in messages leaving this bus for the foreign bus. The foreign bus also uses this user ID to authorize the message to its destination if both buses are secure buses and the foreign bus has not overridden the user ID with its own inbound user ID.

6. Click **Next**.

7. Click **Finish** and save your changes.

Define the WebSphere MQ link

Now that your bus knows about the foreign bus, set up the link to that MQ queue manager. This link is managed by a particular messaging engine on your bus.

Important: If you are unsure of any of the correct MQ values to supply for the MQ link, then refer to your MQ administrator or documentation for more information.

1. Select **Service integration** → **Buses**.
2. Select the bus you want to use.
3. Click **Messaging engines** and select the messaging engine you want to host the link.
4. Select **WebSphere MQ Links** in the Additional Properties section.

5. Click **New**. See Figure 9-65 on page 659 and enter information in the following fields:

The screenshot shows a wizard titled "Create new WebSphere MQ link". The left pane lists four steps: "Step 1: General WebSphere MQ link properties" (selected), "Step 2: Sender channel WebSphere MQ link properties", "Step 3: Receiver channel WebSphere MQ link properties", and "Step 4: Summary of WebSphere MQ link properties". The right pane, titled "General WebSphere MQ link properties", contains the following fields:

- Name:** Text box containing "MyMQLink".
- Description:** Empty text box.
- Foreign bus name:** Dropdown menu showing "QM_itso".
- Queue manager name:** Text box containing "SIB_QM".
- Batch size:** Text box containing "50".
- Maximum message size:** Text box containing "4194304" followed by "bytes".
- Heartbeat interval:** Text box containing "300" followed by "seconds".
- Sequence wrap:** Text box containing "999999999".
- Nonpersistent message speed:** Dropdown menu showing "Fast".
- Adoptable:** Checked checkbox.
- Initial state:** Dropdown menu showing "Started".

At the bottom of the wizard are "Next" and "Cancel" buttons.

Figure 9-65 Defining properties for a new MQ link

– Name

Enter a name for the link. It might be helpful if this name includes the name of the foreign bus for which you are creating the link.

- Foreign bus name

From the menu, select the name of the foreign bus to which this link will connect. This should be the name of the queue manager that is participating in the link.

- Queue manager name

This is the queue manager name by which the MQ queue manager will know this bus. You must ensure that the MQ queue manager participating in this link is configured to know about this bus as another queue manager using this queue manager name.

- Nonpersistent message speed

This field defines whether the channel to MQ will have MQ's NPMSPEED channel attribute set to fast or normal.

- Adoptable

This option, enabled by default, provides function similar to MQ's ADOPTMCA function. If selected, the receiver channel can be reused when the sender channel fails or has to be restarted.

Click **Next**.

6. You must now provide details on how the link will send information to the MQ queue manager. See Figure 9-66 on page 661. Enter the following:

Create new WebSphere MQ link

Wizard to create a new WebSphere MQ link

Step 1: General WebSphere MQ link properties

→ **Step 2: Sender channel WebSphere MQ link properties**

Step 3: Receiver channel WebSphere MQ link properties

Step 4: Summary of WebSphere MQ link properties

Sender channel WebSphere MQ link properties

Sender MQ channel name
ITSOSIBTOMQ

Host name

Port
1414

* Transport chain
OutboundBasicMQLink

Disconnect interval
900 seconds

Short retry count
10

Short retry interval
60 seconds

Long retry count
999999999

Long retry interval
1200 seconds

Initial state
Started

Previous Next Cancel

Figure 9-66 Providing the link with details on how to send messages to MQ

- Sender MQ channel name

This is the name of the receiver channel that the link will send messages to in the MQ queue manager.

- Host name

Enter the host name or IP address of the server hosting the MQ queue manager.

- Port

If the MQ queue manager is using a port other than the default port of 1414, then enter that information.

- Transport chain

Select the appropriate transport chain from the menu. See 9.2.2, “Service integration bus transport chains” on page 563 for further information.

Click **Next**. See Figure 9-67.

7. Enter information about the virtual queue manager. Remember, this link performs as a virtual queue manager for WebSphere MQ. Enter the following information:

Create new WebSphere MQ link

Wizard to create a new WebSphere MQ link

Step 1: General WebSphere MQ link properties

Step 2: Sender channel WebSphere MQ link properties

→ **Step 3: Receiver channel WebSphere MQ link properties**

Step 4: Summary of WebSphere MQ link properties

Receiver channel WebSphere MQ link properties

Receiver MQ channel name
MQToSIB

Inbound nonpersistent message reliability
Reliable

Inbound persistent message reliability
Assured

Initial state
Started

Previous Next Cancel

Figure 9-67 Providing the link with details on how MQ will send messages to it

- Receiver MQ channel name

The MQ queue manager participating in the link must be configured with a sender channel of this name.

- Provide default information for mapping incoming persistent and nonpersistent MQ messages into service integration messages, see “Reliability” on page 551 for more information about service integration message reliability.

Click **Next**.

8. Click **Finish** and save your changes.

WebSphere MQ considerations: Ensure that the WebSphere MQ queue manager participating in the link has the appropriate sender and receiver channels defined. Consult your MQ administrator or documentation for details on how to perform this configuration.

► **Sender channel**

This channel must have the same name as the name defined in the MQLink’s receiver channel.

The connection name is the IP address or host name for the server hosting the messaging engine on which the link is defined.

The port used should match the value of the SIB_MQ_ENDPOINT_ADDRESS port defined for the server hosting the messaging engine on which the link is defined. The default is 5559. To find this value through the administrative console, do the following:

- a. Select **Servers** → **Application Servers**.
- b. Select the server hosting the messaging engine.
- c. Under Communications expand the **Ports** heading. Find the port number for SIB_MQ_ENDPOINT_ADDRESS.

► **Receiver channel**

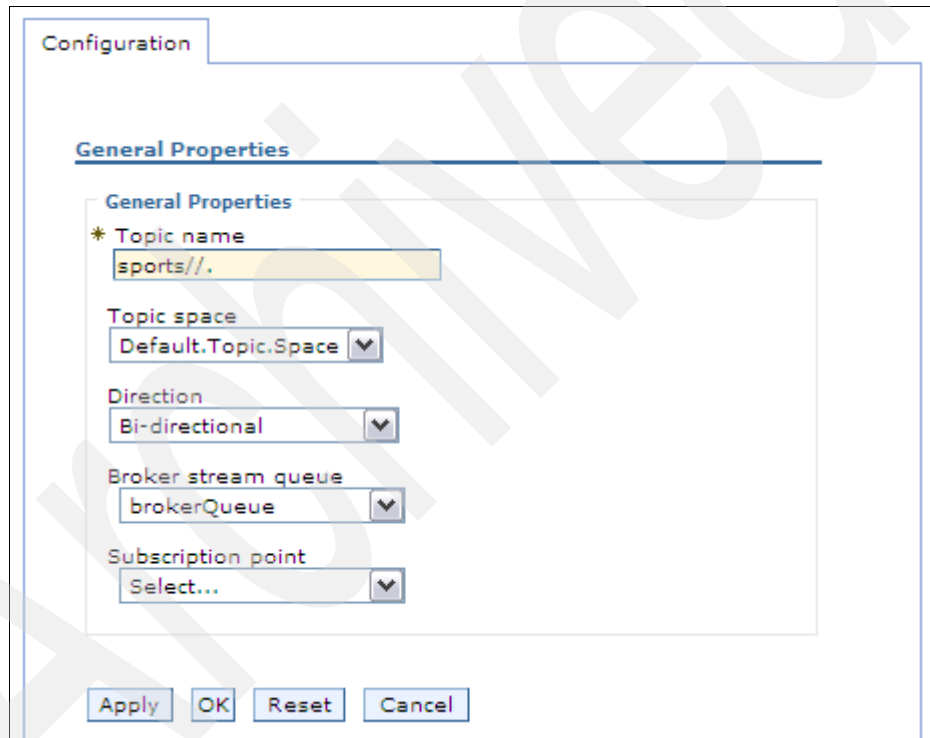
This channel must have the same name as the name defined in the MQLink’s sender channel.

Configuring topic space mappings to WebSphere MQ

To configure an MQ publish/subscribe profile, define a WebSphere MQ link to the WebSphere MQ network. The link does not need to be directly to the broker’s queue manager. However, it must be to a queue manager that is able to route to the broker’s queue manager.

1. Select **Service integration** → **Buses**. Select the bus you want to use.
2. Click **Messaging engines** and select the messaging engine that hosts the MQ link.
3. Select **WebSphere MQ links** in the Additional Properties section.

4. Select the MQ link on which you want to define a MQ publish/subscribe profile.
5. Select **Publish/subscribe broker profiles** in the Additional Properties section. Click **New**.
 - Enter a name for the profile.
 - Enter the name of the queue manager associated with the broker.Click **Apply**. Define some topic mappings to link MQ topics to service integration topics.
6. Select **Topic mappings** in the Additional Properties section.
7. Click **New**. See Figure 9-68 and enter information in the following fields:



The screenshot shows a 'Configuration' dialog box with a 'General Properties' section. The 'General Properties' section contains the following fields:

- Topic name**: A text input field containing 'sports//.'
- Topic space**: A dropdown menu with 'Default.Topic.Space' selected.
- Direction**: A dropdown menu with 'Bi-directional' selected.
- Broker stream queue**: A dropdown menu with 'brokerQueue' selected.
- Subscription point**: A dropdown menu with 'Select...' selected.

At the bottom of the dialog are four buttons: 'Apply', 'OK', 'Reset', and 'Cancel'.

Figure 9-68 Defining a new Topic mapping

- Topic name
Enter the name of the topic that you want to map. This name is the topic name that will be linked in the service integration bus and MQ. You can use the bus wildcard “/.” in this topic name to map a group of topics. For example “stock//.” means all messages with “stock” at the beginning of the

topic. For more information about the use of wildcard characters when specifying topic names, refer to “Topic specific connection properties” on page 474.

► Topic space

Select the Topic space in which this topic will be published on the bus.

– Direction

Select the desired direction of the mapping.

- Bi-directional

Messages published in either WebSphere MQ or the bus will be published in both.

- To WebSphere MQ

Messages published in the bus will be published in WebSphere MQ, but messages published in WebSphere MQ will not be published in the bus.

- From WebSphere MQ

Messages published in WebSphere MQ will be published in the bus, but messages published in the bus will not be published in WebSphere MQ.

– Broker stream queue

Select the appropriate broker stream queue from the menu, if required. The broker stream queue is the queue in MQ to which the message broker is connected. If the queue does not appear in the list then, select **Other, please specify**. A text entry box will appear to the right of the drop-down menu. Enter the name of the queue there.

Note: Broker stream queue is required if you want to send messages to WebSphere MQ. If your topic mapping is Bi-directional, To WebSphere MQ or if it is From WebSphere MQ and your applications need to be able to send reply messages to publications received, then a broker stream queue must be specified.

– Subscription point

Select an appropriate subscription point from the menu, if required. If the subscription point does not appear in the list then select **Other, please specify**. A text entry box will appear to the right of the drop-down menu. Enter the name of the subscription point there.

Ask your WebSphere MQ administrator if a subscription point should be specified and what it should be.

9.8.13 Creating a foreign destination

To create a destination on a foreign bus, do the following:

1. Select **Service integration** → **Buses**.
2. Select the bus on which you want to create a queue.
3. Select **Destinations** in the Destination resources section.
4. Click **New**.
5. Select **Foreign** from the list and click **Next**. Enter the information shown in Figure 9-69.

Create a new foreign destination (a destination on a foreign bus).

→ **Step 1: Set foreign destination attributes**

Step 2: Confirm foreign destination creation

Set foreign destination attributes

Configure the attributes of your new foreign destination

* **Identifier**
MyForeignDest

* **Bus**
QM_itso

Description

Quality of Service

☒ Enable producers to override default reliability

Default reliability
Assured persistent

Maximum reliability
Assured persistent

Next **Cancel**

Figure 9-69 Creating a new foreign destination

– Identifier

Enter the name of the foreign destination for which you want to provide defaults. This must match the name of the destination that exists on the foreign bus.

- Bus

From the drop-down menu, select the foreign bus on which this destination exists. If the foreign bus is not in the list, then select **Other, please specify** and enter the name of the foreign bus in the box.

- Enable producers to override default reliability

If this is selected, it allows applications to specify reliability levels that will override the default reliability setting. If this is set to false, the application's reliability level will be ignored in favor of the default reliability setting.

- Default reliability and Maximum reliability

Select the desired default and maximum reliabilities from the drop-down menus. Consult “Reliability” on page 551.

Click **Next**.

6. Click **Finished**.



Part 3

Working with applications

This part takes you through the process of packaging and deploying applications. In addition, it contains information about concepts that you need to understand to successfully develop and package applications for the WebSphere Application Server V6 run time environment.

This part includes the following chapters:

- ▶ Chapter 10, “Session management” on page 671
- ▶ Chapter 11, “WebSphere naming implementation” on page 741
- ▶ Chapter 12, “Understanding class loaders” on page 795
- ▶ Chapter 13, “Packaging applications” on page 829
- ▶ Chapter 14, “Deploying applications” on page 893

Session management

Session support allows a Web application developer to maintain state information across multiple user visits to the application. In this chapter, we discuss HTTP session support in WebSphere Application Server V6 and how to configure it. We also discuss the new support for stateful session bean failover. The topics include:

- ▶ HTTP session management
- ▶ Session manager configuration
- ▶ Session scope
- ▶ Session identifiers
- ▶ Local sessions
- ▶ General properties for session management
- ▶ Session affinity
- ▶ Persistent session management
- ▶ Invalidating sessions
- ▶ Session security
- ▶ Session performance considerations
- ▶ Stateful session bean failover
- ▶ Session security

10.1 HTTP session management

In many Web applications, users collect data dynamically as they move through the site based on a series of selections on pages they visit. Where the user goes next, and what the application displays as the user's next page, or next choice, depends on what the user has chosen previously from the site. For example, if the user clicks the checkout button on a site, the next page must contain the user's shopping selections.

In order to do this, a Web application needs a mechanism to hold the user's state information over a period of time. However, HTTP alone does not recognize or maintain a user's state. HTTP treats each user request as a discrete, independent interaction.

The Java servlet specification provides a mechanism for servlet applications to maintain a user's state information. This mechanism, known as a *session*, addresses some of the problems of more traditional strategies, such as a pure cookie solution. It allows a Web application developer to maintain all user state information at the host, while passing minimal information back to the user through cookies, or another technique known as *URL rewriting*.

10.2 Session manager configuration

Similar to WebSphere Application Server V5, session management in WebSphere Application Server V6 can be defined at the following levels:

- ▶ Application server
This is the default level. Configuration at this level is applied to all Web modules within the server.
- ▶ Application
Configuration at this level is applied to all Web modules within the application.
- ▶ Web module
Configuration at this level is applied only to that Web module.

10.2.1 Session management properties

With one exception, the session management properties you can set are the same at each configuration level:

- ▶ *Session tracking mechanism* lets you select from cookies, URL rewriting, and SSL ID tracking. Selecting cookies will lead you to a second configuration page containing further configuration options.

- ▶ Select **Maximum in-memory session count** and whether to allow this number to be exceeded, or overflow.
- ▶ *Session timeout* specifies the amount of time to allow a session to remain idle before invalidation.
- ▶ *Security integration* specifies that the user ID be associated with the HTTP session.
- ▶ *Serialize session access* determines if concurrent session access in a given server is allowed.
- ▶ *Overwrite session management*, for enterprise application and Web module level only, determines whether these session management settings are used for the current module, or if the settings are used from the parent object.
- ▶ *Distributed environment settings* determines how to persist sessions (memory-to-memory replication or a database) and set tuning properties. Memory-to-memory persistence is only available in a Network Deployment distributed server environment.

10.2.2 Accessing session management properties

You can access all session management configuration settings using the administrative console.

Application server session management properties

To access session management properties at the application server level, from the administrative console, do the following:

1. Select **Servers** → **Application servers**.
2. Click the application server.
3. In the Container Settings section of the Configuration tab, click **Web Container Settings**.
4. Click **Web Container**. You will see the Web Container setting window.
5. In the Additional Properties section, click **Session management**.

Application session management properties

To access session management properties at the application level, from the administrative console, do the following:

1. Click **Applications** → **Enterprise Applications**.
2. Click the application.
3. In the Web Module Properties section of the Configuration tab, click **Session management**.

Web module session management properties

To access session management properties at the Web module level, from administrative console, do the following:

1. Click **Applications** → **Enterprise Applications**.
2. Click the application.
3. In the Modules section of the Configuration tab, click **Manage Modules**.
4. Click the Web module.
5. In the Additional Properties section, click **Session Management**.

10.3 Session scope

The Servlet 2.4 specification defines session scope at the Web application level. Session information can be accessed only by a single Web application. However, there can be times when there is a logical reason for multiple Web applications to share information, for example, a user name.

WebSphere Application Server provides an IBM extension to the specification allowing session information to be shared among Web applications within an enterprise application. This option is offered as an extension to the application deployment descriptor. No code change is necessary to enable this option. This option is specified during application assembling.

Note: Because session information is shared within the enterprise application, you cannot use the Overwrite Session Management property at the Web module level when the IBM option for shared session context is selected.

Sharing session context

The WebSphere extension for sharing session context is set in the META-INF/ibm-application-ext.xml file in the enterprise project. You can set this using the Application Server Toolkit or from Rational Application Developer:

1. Start the Application Server Toolkit or Rational Application Developer and switch to the J2EE perspective.
2. Double-click the **EAR file** in the J2EE Hierarchy view. This will open the application deployment descriptor.
3. Click the **Overview** tab.
4. Select **Shared session context**. See Figure 10-1 on page 675.

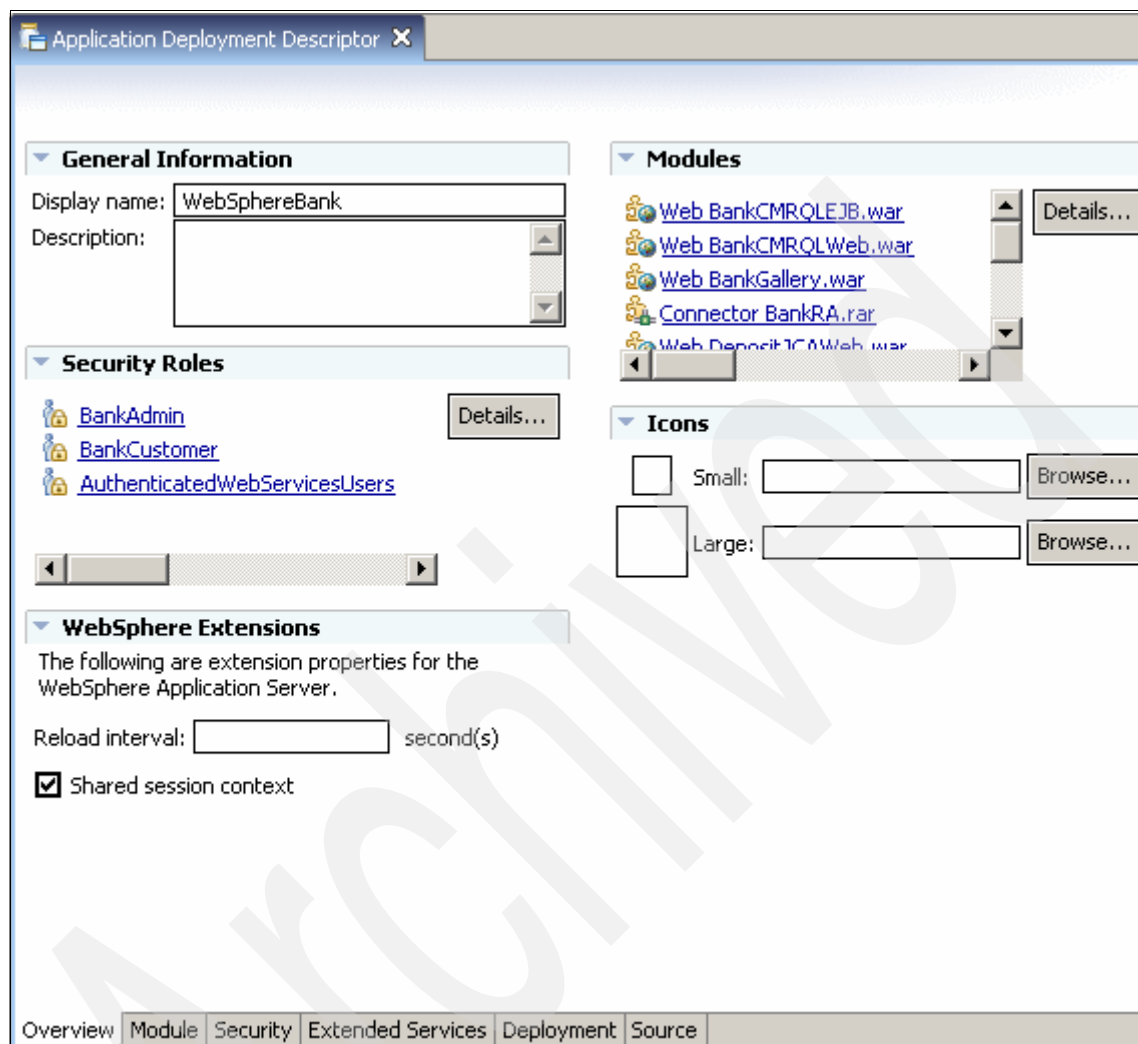


Figure 10-1 Shared HTTP session context using the Application Server Toolkit

5. Save and close the deployment descriptor.

10.4 Session identifiers

WebSphere session support keeps the user's session information about the server. WebSphere passes the user an identifier known as a session ID, which correlates an incoming user request with a session object maintained on the server.

Note: The example session IDs provided in this chapter are for illustrative purposes only and are *not* guaranteed to be absolutely consistent in value, format, and length.

10.4.1 Choosing a session tracking mechanism

WebSphere supports three approaches to tracking sessions:

- ▶ SSL session identifiers
- ▶ Cookies
- ▶ URL rewriting

It is possible to select all three options for a Web application. If you do this:

- ▶ SSL session identifiers are used in preference to cookie and URL rewriting.
- ▶ Cookies are used in preference to URL rewriting.

Note: If SSL session ID tracking is selected, we recommend that you also select cookies or URL rewriting so that session affinity can be maintained. The cookie or rewritten URL contains session affinity information enabling the Web server to properly route a session back to the same server for each request.

To set or change the session mechanism type, do the following:

1. Open the session management properties for the application server, enterprise application, or Web module.
2. Select the session tracking mechanism that you require. See Figure 10-2 on page 677.

Configuration

General Properties

Session tracking mechanism:

☐ Enable SSL ID tracking

☒ [Enable cookies](#)

☐ Enable URL rewriting

☐ Enable protocol switch rewriting

Maximum in-memory session count:

sessions

☒ Allow overflow

Session timeout:

☐ No timeout

☒ Set timeout

minutes

☐ Security integration

Serialize session access:

☐ Allow serial access

Maximum wait time

seconds

☒ Allow access on timeout

Additional Properties

[Distributed environment settings](#)

Figure 10-2 Selecting a session tracking mechanism window

3. Click **OK**.
4. Save and synchronize the configuration changes.
5. Restart the application server or the cluster.

10.4.2 SSL ID tracking

When SSL ID tracking is enabled for requests over SSL, SSL session information is used to track the HTTP session ID.

Because the SSL session ID is negotiated between the Web browser and HTTP server, it cannot survive an HTTP server failure. However, the failure of an application server does not affect the SSL session ID and if the distributed session is not configured, the session itself is lost. In environments that use WebSphere Edge Server with multiple HTTP servers, you must use an affinity mechanism when SSL session ID is used as the session tracking mechanism.

Note: SSL tracking is supported only for the IBM HTTP Server and SUN ONE Web Server.

The lifetime of an SSL session ID can be controlled by configuration options in the Web server. For example, in the IBM HTTP Server, the configuration variable SSLV3TIMEOUT must be set to allow for an adequate lifetime for the SSL session ID. Too short an interval could result in premature termination of a session. Also, some Web browsers might have their own timers that affect the lifetime of the SSL session ID. These Web browsers might not leave the SSL session ID active long enough to be useful as a mechanism for session tracking.

When the SSL session ID is to be used as the session tracking mechanism in a clustered environment, either cookies or URL rewriting must be used to maintain session affinity. The cookie or rewritten URL contains session affinity information that enables the Web server to properly route requests back to the same server once the HTTP session has been created on a server. The SSL ID is not sent in the cookie or rewritten URL but is derived from the SSL information.

Disadvantages of SSL ID tracking

The main disadvantage of using SSL ID tracking is the performance hit of using SSL. If you have a business requirement to use SSL, then this would be a good choice. If you do not have such a requirement, it is probably a good idea to consider using cookies instead.

As discussed previously, Web server and Web browser SSL session timeout settings can also limit the usefulness of SSL ID tracking.

10.4.3 Cookies

Many sites choose cookie support to pass the user's identifier between WebSphere and the user. WebSphere Application Server session support generates a unique session ID for each user, and returns this ID to the user's browser with a cookie. The default name for the session management cookie is JSESSIONID. See Figure 10-3 on page 679.

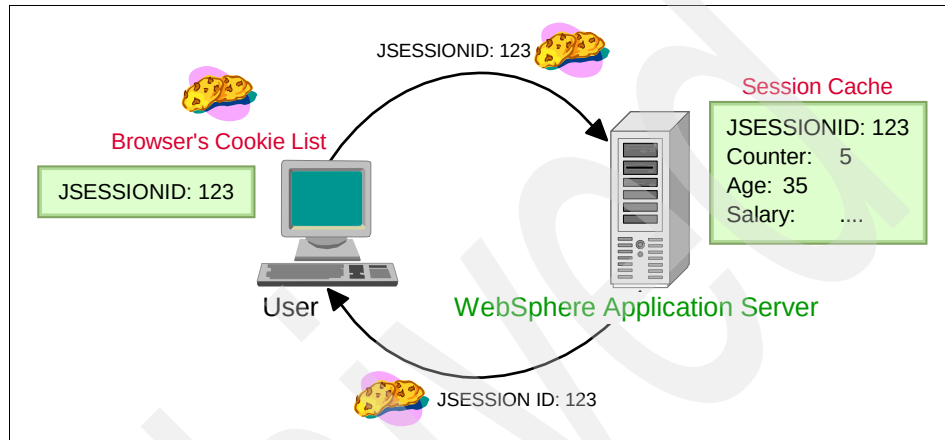


Figure 10-3 Cookie overview

A cookie consists of information embedded as part of the headers in the HTML stream passed between the server and the browser. The browser holds the cookie and returns it to the server whenever the user makes a subsequent request. By default, WebSphere defines its cookies so they are destroyed if the browser is closed. This cookie holds a session identifier. The remainder of the user's session information resides at the server.

The Web application developer uses the HTTP request object's standard interface to obtain the session:

```
HttpSession session = request.getSession(true);
```

WebSphere places the user's session identifier in the outbound cookie whenever the servlet completes its execution, and the HTML response stream returns to the end user. Again, neither the cookie or the session ID within it require any direct manipulation by the Web application. The Web application only sees the contents of the session.

Cookie disadvantages

The main disadvantage with cookies is that some users, either by choice or mandate, disable them from within their browser.

Cookie settings

To configure session management using cookies, do the following from administrative console:

1. Open the Session Manager window at your preferred level.
2. Click the box for **Enable Cookies** as the session tracking mechanism. See Figure 10-4 on page 680.

The screenshot shows the 'Configuration' window for the Session Manager. The 'General Properties' tab is active. Under 'Session tracking mechanism', the 'Enable cookies' checkbox is checked. Below this, the 'Maximum in-memory session count' is set to 1000 sessions, and the 'Allow overflow' checkbox is checked. The 'Session timeout' is set to 30 minutes. The 'Serialize session access' section shows 'Allow serial access' is unchecked, 'Maximum wait time' is 5 seconds, and 'Allow access on timeout' is checked. The 'Additional Properties' tab is also visible, showing a link to 'Distributed environment settings'. At the bottom, there are buttons for 'Apply', 'OK', 'Reset', and 'Cancel'.

Figure 10-4 Session tracking mechanism

3. If you would like to view or change the cookies settings, select the **Enable Cookies** hot link. The following cookie settings are available:

- Cookie name

The cookie name for session management should be unique. The default cookie name is JSESSIONID, which is required by the Servlet 2.4 specification for all cookie-based session IDs. However, this value can be configured for flexibility.

- Restrict cookies to HTTPS sessions

Enabling this feature restricts the exchange of cookies only to HTTPS sessions. If it is enabled, the session cookie's body includes the secure indicator field.

- Cookie domain

This value dictates to the browser whether or not to send a cookie to particular servers. For example, if you specify a particular domain, the browser will only send back session cookies to hosts in that domain. The default value in the session manager restricts cookies to the host that sent them.

Note: The LTPA token/cookie that is sent back to the browser is scoped by a single DNS domain specified when security is configured. This means that *all* application servers in an *entire* WebSphere Application Server domain must share the same DNS domain for security purposes.

- Cookie path

The paths on the server to which the browser will send the session tracking cookie. Specify any string representing a path on the server. Use the slash (/) to indicate the root directory.

Specifying a value restricts the paths to which the cookie will be sent. By restricting paths, you can keep the cookie from being sent to certain URLs on the server. If you specify the root directory, the cookie will be sent no matter which path on the given server is accessed.

- Cookie maximum age

The amount of time that the cookie will live in the client browser. There are two choices:

- Expire at the end of the current browser session
- Expire at a configurable maximum age

If you choose the maximum age option, specify the age in seconds.

4. Click **OK** to exit the page and change your settings.

5. Click **OK** to exit the session management settings.
6. Save and synchronize your configuration changes.
7. Restart the application server or the cluster.

For more information about cookie properties, see *Persistent Client State HTTP Cookies* at:

http://home.netscape.com/newsref/std/cookie_spec.html

10.4.4 URL rewriting

WebSphere also supports URL rewriting for session ID tracking. While session management using SSL IDs or cookies is transparent to the Web application, URL rewriting requires the developer to use special encoding APIs, and to set up the site page flow to avoid losing the encoded information.

URL rewriting works by storing the session identifier in the page returned to the user. WebSphere encodes the session identifier as a parameter on URLs that have been encoded programmatically by the Web application developer. This is an example of a Web page link with URL encoding:

```
<a href="/store/catalog;$jsessionid=DA32242SSGE2">
```

When the user clicks this link to move to the /store/catalog page, the session identifier passes into the request as a parameter.

URL rewriting requires explicit action by the Web application developer. If the servlet returns HTML directly to the requester, without using a JavaServer Page, the servlet calls the API, as shown in Example 10-1, to encode the returning content.

Example 10-1 URL encoding from a servlet

```
out.println("<a href=\"");  
out.println(response.encodeURL ("/store/catalog"));  
out.println("\>catalog</a>");
```

Even pages using redirection, a common practice, particularly with servlet or JSP combinations, must encode the session ID as part of the redirect, as shown in Example 10-2.

Example 10-2 URL encoding with redirection

```
response.sendRedirect(response.encodeRedirectURL("http://myhost/store/catalog")  
);
```

When JavaServer Pages (JSPs) use URL rewriting, the JSP calls a similar interface to encode the session ID:

```
<% response.encodeURL ("/store/catalog"); %>
```

URL rewriting configuration

URL rewriting is selected in the same way as cookies. The only additional configuration option is Enable protocol switch rewriting.

This option defines whether the session ID, added to a URL as part of URL encoding, should be included in the new URL if a switch from HTTP to HTTPS or from HTTPS to HTTP is required. For example, if a servlet is accessed over HTTP and that servlet is doing encoding of HTTPS URLs, URL encoding will be performed only when protocol switch rewriting is enabled, and vice versa.

Disadvantages of using URL rewriting

The fact that the servlet or JSP developer has to write extra code is a major drawback over the other available session tracking mechanisms.

URL rewriting limits the flow of site pages exclusively to dynamically generated pages, such as pages generated by servlets or JSPs. WebSphere inserts the session ID into dynamic pages, but cannot insert the user's session ID into static pages, .htm, or .html.

Therefore, after the application creates the user's session data, the user must visit dynamically generated pages exclusively until they finish with the portion of the site requiring sessions. URL rewriting forces the site designer to plan the user's flow in the site to avoid losing their session ID.

10.5 Local sessions

Many Web applications use the simplest form of session management: the in-memory, local session cache. The local session cache keeps session information in memory and local to the machine and WebSphere Application Server where the session information was first created.

Local session management does not share user session information with other clustered machines. Users only obtain their session information if they return to the machine and WebSphere Application Server holds their session information about subsequent accesses to the Web site.

Most importantly, local session management lacks a persistent store for the sessions it manages. A server failure takes down not only the WebSphere instances running on the server, but also destroys any sessions managed by those instances.

WebSphere allows the administrator to define a limit on the number of sessions held in the in-memory cache from the administrative console settings on the session manager. This prevents the sessions from acquiring too much memory in the Java VM associated with the application server.

The session manager also allows the administrator to permit an unlimited number of sessions in memory. If the administrator enables the **Allow overflow** setting on the session manager, the session manager permits two in-memory caches for session objects. The first cache contains only enough entries to accommodate the session limit defined to the session manager, 1000 by default. The second cache, known as the overflow cache, holds any sessions the first cache cannot accommodate, and is limited in size only by available memory. The session manager builds the first cache for optimized retrieval, while a regular, un-optimized hash table contains the overflow cache.

For best performance, define a primary cache of sufficient size to hold the normal working set of sessions for a given Web application server.

Important: If you enable overflow, the session manager permits an unlimited number of sessions in memory. Without limits, the session caches might consume all available memory in the WebSphere instance's heap, leaving no room to execute Web applications. For example, two scenarios under which this could occur are:

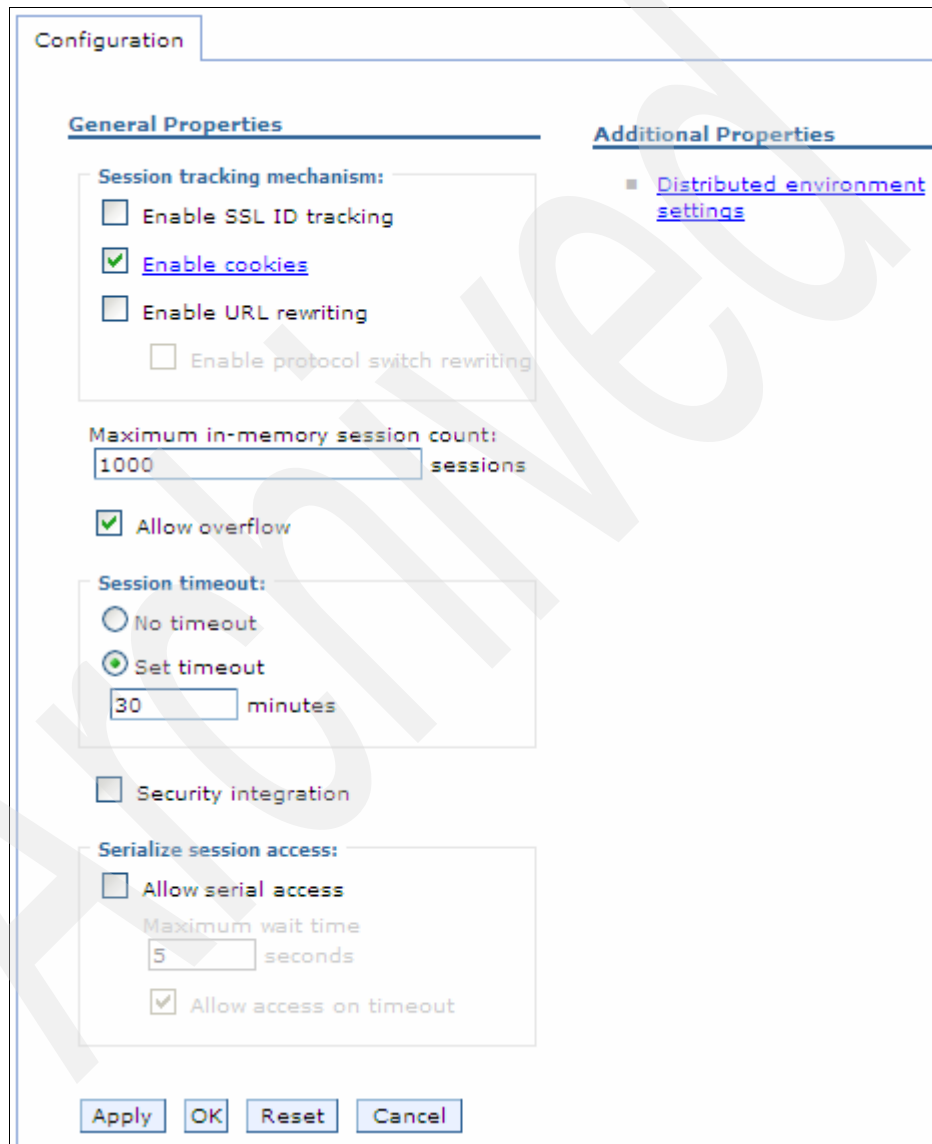
- ▶ The site receives greater traffic than anticipated, generating a large number of sessions held in memory.
- ▶ A malicious attack occurs against the site where a user deliberately manipulates their browser so the application creates a new session repeatedly for the same user.

If you choose to enable session overflow, the state of the session cache should be monitored closely.

Note: Each Web application will have its own base, or primary, in-memory session cache, and with overflow allowed, its own overflow, or secondary, in-memory session cache.

10.6 General properties for session management

The session management settings allow the administrator to tune a number of parameters that are important for both local or persistent sessions. See Figure 10-5 on page 685.



The image shows a 'Configuration' window for session management. It has two tabs: 'General Properties' and 'Additional Properties'. The 'General Properties' tab is active. It contains several settings: 'Session tracking mechanism' with checkboxes for 'Enable SSL ID tracking' (unchecked), 'Enable cookies' (checked), 'Enable URL rewriting' (unchecked), and 'Enable protocol switch rewriting' (unchecked). Below this is 'Maximum in-memory session count' set to '1000 sessions' and 'Allow overflow' (checked). The 'Session timeout' section has radio buttons for 'No timeout' (unchecked) and 'Set timeout' (checked), with a text box for '30 minutes'. There is also a checkbox for 'Security integration' (unchecked). The 'Serialize session access' section has a checkbox for 'Allow serial access' (unchecked), a 'Maximum wait time' of '5 seconds', and a checked checkbox for 'Allow access on timeout'. At the bottom are buttons for 'Apply', 'OK', 'Reset', and 'Cancel'.

Configuration

General Properties

Session tracking mechanism:

- ☐ Enable SSL ID tracking
- ☒ Enable cookies
- ☐ Enable URL rewriting
- ☐ Enable protocol switch rewriting

Maximum in-memory session count:
1000 sessions

☒ Allow overflow

Session timeout:

☐ No timeout

☒ Set timeout
30 minutes

☐ Security integration

Serialize session access:

☐ Allow serial access

Maximum wait time
5 seconds

☒ Allow access on timeout

Apply OK Reset Cancel

Additional Properties

- [Distributed environment settings](#)

Figure 10-5 Session Management configuration window

► Maximum in-memory session count

This field specifies the maximum number of sessions to maintain in memory. The meaning differs depending on whether you are using local or persistent sessions. For local sessions, this value specifies the number of sessions in the base session table. Select **Allow overflow** to specify whether to limit sessions to this number for the entire session manager, or to allow additional sessions to be stored in secondary tables. Before setting this value, see 10.5, “Local sessions” on page 683.

For persistent sessions, this value specifies the size of the general cache. This value determines how many sessions will be cached before the session manager reverts to reading a session from the database automatically. Session manager uses a least recently used (LRU) algorithm to maintain the sessions in the cache.

This value holds when you use local sessions, persistent sessions with caching, or persistent sessions with manual updates. The manual update cache keeps the last n time stamps representing the last access times, where n is the maximum in-memory session count value.

► Allow overflow

Choosing this option specifies whether to allow the number of sessions in memory to exceed the value specified in the maximum in-memory session count field. If **Allow overflow** is not checked, then WebSphere limits the number of sessions held in memory to this value.

For local sessions, if this maximum is exceeded and Allow overflow is not checked, then sessions created thereafter will be dummy sessions and will not be stored in the session manager. Before setting this value, see 10.5, “Local sessions” on page 683.

As shown in Example 10-3, the IBM HttpSession extension can be used to react if sessions exceed the maximum number of sessions specified when overflow is disabled.

Example 10-3 Using IBMSession to react to session overflow

```
com.ibm.websphere.servlet.session.IBMSession sess =  
    (com.ibm.websphere.servlet.session.IBMSession) req.getSession(true);  
if(sess.isOverflow()) {  
    //Direct to a error page...  
}
```

Note: Allowing an unlimited amount of sessions can potentially exhaust system memory and even allow for system sabotage. Someone could write a malicious program that continually hits your site, creating sessions, but ignoring any cookies or encoded URLs and never utilizing the same session from one HTTP request to the next.

► Session timeout

If you select **Set timeout**, when a session is not accessed for this many minutes it can be removed from the in-memory cache and, if persistent sessions are used, from the persistent store. This is important for performance tuning. It directly influences the amount of memory consumed by the JVM in order to cache the session information.

Note: For performance reasons, the session manager invalidation process runs at regular intervals to invalidate any invalid sessions. This interval is determined internally based on the Session timeout interval specified in the Session manager properties. For the default timeout value of 30 minutes, the invalidation process interval is around 300 seconds. In this case, it could take up to 5 minutes (300 seconds) beyond the timeout threshold of 30 minutes for a particular session to become invalidated.

The value of this setting is used as a default when the session timeout is not specified in a Web module's deployment descriptor.

If you select **No timeout**, a session will be never removed from the memory unless explicit invalidation has been performed by the servlet. This can cause a memory leak when the user closes the window without logging out from the system. This option might be useful when sessions should be kept for a while until explicit invalidation has been done, when an employee leaves the company, for example. To use this option, make sure that enough memory or space in a persistent store is kept to accommodate all sessions.

► Security integration

When security integration is enabled, the session manager associates the identity of users with their HTTP sessions. See 10.10, "Session security" on page 725 for more information.

Note: Do not enable this property if the application server contains a Web application that has form-based login configured as the authentication method and the local operating system is the authentication mechanism. It will cause authorization failures when users try to use the Web application.

- **Serialize session access**

In WebSphere V4, sessions could be accessed concurrently, meaning multiple threads could access the same session at the same time. It was the programmer's responsibility to serialize access to the session to avoid inconsistencies.

In WebSphere V5 and WebSphere V6, this option is available to provide serialized access to the session in a given JVM. This ensures thread-safe access when the session is accessed by multiple threads. No special code is necessary for using this option. This option is not recommended when framesets are used heavily because it can affect performance.

An optional property, Maximum wait time, can be set to specify the maximum amount of time a servlet request waits on an HTTP session before continuing execution. The default is two minutes.

If you set the Allow access on timeout option, multiple servlet requests that have timed out concurrently will execute normally. If it is false, servlet execution aborts.

10.7 Session affinity

The Servlet 2.4 specification requires that an HTTP session be:

- **Accessible only to the Web application that created the session**

The session ID, but not the session data, can be shared across Web applications.

- **Handled by a single JVM for that application at any one time**

In a clustered environment, any HTTP requests associated with an HTTP session must be routed to the same Web application in the same JVM. This ensures that all of the HTTP requests are processed with a consistent view of the user's HTTP session. The exception to this rule is when the cluster member fails or has to be shut down.

WebSphere is able to assure that session affinity is maintained in the following way: Each server ID is appended to the session ID. When an HTTP session is created, its ID is passed back to the browser as part of a cookie or URL encoding. When the browser makes further requests, the cookie or URL encoding will be sent back to the Web server. The Web server plug-in examines the HTTP session ID in the cookie or URL encoding, extracts the unique ID of the cluster member handling the session, and forwards the request.

This can be seen in Figure 10-6 on page 689, where the session ID from the HTTP header, `request.getHeader("Cookie")`, is displayed along with the session ID from `session.getId()`. The application server ID is appended to the session ID from the HTTP header. The first four characters of HTTP header session ID are the cache identifier that determines the validity of cache entries.

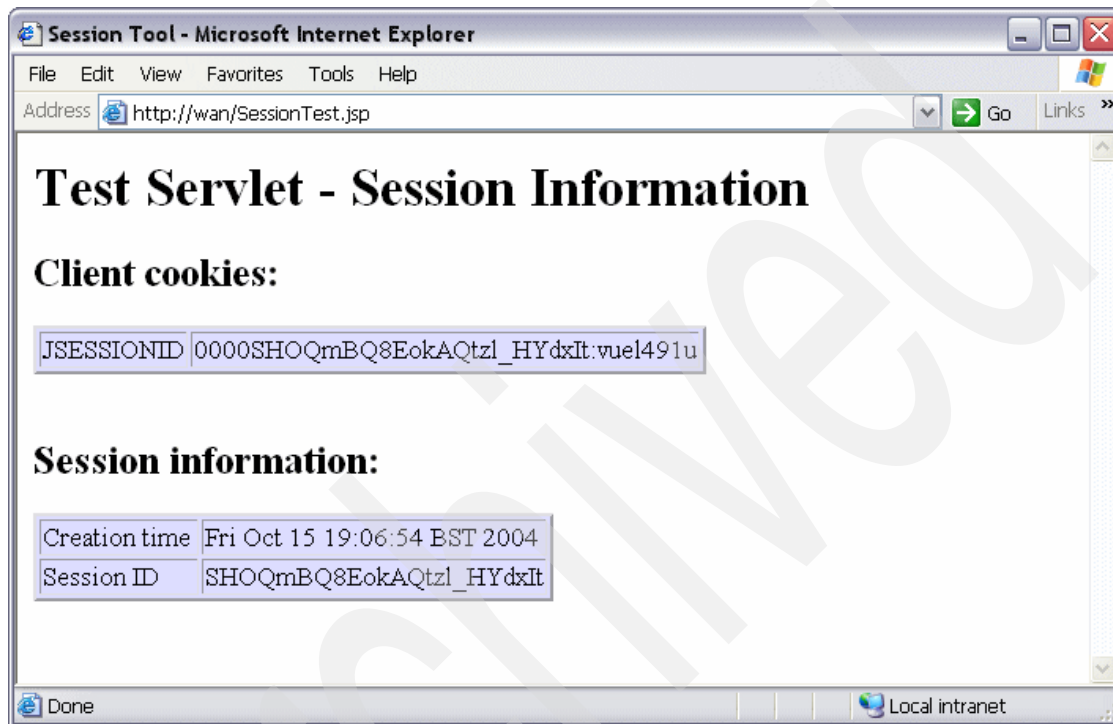


Figure 10-6 Session ID containing the server ID and cache ID

The JSESSIONID cookie can be divided into four parts: cache ID, session ID, separator, clone ID, and partition ID (new in V6). JSESSION ID will include a partition ID instead of a clone ID when memory-to-memory replication in peer-to-peer mode is selected. Typically, the partition ID is a long numeric number.

Table 10-1 shows their mappings based on the example in Figure 10-6. A clone ID is an ID of a cluster member.

Table 10-1 Cookie mapping

content	value in the example
Cache ID	0000
Session ID	SHOQmBQ8EokAQzI_HYdxIt
separator	:
Clone ID	vuel491u

The application server ID can be seen in the Web server plug-in configuration file, plug-in-cfg.xml file, as shown in Example 10-4.

Example 10-4 Server ID from plugin-cfg.xml file

```
<?xml version="1.0" encoding="ISO-8859-1"?><!--HTTP server plugin config file
for the cell ITS0Cell generated on 2004.10.15 at 07:21:03 PM BST-->
<Config>
.....
  <ServerCluster Name="MyCluster">
    <Server CloneID="vuel491u" LoadBalanceWeight="2" Name="NodeA_server1">
      <Transport Hostname="wan" Port="9080" Protocol="http"/>
      <Transport Hostname="wan" Port="9443" Protocol="https">
.....
    </Server>
  </ServerCluster>
</Config>
```

Note: Session affinity can still be broken if the cluster member handling the request fails. To avoid losing session data, use persistent session management. In persistent sessions mode, cache ID and server ID will change in the cookie when there is a failover or when the session is read from the persistent store, so do not rely on the value of the session cookie remaining the same for a given session.

10.7.1 Session affinity and failover

Server clusters provide a solution for failure of an application server. Sessions created by cluster members in the server cluster share a common persistent session store. Therefore, any cluster member in the server cluster has the ability to see any user's session saved to persistent storage. If one of the cluster members fail, the user can continue to use session information from another cluster member in the server cluster. This is known as failover. Failover works regardless of whether the nodes reside on the same machine or several machines. See Figure 10-7 on page 691.

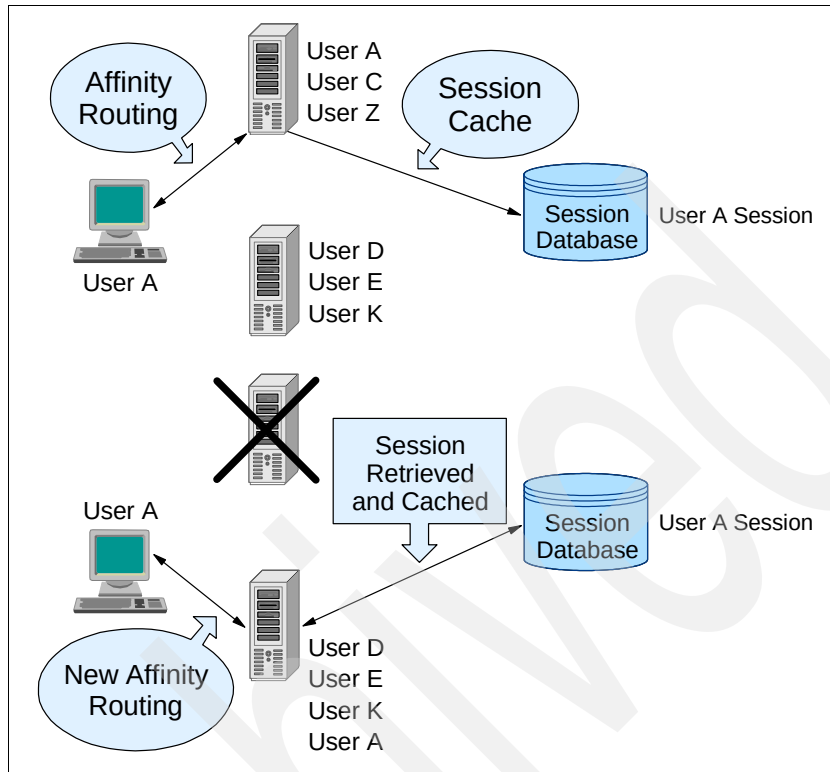


Figure 10-7 Session affinity and failover

Note: According to the Servlet 2.4 specification, only a single cluster member can control and access a given session at a time.

After a failure, WebSphere redirects the user to another cluster member, and the user's session affinity switches to this replacement cluster member. After the initial read from the persistent store, the replacement cluster member places the user's session object in the in-memory cache, assuming the cache has space available for additional entries.

The Web server plug-in maintains the cluster member list in order and picks the cluster member next in its list to avoid the breaking of session affinity. From then on, requests for that session go to the selected cluster member. The requests for the session go back to the failed cluster member when the failed cluster member restarts.

WebSphere provides session affinity on a best-effort basis. There are narrow windows where session affinity fails. These windows are:

- ▶ When a cluster member is recovering from a crash, a window exists where concurrent requests for the same session could end up in different cluster members. The reason for this is the Web server is multi-processed and each process separately maintains its own retry timer value and list of available cluster members. The end result is that requests being processed by different processes might end up being sent to more than one cluster member after at least one process has determined that the failed cluster member is running again.

To avoid or limit exposure in this scenario, if your cluster members are expected to crash very seldom and are expected to recover fairly quickly, consider setting the retry timeout to a small value. This narrows the window during which multiple requests being handled by different processes get routed to multiple cluster members.

- ▶ A server overload can cause requests belonging to the same session to go to different cluster members. This can occur even if all the cluster members are running. For each cluster member, there is a backlog queue where an entry is made for each request sent by the Web server plug-in waiting to be picked up by a worker thread in the servlet engine. If the depth of this queue is exceeded, the Web server plug-in starts receiving responses that the cluster member is not available. This failure is handled in the same way by the Web server plug-in as an actual JVM crash. Examples of when this can happen are:
 - The servlet engine does not have an appropriate number of threads to handle the user load.
 - The servlet engine threads take a long time to process the requests. Reasons for this include: applications taking a long time to execute, resources being used by applications taking a long time, and so on.

10.8 Persistent session management

By default, WebSphere places session objects in memory. However, the administrator has the option of enabling persistent session management, which instructs WebSphere to place session objects in a persistent store.

Administrators should enable persistent session management when:

- ▶ The user's session data must be recovered by another cluster member after a cluster member in a cluster fails or is shut down.
- ▶ The user's session data is too valuable to lose through unexpected failure at the WebSphere node.

- ▶ The administrator desires better control of the session cache memory footprint. By sending cache overflow to a persistent session store, the administrator controls the number of sessions allowed in memory at any given time.

There are two ways to configure session persistence in WebSphere Application Server V6, as in Figure 10-8:

- ▶ Database persistence
- ▶ Memory-to-memory session state replication using the data replication service available in distributed server environments

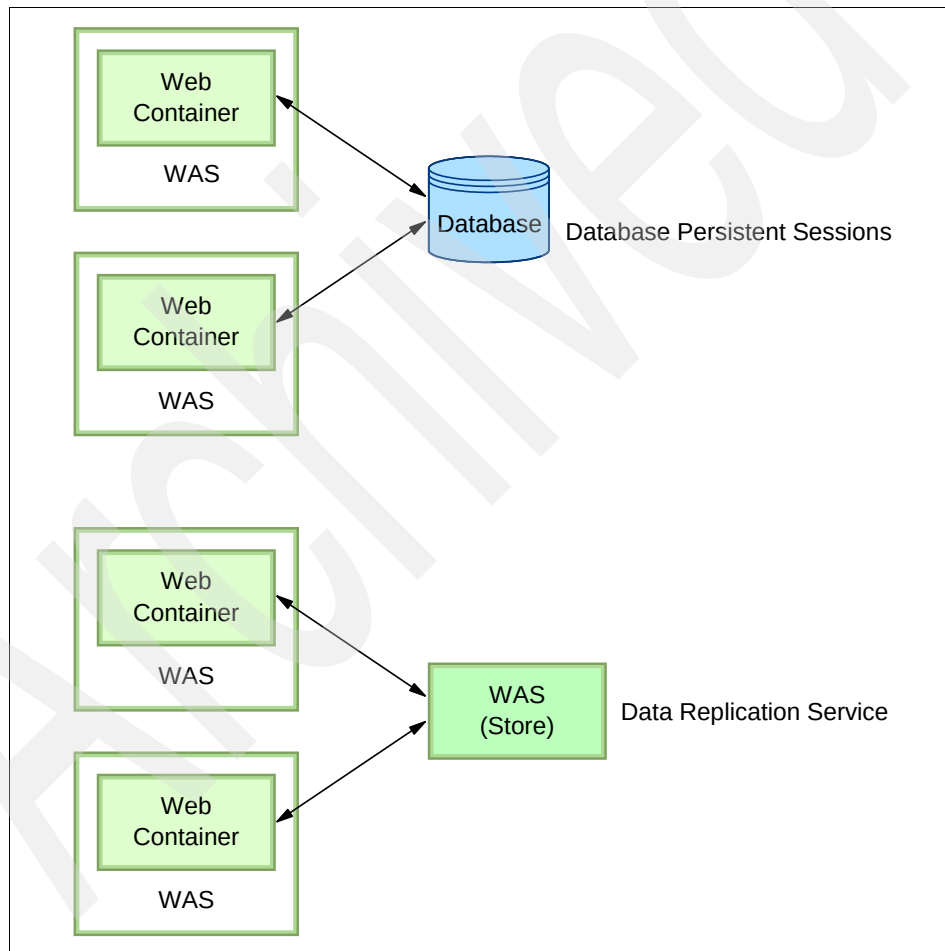


Figure 10-8 Persistent session options

All information stored in a persistent session store must be serialized. As a result, all of the objects held by a session must implement `java.io.Serializable` if the session needs to be stored in a persistent session store.

In general, consider making all objects held by a session serialized, even if immediate plans do not call for the use of persistent session management. If the Web site grows, and persistent session management becomes necessary, the transition between local and persistent management occurs transparently to the application if the sessions only hold serialized objects. If not, a switch to persistent session management requires coding changes to make the session contents serialized.

Persistent session management does not impact the session API, and Web applications require no API changes to support persistent session management. However, as mentioned previously, applications storing unserializable objects in their sessions require modification before switching to persistent session management.

If you use database persistence, using multi-row sessions becomes important if the size of the session object exceeds the size for a row, as permitted by the WebSphere session manager. If the administrator requests multi-row session support, the WebSphere session manager breaks the session data across multiple rows as needed. This allows WebSphere to support large session objects. Also, this provides a more efficient mechanism for storing and retrieving session contents under certain circumstances. See 10.8.6, “Single and multi-row schemas (database persistence)” on page 717 for information about this feature.

Using a cache lets the session manager maintain a cache of most recently used sessions in memory. Retrieving a user session from the cache eliminates a more expensive retrieval from the persistent store. The session manager uses a *least recently used* scheme for removing objects from the cache. Session data is stored to the persistent store based on your selections for write frequency and write option.

10.8.1 Enabling database persistence

It is assumed in this section that the following tasks have already completed before enabling database persistence:

1. Create a session database. In this example, it is assumed that the data source JNDI name is `jdbc/Sessions`.
2. (z/OS DB2) Create a table for the session data. Name the table `SESSIONS`. If you choose to use another name, update the Web container custom property `SessionTableName` value to the new table name. Grant ALL authority for the

server region user ID to the table. An example of creating the table can be found in the Information Center at:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.zseries.doc/info/zseries/ae/tprs_db2tzos.html

In distributed environments, the session table will be created automatically for you when you define the data source for the database as the session management table; however, if you want to use a page (row) size greater than 4 KB, you will need to create the tablespace manually. An example of creating the tablespace can be found in the Information Center at:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/tprs_db2t.html

3. Create a JDBC provider and data source for the database. The data source should be non-XA enabled. See 6.2, “JDBC resources” on page 305 and 6.2.3, “Creating a data source” on page 311.

Note: The following example illustrates the steps to enable database persistence at the application server level. Session management settings can also be performed at the enterprise application level and the Web application level.

To enable database persistence, repeat the following steps for each application server:

1. Select **Servers** → **Application servers**.
2. Select the server.
3. Click **Session management** under Web container in the Additional Properties section.
4. Click **Distributed environment settings**.

5. Select **Database** and click **Database**. See Figure 10-9.

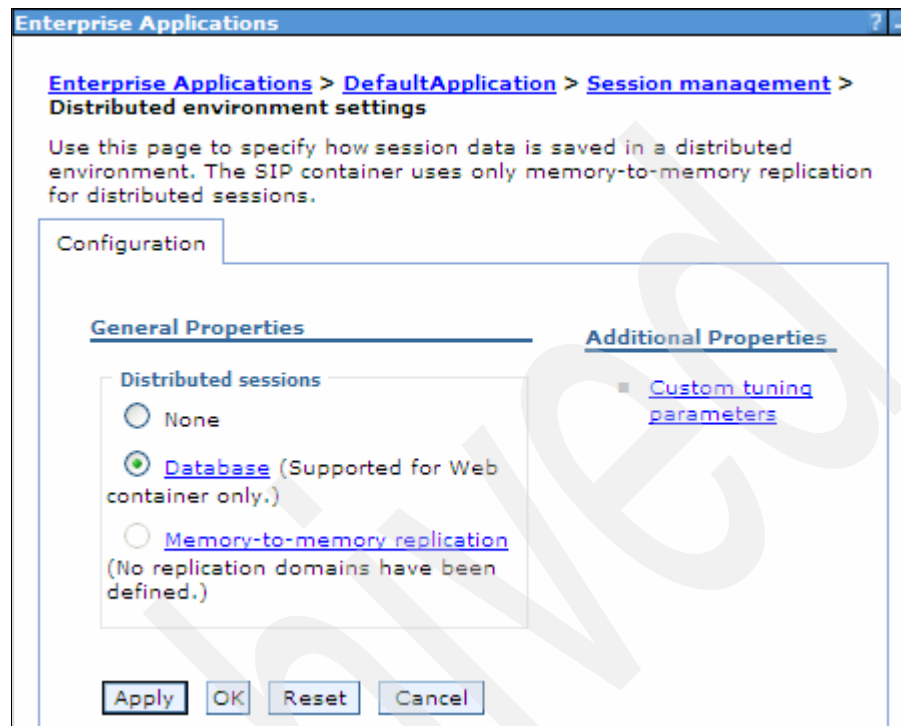


Figure 10-9 Distributed Environment Setting (database)

6. Enter the database information:
- Enter the data source JNDI name. The data source must be a non-JTA enabled data source.
 - Enter the user ID and password to access the database.
 - If you are using DB2 and you anticipate requiring row sizes greater than 4 KB, select the appropriate value from the DB2 row size menu. See 10.8.5, "Larger DB2 page sizes and database persistence" on page 716 for more information.
 - If DB2 row size is other than 4 KB, you are required to enter the name of tablespace. See "Larger DB2 page sizes and database persistence" on page 716.
 - If you intend to use a multi-row schema, select **Use Multi row schema**. See 10.8.6, "Single and multi-row schemas (database persistence)" on page 717 for more information. See Figure 10-10.

The screenshot shows a web-based configuration interface for 'Enterprise Applications'. The breadcrumb navigation at the top reads: [Enterprise Applications](#) > [DefaultApplication](#) > [Session management](#) > [Distributed environment settings](#) > **Database settings**. Below the navigation, a message states: 'Use this page to specify your database settings.' The main configuration area is titled 'Configuration' and contains a section for 'General Properties'. This section includes the following fields and controls:

- * Datasource JNDI name:** A text input field.
- User ID:** A text input field.
- Password:** A text input field.
- DB2 row size:** A dropdown menu currently showing 'ROW_SIZE_4KB'.
- Table space name:** A text input field.
- ☐ **Use multi row schema**

At the bottom of the configuration area are four buttons: 'Apply', 'OK', 'Reset', and 'Cancel'.

Figure 10-10 Database settings for session persistence

Note: For z/OS implementations, only the Datasource JNDI name and Use multi row schema settings will appear in the window.

7. Click **OK**.

After you have updated each server, save the configuration changes, synchronize them with the servers, and restart the application servers.

10.8.2 Memory-to-memory replication

Memory-to-memory replication uses the data replication service to replicate data across many application servers in a cluster without using a database. Using this method, sessions are stored in the memory of an application server, providing the same functionality as a database for session persistence. Separate threads handle this functionality within an existing application server process.

The data replication service is an internal WebSphere Application Server component. In addition to its use by the session manager, it is also used to replicate dynamic cache data and stateful session beans across many application servers in a cluster.

The advantages of using this method of session persistence are:

- ▶ Flexible configuration options, such as peer-peer and client/server
- ▶ Elimination of the overhead and cost of setting up and maintaining a real-time production database.
- ▶ Elimination of single point of failure that can occur with a database.
- ▶ Encrypted session information between application servers.

Version 5.X versus Version 6 data replication service

The following changes have been made to the data replication service in WebSphere Application Server V6:

- ▶ Simplified configuration

Many fields from V5.X have been deprecated and the configuration windows are now more intuitive.
- ▶ Terminology changes

Some of the terms from V5.X have been deprecated to reflect the new topologies and configuration needs. The terms *replicas* and *partitions* have been removed. In V6, we have *client* and *servers* in a replication domain.
- ▶ Topology changes

Partitions from V5.X have been deprecated. Replication domain still exists, but defined differently. It is no longer a collection of replicas. You can select the replication mode of server, client, or both when configuring the session management facility for memory-to-memory replication in WebSphere Application Server V6. The default is both.
- ▶ Integration with workload management to provide hot failover in peer-to-peer mode
- ▶ Ability to collocate stateful session EJB replicas with HTTP session replicas in hot failover

V5.X `wsadmin` DRS scripts continue to work with V6.

Data replication service modes

The memory-to-memory replication function is accomplished by the creation of a data replication service instance in an application server that communicates to other data replication service instances in remote application servers.

There are three possible modes you can set up a replication service instance to run in:

- ▶ **Server mode**
In this mode, a server only stores backup copies of other application server sessions. It does not send copies of sessions created in that particular server.
- ▶ **Client mode**
In this mode, a server only broadcasts or sends copies of the sessions it owns. It does not receive backup copies of sessions from other servers
- ▶ **Both mode**
In this mode, the server simultaneously sends copies of the sessions it owns, and acts as a backup table for sessions owned by other application servers.

You can select the replication mode of server, client, or both when configuring the session management facility for memory-to-memory replication. The default is both.

With respect to mode, the following are the primary examples of memory-to-memory replication configuration:

- ▶ Peer-to-peer replication
- ▶ Client/server replication

Although the administrative console allows flexibility and additional possibilities for memory-to-memory replication configuration, only these configurations are officially supported.

There is a single replica in a cluster by default. You can modify the number of replicas through the replication domain.

Peer-to-peer topology

Figure 10-11 on page 700 shows an example of peer-to-peer topology. Each application server stores sessions in its own memory. It also stores sessions to and retrieves sessions from other application servers. Each application server acts as a client by retrieving sessions from other application servers. Each application server acts as a server by providing sessions to other application servers.

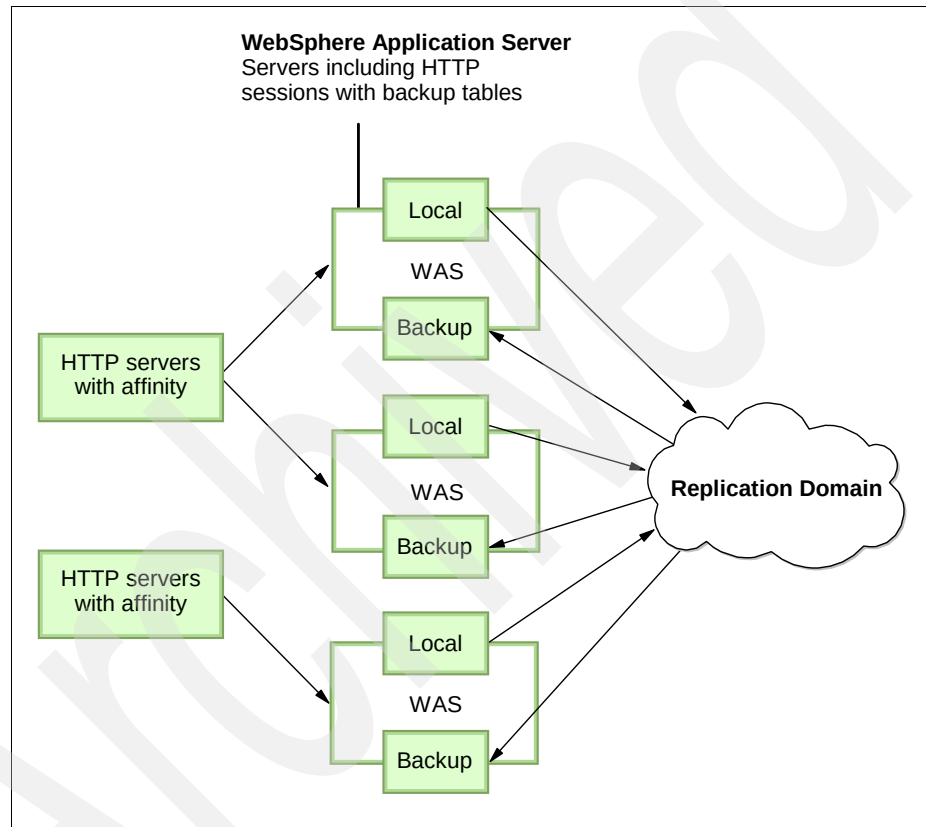


Figure 10-11 Example of peer-to-peer topology

The basic peer-to-peer (both mode) topology is the default configuration and has a single replica. However, you can also add additional replicas by configuring the replication domain.

In this basic peer-to-peer topology, each application server can:

- ▶ Host the Web application leveraging the HTTP session
- ▶ Send changes to the HTTP session that it owns
- ▶ Receive backup copies of the HTTP session from all of the other servers in the cluster

This configuration represents the most consolidated topology, where the various system parts are collocated and requires the fewest server processes. When using this configuration, the most stable implementation is achieved when each node has equal capabilities (CPU, memory, and so on), and each handles the same amount of work.

The advantage of this topology is that no additional processes and products are required to avoid a single point of failure. This reduces the time and cost required to configure and maintain additional processes or products.

One of the disadvantages of this topology is that it can consume large amounts of memory in networks with many users, because each server has a copy of all sessions. For example, assuming that a single session consumes 10 KB and one million users have logged into the system, each application server consumes 10 GB of memory in order to keep all sessions in its own memory. Another disadvantage is that every change to a session must be replicated to all application servers. This can cause a performance impact.

Client/server topology

Figure 10-12 on page 702 shows an example of client/server topology. In this setup, application servers act as either a replication client or a server. Those that act as replication servers store sessions in their own memory and provide session information to clients. They are dedicated replication servers that just store sessions but do not respond to the users' requests. Client application servers send session information to the replication servers and retrieve sessions from the servers. They respond to user requests and store only the sessions of the users with whom they interact.

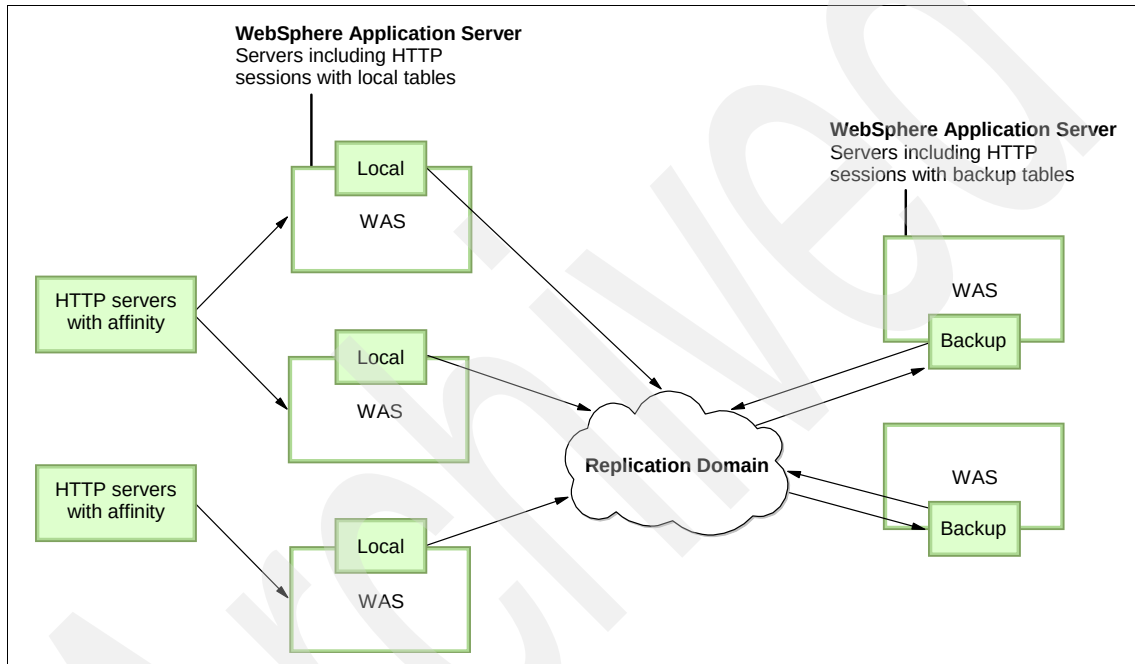


Figure 10-12 Example of client/server topology

The advantage of this topology is that it clearly distinguishes the role of client and server. Only replication servers keep all sessions in their memory and only the clients interact with users. This reduces the consumption of memory on each application server and reduces the performance impact, because session information is only sent to the servers.

You can recycle a backup server without affecting the servers running the application. When there are two or more backups, failure recovery is possible. Conversely, you can recycle an application server without losing the backup data.

When running Web applications on lower-end hardware, you can choose to have one or two more powerful computers that have the capacity to run a couple of

session managers in replication server mode, allowing you to reduce the load on the Web application hardware.

One of the disadvantages of this topology is that additional application servers have to be configured and maintained over and above those that interact with users. We recommended that you have multiple replication servers configured to avoid a single point of failure.

Replication domain

The memory-to-memory replication function is accomplished by the creation of a data replication service instance in an application server that communicates to other data replication service instances in remote application servers. You must configure this data replication service instance as a part of a replication domain.

Data replication service instances on disparate application servers that replicate to one another must be configured as a part of the same domain. You must configure all session managers connected to a replication domain to have the same topology. If one session manager instance in a domain is configured to use the client/server topology, then the rest of the session manager instances in that domain must be a combination of servers configured as Client only and Server only.

If one session manager instance is configured to use the peer-to-peer topology, then all session manager instances must be configured as both client and server. For example, a server-only data replication service instance and a both client and server data replication service instance cannot exist in the same replication domain. Multiple data replication service instances that exist on the same application server due to session manager memory-to-memory configuration at various levels that are configured to be part of the same domain must have the same mode.

You should create a separate replication domain for each consumer. For example, create one replication domain for session manager and another replication domain for dynamic cache.

The only situation where you should configure one replication domain is when you configure session manager replication and stateful session bean failover. Using one replication domain in this case ensures that the backup state information of HTTP sessions and stateful session beans are on the same application servers.

Note: A replication domain created with WebSphere Application Server V5.X is referred to as a *multi-broker domain*. This type of replication domain consists of replicator entries. This is deprecated in WebSphere Application Server V6 and supported only for backward compatibility. Multi-broker replication domains do not communicate with each other, so migrate any multi-broker replication domains to the new data replication domains. You cannot create a multi-broker domain or replicator entries in the administrative console of WebSphere Application Server V6.

Enabling memory-to-memory replication

It is assumed in this section that the following tasks have already been completed before enabling data for the replication service:

1. You have created a cluster consisting of at least two application servers.
In this example, we are working with a cluster called MyCluster. It has two servers, server1 and server2.
2. You have installed applications to the cluster.

Note: This example illustrates setting up the replication domain and replicators after the cluster has been created. You also have the option of creating the replication domain and the replicator in the first server in the cluster when you create the cluster.

To enable memory-to-memory replication, do the following:

1. Create a replication domain to define the set of replicator processes that communicate with each other.
 - a. Select **Environment** → **Replication domains**. Click **New**. See Figure 10-13 on page 705, and enter information in the fields.

Configuration

General Properties

* Name
MyClusterRepDomain

* Request timeout
5 seconds

Encryption

Encryption type
none

Regenerate encryption key

Number of replicas

☒ Single replica
☐ Entire Domain
☐ Specify

Number of replicas

Apply OK Reset Cancel

Figure 10-13 Create a replication domain

- Name

At a minimum, you need to enter a name for the replication domain. The name must be unique within the cell. In this example, we used MyClusterRepDomain as the name, and defaults are used for the other properties.

- Encryption

Encrypted transmission achieves better security but can impact performance. If DES or TRIPLE_DES is specified, a key for data transmission is generated. We recommend that you generate a key by clicking the **Regenerate encryption key** button periodically to enhance security.

- Number of replicas

A single replica allows you to replicate a session to only one other server. This is the default. When you choose this option, a session manager picks another session manager connected to the same replication domain to which to replicate the HTTP session during session creation. All updates to the session are only replicated to that single server. This option is set at the replication domain level. When this option is set, every session manager connected to this replication domain creates a single backup copy of HTTP session state information about a backup server.

Alternatively, you can replicate to every application server that is configured as a consumer of the replication domain with the Entire Domain option or to a specified number of replicas within the domain.

- b. Click **Apply**.
 - c. Click **OK**.
 - d. Save the configuration changes.
2. Configure cluster members.
- Repeat the following steps for each application server:
- a. Select **Servers** → **Application servers**.
 - b. Click the application server name. In this example, **server1** and **server2** are selected as application servers respectively.
 - c. Click **Web container** in the Container settings section.
 - d. Click **Session management**.
 - e. Click **Distributed environment settings**.
 - f. Select **Memory-to-memory replication**. See Figure 10-14 on page 707.

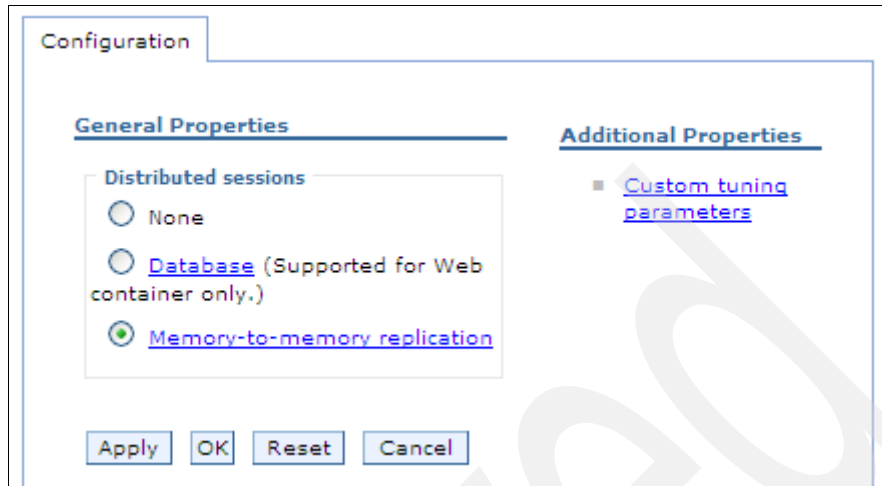


Figure 10-14 Distributed environment settings

- g. Choose a replicator domain and replicator mode either from listed domains. See Figure 10-15 on page 707.

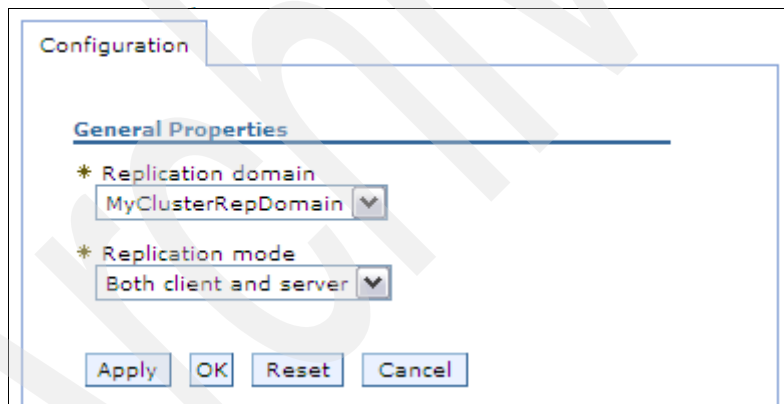


Figure 10-15 Data replication service settings

Select the replication topology by specifying the replication mode. Selecting **Both client and server** identifies this as a peer-to-peer topology. In a client/server topology, select **Client only** for application servers that will be responding to user requests. Select **Server only** for those that will be used as replication servers.

- h. Click **OK**.

3. Save the configuration and restart the cluster. You can restart the cluster by selecting **Servers** → **Clusters**. Check the cluster, and click **Stop**. After the messages indicate the cluster has stopped, click **Start**.

Configuration file results

The replication domain configuration is written to
<profile_home>/config/cells/<cell>/multibroker.xml.

See Example 10-5 on page 708.

Example 10-5 Replication domain configuration in multibroker.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<multibroker:DataReplicationDomain xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:multibroker="http://www.ibm.com/websphere/appserver/schemas/5.0/multibroker.xmi" xmi:id="DataReplicationDomain_1098124985651" name="MyClusterRepDomain">
  <defaultDataReplicationSettings xmi:id="DataReplication_1098124985651"
requestTimeout="5" encryptionType="NONE" numberOfReplicas="1">
    <partition xmi:id="DRSPartition_1098124985651" size="10"
partitionOnEntry="false"/>
    <serialization xmi:id="DRSSerialization_1098124985651"
entrySerializationKind="BYTES" propertySerializationKind="BYTES"/>
    <pooling xmi:id="DRSConnectionPool_1098124985651" size="10"
poolConnections="false"/>
  </defaultDataReplicationSettings>
</multibroker:DataReplicationDomain>
```

Configuring an application server to use a replication domain for session persistence updates the server.xml file:

<profile_home>/config/cells/<cell>/nodes/<node>/servers/<server>/server.xml

See Example 10-6.

Example 10-6 Server.xml updates

```
<sessionDRSPersistence xmi:id="DRSSettings_1097867741921"
messageBrokerDomainName="MyClusterRepDomain"/>
```

10.8.3 Session management tuning

Performance tuning for session management persistence consists of defining the following:

- ▶ How often session data is written (write frequency settings).
- ▶ How much data is written (write contents settings).

- ▶ When the invalid sessions are cleaned up (session cleanup settings).

These settings are set in the Custom tuning parameters found under the Additional properties section for session management settings. Several combinations of these settings are predefined and available for selection, or you can customize them.

Writing frequency settings

You can select from three different settings that determine how often session data is written to the persistent data store:

- ▶ End of servlet service
If the session data has changed, it will be written to the persistent store after the servlet finishes processing an HTTP request.
- ▶ Manual update
The session data will be written to the persistent store when the `sync()` method is called on the `IBMSession` object.
- ▶ Time-based
The session data will be written to the persistent store based on the specified write interval value.

Note: The last access time attribute is always updated each time the session is accessed by the servlet or JSP, whether the session is changed or not. This is done to make sure the session does not time out.

- ▶ If you choose the end of servlet service option, each servlet or JSP access will result in a corresponding persistent store update of the last access time.
- ▶ If you select the manual update option, the update of the last access time in persistent store occurs on `sync()` call or at later time.
- ▶ If you use time-based updates, the changes are accumulated and written in a single transaction. This can significantly reduce the amount of I/O to the persistent store.

See 10.11.2, “Reducing persistent store I/O” on page 730 for options to change this database update behavior.

Consider an example where the Web browser accesses the application once every five seconds:

- ▶ In End of servlet service mode, the session would be written out every five seconds.

- ▶ In Manual update mode, the session would be written out whenever the servlet issues `IBMSession.sync()`. It is the responsibility of the servlet writer to use the `IBMSession` interface instead of the `HttpSession` Interface and the servlets/JSPs must be updated to issue the `sync()`.
- ▶ In Time-based mode, the servlet or JSP need not use the `IBMSession` class nor issue `IBMSession.sync()`. If the write interval is set to 120 seconds, then the session data is written out at most every 120 seconds.

End of servlet service

When the write frequency is set to the end of servlet service option, WebSphere writes the session data to the persistent store at the completion of the `HttpServlet.service()` method call. The write content settings determine output.

Manual update

In manual update mode, the session manager only sends changes to the persistent data store if the application explicitly requests a save of the session information.

Note: Manual updates use an IBM extension to `HttpSession` that is not part of the Servlet 2.4 API.

Manual update mode requires an application developer to use the `IBMSession` class for managing sessions. When the application invokes the `sync()` method, the session manager writes the modified session data and last access time to the persistent store. The session data written to the persistent store is controlled by the write contents option selected.

If the servlet or JSP terminates without invoking the `sync()` method, the session manager saves the contents of the session object into the session cache (if caching is enabled), but does not update the modified session data in the session database. The session manager will only update the last access time in the persistent store asynchronously, at later time. Example 10-7 shows how the `IBMSession` class can be used to manually update the persistent store.

Example 10-7 Using IBMSession for manual update of the persistent store

```
public void service (HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
{
    // Use the IBMSession to hold the session information
    // We need the IBMSession object because it has the manual update
    // method sync()
    com.ibm.websphere.servlet.session.IBMSession session =
        (com.ibm.websphere.servlet.session.IBMSession)req.getSession(true);

    Integer value = 1;
```

```
//Update the in-memory session stored in the cache
session.putValue("MyManualCount.COUNTER", value);

//The servlet saves the session to the persistent store
session.sync();
}
```

This interface gives the Web application developer additional control of when and if session objects go to the persistent data store. If the application does not invoke the `sync()` method, and manual update mode is specified, the session updates go only to the local session cache, not the persistent data store. Web developers use this interface to reduce unnecessary writes to the session database, and thereby to improve overall application performance.

All servlets in the Web application server must perform their own session management in manual update mode.

Time-based writes to the session database

Using the time-based write option will write session data to the persistent store at a defined write interval. The reasons for implementing time-based write lies in the changes introduced with the Servlet 2.2 API. The Servlet 2.2 specification introduced two key concepts:

- ▶ It limits the scope of a session to a single Web application.
- ▶ It both explicitly prohibits concurrent access to an `HttpSession` from separate Web applications, and allows for concurrent access within a given JVM.

Because of these changes, WebSphere provides the session affinity mechanism that assures an HTTP request is routed to the Web application handling its `HttpSession`. This assurance still holds in a WLM environment when using persistent `HttpSessions`. This means that the necessity to immediately write the session data to the persistent store can now be relaxed somewhat in these environments, as well as non-clustered environments, because the persistent store is used now only for failover and session cache full scenarios.

With this in mind, it is now possible to gain potential performance improvements by reducing the frequency of persistent store writes.

Note: Time-based writes requires session affinity for session data integrity.

The following details apply to time-based writes:

- ▶ The expiration of the write interval does not necessitate a write to the persistent store unless the session has been touched (getAttribute/setAttribute/removeAttribute was called since the last write).
- ▶ If a session write interval has expired and the session has only been retrieved (request.getSession() was called since the last write), then the last access time will be written to the persistent store regardless of the write contents setting.
- ▶ If a session write interval has expired and the session properties have been either accessed or modified since the last write, then the session properties will be written in addition to the last access time. Which session properties get written is dependent on the write contents settings.
- ▶ Time-based write allows the servlet or JSP to issue IBMSession.sync() to force the write of session data to the database.
- ▶ If the time between session servlet requests for a particular session is greater than the write interval, then the session effectively gets written after each service method invocation.
- ▶ The session cache should be large enough to hold all of the active sessions. Failure to do this will result in extra persistent store writes, because the receipt of a new session request can result in writing out the oldest cached session to the persistent store. To put it another way, if the session manager has to remove the least recently used HttpSession from the cache during a full cache scenario, the session manager will write that HttpSession using the Write contents settings upon removal from the cache.
- ▶ The session invalidation time must be at least twice the write interval to ensure that a session does not inadvertently get invalidated prior to getting written to the persistent store.
- ▶ A newly created session will always be written to the persistent store at the end of the service method.

Writing content settings

These options control what is written. See 10.8.7, “Contents written to the persistent store using a database” on page 719 before selecting one of the options, because there are several factors to decide. The options available are:

- ▶ Only update attributes are written to the persistent store.
- ▶ All session attributes are written to the persistent store.

Session cleanup settings

WebSphere allows the administrator to defer (to off hours) the clearing of invalidated sessions from the persistent store. *Invalidated sessions* are sessions that are no longer in use and timed out. For more information, see 10.9, “Invalidating sessions” on page 723. This can be done either once or twice a day. The fields available are:

- ▶ First time of day (0-23) is the first hour during which the invalidated persistent sessions will be cleared from the persistent store. This value must be a positive integer between 0 and 23.
- ▶ Second time of day (0-23) is the second hour during which the invalidated persistent sessions will be cleared from the persistent store. This value must be a positive integer between 0 and 23.
- ▶ Select **Schedule sessions cleanup** to enable this option.

Also, consider using schedule invalidation for intranet-style applications that have a somewhat fixed number of users wanting the same HTTP session for the whole business day.

Configuration

The session management tuning parameters can be set by selecting a predefined tuning level or by specifying each parameter. To specify the performance settings for session management, do the following:

1. Select **Servers** → **Application Servers** and click the application server.

Note: Remember, session management options can also be set at the enterprise application level (see “Application session management properties” on page 673) or at the Web module level (see “Web module session management properties” on page 674).

2. Expand the **Web Container Settings** and click **Web container**.
3. Click **Session management**.
4. Click **Distributed environment settings**.
5. Select from the predefined tuning levels or click **Custom tuning parameters**.

See Figure 10-16 on page 714.

The screenshot shows a 'Configuration' window with a 'General Properties' tab. Under the 'Tuning level:' section, there are five radio button options: 'Very high (optimize for performance)', 'High', 'Medium', 'Low (optimize for failover)', and 'Custom settings'. The 'Custom settings' option is selected. Each option lists its corresponding 'Write frequency', 'Write contents', and 'Schedule sessions cleanup' parameters. At the bottom of the dialog are 'Apply', 'OK', 'Reset', and 'Cancel' buttons.

Tuning level	Write frequency	Write contents	Schedule sessions cleanup
☐ Very high (optimize for performance)	Time based: 300 seconds	Only updated attributes	true
☐ High	Time based: 300 seconds	All session attributes	false
☐ Medium	End of servlet service	Only updated attributes	false
☐ Low (optimize for failover)	End of servlet service	All session attributes	false
☒ Custom settings	Time based: 10 seconds	Only updated attributes	false

Figure 10-16 Session management tuning parameters

If you want to set each tuning parameter explicitly, select **Custom settings**. See Figure 10-17 on page 715.

The screenshot shows the 'Configuration' tab in the WebSphere Configuration console. Under the 'General Properties' section, there are three main configuration areas:

- Write frequency:** Three radio buttons are present: 'End of servlet service', 'Manual update', and 'Time based:'. The 'Time based:' option is selected, and a text box next to it contains the value '10', followed by the word 'seconds'.
- Write contents:** Two radio buttons are present: 'Only updated attributes' (which is selected) and 'All session attributes'.
- Schedule sessions cleanup:** A checkbox labeled 'Specifies distributed sessions cleanup schedule' is currently unchecked. Below this checkbox are two text boxes: 'First time of day (0-23):' and 'Second time of day (0-23):', both of which are empty.

At the bottom of the configuration area, there are four buttons: 'Apply', 'OK', 'Reset', and 'Cancel'.

Figure 10-17 Session management tuning parameters

10.8.4 Persistent sessions and non-serializable J2EE objects

In order for the WebSphere session manager to persist a session to the persistent store, all of the Java objects in an HttpSession must be serializable. They must implement the java.io.Serializable interface. The HttpSession can also contain the following J2EE objects, which are not serializable:

- ▶ javax.ejb.EJBObject
- ▶ javax.ejb.EJBHome
- ▶ javax.naming.Context
- ▶ javax.transaction.UserTransaction

The WebSphere session manager works around the problem of serializing these objects in the following manner:

- ▶ EJBObject and EJBHome each have Handle and HomeHandle object attributes that are serializable and can be used to reconstruct the EJBObject and EJBHome.
- ▶ Context is constructed with a hash table based environment, which is serializable. WebSphere will retrieve the environment, then wrap it with an internal, serializable object. On reentry, it can check the object type and reconstruct the Context.
- ▶ UserTransaction has no serializable attributes. WebSphere provides two options:
 - a. The Web developer can place the object in the HttpSession, but WebSphere will not persist it outside the JVM.
 - b. WebSphere has a new public wrapper object, `com.ibm.websphere.servlet.session.UserTransactionWrapper`, which is serializable and requires the InitialContext used to construct the UserTransaction. This will be persisted outside the JVM and be used to reconstruct the UserTransaction.

Note: According to J2EE, a Web component can only start a transaction in a service method. A transaction that is started by a servlet or JSP must be completed before the service method returns. That is, transactions cannot span Web requests from a client. If there is an active transaction after returning from the service method, WebSphere will detect it and abort the transaction.

In general, Web developers should consider making all other Java objects held by HttpSession serializable, even if immediate plans do not call for the use of persistent session management. If the Web site grows, and persistent session management becomes necessary, the transition between local and persistent management occurs transparently to the application if the sessions hold only serializable objects. If not, a switch to persistent session management requires coding changes to make the session contents serializable.

10.8.5 Larger DB2 page sizes and database persistence

WebSphere supports 4 KB, 8 KB, 16 KB, and 32 KB page sizes, and can have larger varchar for bit data columns of about 7 KB, 15 KB, or 31 KB. Using this performance feature, we see faster persistence for HttpSession of sizes of 7 KB to 31 KB in the single-row case, or attribute sizes of 4 KB to 31 KB in the multi-row case.

Enabling this feature involves dropping any existing table created with a 4 KB buffer pool and tablespace. This also applies if you subsequently change between 4 KB, 8 KB, 16 KB, or 32 KB.

To use a page size other than the default 4 KB, do the following:

1. If the SESSIONS table already exists, drop it from the DB2 database:

```
DB2 connect to session
DB2 drop table sessions
```
2. Create a new DB2 buffer pool and tablespace, specifying the same page size (8 KB, 16 KB, or 32 KB) for both, and assign the new buffer pool to this tablespace. Example 10-8 shows simple steps for creating an 8 KB page.

Example 10-8 Creating an 8K page size

```
DB2 connect to session
DB2 CREATE BUFFERPOOL sessionBP SIZE 1000 PAGESIZE 8K
DB2 connect reset
DB2 connect to session
DB2 CREATE TABLESPACE sessionTS PAGESIZE 8K MANAGED BY SYSTEM USING
('D:\DB2\NODE0000\SQL00005\sessionTS.0') BUFFERPOOL sessionBP
DB2 connect reset
```

Refer to the DB2 product documentation for details.

3. Configure the correct tablespace name and page size, DB2 row size, in the session management database configuration. See Figure 10-10 on page 697.

Restart WebSphere. On startup, the session manager creates a new SESSIONS table based on the page size and tablespace name specified.

10.8.6 Single and multi-row schemas (database persistence)

When using the single-row schema, each user session maps to a single database row. This is WebSphere's default configuration for persistent session management. With this setup, there are hard limits to the amount of user-defined, application-specific data that WebSphere Application Server can access.

When using the multi-row schema, each user session maps to multiple database rows, with each session attribute mapping to a database row.

In addition to allowing larger session records, using a multi-row schema can yield performance benefits, as discussed in 10.11.3, "Multirow persistent sessions: Database persistence" on page 731.

Switching from single-row to multi-row schema

To switch from single-row to multi-row schema for sessions, do the following:

1. Modify the session manager properties to switch from single to multi-row schema. Select the **Use multi row schema** on the Database setting of the Session Manager window, shown in Figure 10-10 on page 697.
2. Manually drop the database table or delete all the rows in the session database table. To drop the table:
 - a. Determine which data source the session manager is using. This is set in the session management distributed settings window. See 10.8.1, “Enabling database persistence” on page 694.
 - b. Look up the database name in the data source settings. Find the JDBC provider, then the data source. The database name is in the custom settings.
 - c. Use the database facilities to connect to the database and drop it.
3. Restart the application server or cluster.

Design considerations

Consider configuring direct, single-row usage to one database and multi-row usage to another database while you verify which option suits your application's specific needs. You can do this by switching the data source used, then monitoring the performance. Table 10-2 provides an overview.

Table 10-2 Single versus multi-row schemas

Programming issue	Application scenario
Reasons to use single-row	▶ You can read/write all values with just one record read/write. ▶ This takes up less space in a database, because you are guaranteed that each session is only one record long.
Reasons <i>not</i> to use single-row	There is a 2 MB limit of stored data per session. The sum of sizes of all session attributes is limited to 2 MB.

Programming issue	Application scenario
Reasons to use multi-row	▶ The application can store an unlimited amount of data. You are limited only by the size of the database and a 2 MB-per-record limit. The size of each session attribute can be 2 MB. ▶ The application can read individual fields instead of the whole record. When large amounts of data are stored in the session but only small amounts are specifically accessed during a given servlet's processing of an HTTP request, multi-row sessions can improve performance by avoiding unneeded Java object serialization.
Reasons <i>not</i> to use multi-row	If data is small in size, you probably do not want the extra overhead of multiple row reads when everything could be stored in one row.

In the case of multi-row usage, design your application data objects so they do not have references to each other. This is to prevent circular references.

For example, suppose you are storing two objects (A and B) in the session using `HttpSession.put(..)`, and A contains a reference to B. In the multi-row case, because objects are stored in different rows of the database, when objects A and B are retrieved later, the object graph between A and B is different from that stored. A and B behave as independent objects.

10.8.7 Contents written to the persistent store using a database

WebSphere supports two modes for writing session contents to the persistent store:

- ▶ Only updated attributes
Write only the `HttpSession` properties that have been updated via `setAttribute()` and `removeAttribute()`.
- ▶ All session attributes
Write all the `HttpSession` properties to the database.

When a new session is initially created with either of the above two options, the entire session is written, including any Java objects bound to the session. When using database persistence, the behavior for subsequent servlet or JSP requests for this session varies depending on whether the single-row or multi-row database mode is in use.

- ▶ In single-row mode, choose from the following:
 - Only updated attributes
If any session attribute has been updated, through `setAttribute` or `removeAttribute`, then all of the objects bound to the session will be written to the database.
 - All session attributes
All bound session attributes will be written to the database.
- ▶ In multi-row mode:
 - Only updated attributes
Only the session attributes that were specified via `setAttribute` or `removeAttribute` will be written to the database.
 - All session attributes
All of the session attributes that reside in the cache will be written to the database. If the session has never left the cache, then this should contain all of the session attributes.

By using the All session attributes mode, servlets and JSPs can change Java objects that are attributes of the `HttpSession` without having to call `setAttribute()` on the `HttpSession` for that Java object in order for the changes to be reflected in the database.

Using the All session attributes mode provides some flexibility to the application programmer and protects against possible side effects of moving from local sessions to persistent sessions.

However, using All session attributes mode can potentially increase activity and be a performance drain. Individual customers will have to evaluate the pros and cons for their installation. It should be noted that the combination of All session attributes mode with time-based write could greatly reduce the performance penalty and essentially give you the best of both worlds.

As shown in Example 10-9 and Example 10-10, the initial session creation contains a `setAttribute`, but subsequent requests for that session do not need to use `setAttribute`.

Example 10-9 Initial servlet

```
HttpSession sess = request.getSession(true);
myClass myObject = new myClass();
myObject.someInt = 1;
sess.setAttribute("myObject", myObject); // Bind object to the session
```

Example 10-10 Subsequent servlet

```
HttpSession sess = request.getSession(false);
myObject = sess.getAttribute("myObject"); // get bound session object
myObject.someInt++; // change the session object
// setAttribute() not needed with write "All session attributes" specified
```

Example 10-11 and Example 10-12 show `setAttribute` is still required even though the write all session attributes option is enabled.

Example 10-11 Initial servlet

```
HttpSession sess = request.getSession(true);
String myString = new String("Initial Binding of Session Object");
sess.setAttribute("myString", myString); // Bind object to the session
```

Example 10-12 Subsequent servlet

```
HttpSession sess = request.getSession(false);
String myString = sess.getAttribute("myString"); // get bound session object
...
myString = new String("A totally new String"); // get a new String object
sess.setAttribute("myString", myString); // Need to bind the object to the session since a NEW Object is used
```

HttpSession set/getAttribute action summary

Table 10-3 summarizes the action of the `HttpSession setAttribute` and `removeAttribute` methods for various combinations of the row type, write contents, and write frequency session persistence options.

Table 10-3 Write contents versus write frequency

Row type	Write contents	Write frequency	Action for <code>setAttribute</code>	Action for <code>removeAttribute</code>
Single-row	Only updated attributes	End of servlet service / <code>sync()</code> call with Manual update	If any of the session data has changed, then write all of this session's data from cache. ¹	If any of the session data has changed, then write all of this session's data from cache. ¹

Row type	Write contents	Write frequency	Action for setAttribute	Action for removeAttribute
Single-row	Only updated attributes	Time-based	If any of the session data has changed, then write all of this session's data from cache. ¹	If any of the session data has changed, then write all of this session's data from cache. ¹
	All session attributes	End of servlet service / sync() call with Manual update	Always write all of this session's data from cache. ²	Always write all of this session's data from cache. ²
		Time-based	Always write all of this session's data from cache.	Always write all of this session's data from cache.
Multi-row	Only updated attributes	End of servlet service / sync() call with Manual update	Write only thread-specific data that has changed.	Delete only thread-specific data that has been removed.
		Time-based	Write thread-specific data that has changed for <i>all</i> threads using this session.	Delete thread-specific data that has been removed for <i>all</i> threads using this session.
	All session attributes	End of servlet service / sync() call with Manual update	Write all session data from cache.	Delete thread-specific data that has been removed for <i>all</i> threads using this session.
		Time-based	Write all session data from cache.	Delete thread-specific data that has been removed for <i>all</i> threads using this session.

¹ When a session is written to the database while using single-row mode, *all* of the session data is written. Therefore, no database deletes are necessary for properties removed with removeAttribute(), because the write of the entire session does not include removed properties.

Multi-row mode has the notion of thread-specific data. *Thread-specific data* is defined as session data that was added or removed while executing under this thread. If you use End of servlet service or Manual update modes and enable **Only updated attributes**, then only the thread-specific data is written to the database.

10.9 Invalidating sessions

This section discusses invalidating sessions when the user no longer needs the session object. For example, when the user has logged off a site. Invalidating a session removes it from the session cache, as well as from the persistent store.

WebSphere offers three methods for invalidating session objects:

- ▶ Programmatically, you can use the `invalidate()` method on the session object. If the session object is accessed by multiple threads in a Web application, be sure that none of the threads still have references to the session object.
- ▶ An invalidator thread scans for timed-out sessions every *n* seconds, where *n* is configurable from the administrative console. The `session timeout` setting is in the general properties of the session management settings.
- ▶ For persistent sessions, the administrator can specify times when the scan runs. This feature has the following benefits when used with persistent session:
 - Persistent store scans can be scheduled during periods that normally have low demand. This avoids slowing down online applications due to contention in the persistent store.
 - When this setting is used with the `End of servlet service write frequency` option, WebSphere does not have to write the last access time with every HTTP request. The reason is that WebSphere does not have to synchronize the invalidator thread's deletion with the HTTP request access.

You can find the session cleanup schedule setting in the Session management settings under the Custom tuning properties for distributed environments.

If you are going to use session cleanup, be aware of the following:

- `HttpSession` timeouts are not enforced. Instead, all invalidation processing is handled at the configured invalidation times.
- With listeners, described in 10.9.1, “Session listeners”, processing is potentially delayed by this configuration. It is not recommended if listeners are used.

10.9.1 Session listeners

Some listener classes are defined in the Servlet 2.4 specification to listen for state changes of a session and its attributes. This allows greater control over interactions with sessions, leading programmers to monitor creation, deletion, and modification of sessions. Programmers can perform initialization tasks when a session is created, or clean up tasks when a session is removed. It is also

possible to perform some specific tasks for the attribute when an attribute is added, deleted, or modified.

The following are the Listener interfaces to monitor the events associated with the HttpSession object:

- ▶ javax.servlet.http.HttpSessionListener
Use this interface to monitor creation and deletion, including session timeout, of a session.
- ▶ javax.servlet.http.HttpSessionAttributeListener
Use this interface to monitor changes of session attributes, such as add, delete, and replace.
- ▶ javax.servlet.http.HttpSessionActivationListener
This interface monitors activation and passivation of sessions. This interface is useful to monitor if the session exists, whether in memory or not, when persistent session is used.

Table 10-4 is a summary of the interfaces and methods.

Table 10-4 Listener interfaces and their methods

Target	Event	Interface	Method
session	create	HttpSessionListener	sessionCreated()
	destroy	HttpSessionListener	sessionDestroyed()
	activate	HttpSessionActivationListener	sessionDidActivate()
	passivate	HttpSessionActivationListener	sessionWillPassivate()
attribute	add	HttpSessionAttributeListener	attributeAdded()
	remove	HttpSessionAttributeListener	attributeRemoved()
	replace	HttpSessionAttributeListener	attributeReplaced()

For more information, see *Java 2 Platform Enterprise Edition, v 1.4 API Specification* at:

<http://java.sun.com/j2ee/1.4/docs/api/index.html>

10.10 Session security

WebSphere Application Server maintains the security of individual sessions. When session manager integration with WebSphere security is enabled, the session manager checks the user ID of the HTTP request against the user ID of the session held within WebSphere. This check is done as part of the processing of the request.getSession() function. If the check fails, WebSphere throws an `com.ibm.websphere.servlet.session.UnauthorizedSessionRequestException` exception. If it succeeds, the session data is returned to the calling servlet or JSP.

Session security checking works with the standard `HttpSession`. The identity or user name of a session can be accessed through the `com.ibm.websphere.servlet.session.IBMSession` interface. An unauthenticated identity is denoted by the user name *anonymous*.

The session manager uses WebSphere's security infrastructure to determine the authenticated identity associated with a client HTTP request that either retrieves or creates a session. For information about WebSphere security features, see *WebSphere Application Server V6.1 Security Handbook*, SG24-6316.

Security integration rules for HTTP sessions

Session management security has the following rules:

- ▶ Sessions in unsecured pages are treated as accesses by the anonymous user.
- ▶ Sessions created in unsecured pages are created under the identity of that anonymous user.
- ▶ Sessions in secured pages are treated as accesses by the authenticated user.
- ▶ Sessions created in secured pages are created under the identity of the authenticated user. They can only be accessed in other secured pages by the same user. To protect these sessions from use by unauthorized users, they cannot be accessed from an unsecure page. Do not mix access to secure and unsecure pages.
- ▶ Security integration in session manager is not supported in HTTP form-based login with Simple WebSphere Authentication Mechanism (SWAM).

Table 10-5 lists possible scenarios when security integration is enabled, where outcomes depend on whether the HTTP request was authenticated and whether a valid session ID and user name was passed to the session manager.

Table 10-5 HTTP session security

Request session ID/ user name.	Unauthenticated HTTP request is used to retrieve the session.	Authenticated HTTP request is used to retrieve the session. The user ID in the HTTP request is FRED.
No session ID was passed in for this request, or the ID is for a session that is no longer valid.	A new session is created. The user name is anonymous.	A new session is created. The user name is FRED.
A valid session ID is received. The current session user name is anonymous.	The session is returned.	The session is returned. The session manager changes the user name to FRED.
A valid session ID is received. The current session user name is FRED.	The session is not returned. <code>UnauthorizedSession-RequestException</code> is thrown. ¹	The session is returned.
A valid session ID is received. The current session user name is BOB.	The session is not returned. <code>UnauthorizedSession-RequestException</code> is thrown. ¹	The session is not returned. <code>UnauthorizedSession-RequestException</code> is thrown. ¹
¹ <code>com.ibm.websphere.servlet.session.UnauthorizedSessionRequestException</code> is thrown to the servlet or JSP.		

See 10.6, “General properties for session management” on page 685 for more information about the security integration setting.

10.11 Session performance considerations

This section includes guidance for developing and administering scalable, high-performance Web applications using WebSphere Application Server session support.

10.11.1 Session size

Large session objects pose several problems for a Web application. If the site uses session caching, large sessions reduce the memory available in the WebSphere instance for other tasks, such as application execution.

For example, assume a given application stores 1 MB of information for each user session object. If 100 users arrive over the course of 30 minutes, and assume the session timeout remains at 30 minutes, the application server instance must allocate 100 MB just to accommodate the newly arrived users in the session cache:

1 MB for each user session * 100 users = 100 MB

Note this number does not include previously allocated sessions that have not timed out yet. The memory required by the session cache could be considerably higher than 100 MB.

Web developers and administrators have several options for improving the performance of session management:

- ▶ Reduce the size of the session object.
- ▶ Reduce the size of the session cache.
- ▶ Add additional application servers.
- ▶ Invalidate unneeded sessions.
- ▶ Increase the memory available.
- ▶ Reduce the session timeout interval.

Reducing session object size

Web developers must consider carefully the information kept by the session object:

- ▶ Removing information easily obtained or easily derived helps keep the session object small.
- ▶ Rigorous removal of unnecessary, unneeded, or obsolete data from the session.
- ▶ Consider whether it would be better to keep a certain piece of data in an application database rather than in the HTTP session. This gives the developer full control over when the data is fetched or stored and how it is combined with other application data. Web developers can leverage the power of SQL if the data is in an application database.

Reducing object size becomes particularly important when persistent sessions are used. Serializing a large amount of data and writing it to the persistent store requires significant WebSphere performance overhead. Even if the Write contents option is enabled, if the session object contains large Java objects or collections of objects that are updated regularly, there is a significant performance penalty in persisting these objects. This penalty can be reduced by using time-based writes.

Notes: In general, you can obtain the best performance with session objects that are less than 2 KB in size. When the session object exceeds 4-5 KB, you can expect a significant decrease in performance.

Even if session persistence is not an issue, minimizing the session object size will help to protect your Web application from scale-up disasters as user numbers increase. Large session objects will require more and more JVM memory, leaving no room to run servlets.

See 10.8.5, “Larger DB2 page sizes and database persistence” on page 716 to learn how WebSphere can provide faster persistence of larger session objects when using DB2.

Session cache size

The session manager allows administrators to change the session cache size to alter the cache’s memory footprint. By default, the session cache holds 1000 session objects. By lowering the number of session objects in the cache, the administrator reduces the memory required by the cache.

However, if the user’s session is not in the cache, WebSphere must retrieve it from either the overflow cache, for local caching, or the session database, for persistent sessions. If the session manager must retrieve persistent sessions frequently, the retrievals can impact overall application performance.

WebSphere maintains overflowed local sessions in memory, as discussed in 10.5, “Local sessions” on page 683. Local session management with cache overflow enabled allows an unlimited number of sessions in memory. To limit the cache footprint to the number of entries specified in session manager, use persistent session management, or disable overflow.

Note: When using local session management without specifying the Allow overflow property, a full cache will result in the loss of user session objects.

Creating additional application servers

WebSphere also gives the administrator the option of creating additional application servers. Creating additional instances spreads the demand for memory across more JVMs, thus reducing the memory burden on any particular instance. Depending on the memory and CPU capacity of the machines involved, the administrator can add additional instances within the same machine. Alternatively, the administrator can add additional machines to form a hardware cluster, and spread the instances across this cluster.

Note: When configuring a session cluster, session affinity routing provides the most efficient strategy for user distribution within the cluster, even with session persistence enabled. With cluster members, the Web server plug-in provides affinity routing among cluster member instances.

Invalidating unneeded sessions

If the user no longer needs the session object, for example, when the user has logged out of the site, it should be invalidated. Invalidating a session removes it from the session cache, as well as from the session database. For more information, see 10.9, “Invalidating sessions” on page 723.

Increasing available memory

WebSphere allows the administrator to increase an application server's heap size. By default, WebSphere allocates 256 MB as the maximum heap size. Increasing this value allows the instance to obtain more memory from the system, and thus hold a larger session cache.

A practical limit exists, however, for an instance heap size. The machine memory containing the instance needs to support the heap size requested. Also, if the heap size grows too large, the length of the garbage collection cycle with the JVM might impact overall application performance. This impact has been reduced with the introduction of multi-threaded garbage collection.

Session timeout interval

By default, each user receives a 30 minute interval between requests before the session manager invalidates the user's session. Not every site requires a session timeout interval this generous. By reducing this interval to match the requirements of the average site user, the session manager purges the session from the cache and the persistent store, if enabled, more quickly.

Avoid setting this parameter too low and frustrating users. The administrator must take into account a reasonable time for an average user to interact with the site when setting the interval. User activities include reading returned data, filling out forms, and so on. Also, the interval must represent any increased response time during peak times on the site, such as heavy trading days on a brokerage site, for example.

Finally, in some cases where the persistent store contains a large number of entries, frequent execution of the timeout scanner reduces overall performance. In cases where the persistent store contains many session entries, avoid setting the session timeout so low it triggers frequent, expensive scans of the persistent store for timed-out sessions. Alternatively, the administrator should consider schedule-based invalidation where scans for invalid object can be deferred to a time that normally has low demand. See 10.9, “Invalidating sessions” on page 723.

10.11.2 Reducing persistent store I/O

From a performance point of view, the Web developer’s considerations are the following:

- ▶ Optimize the use of the HttpSession within a servlet. Only store the minimum amount of data required in HttpSession. Data that does not have to be recovered after a cluster member fails or is shut down can be best kept elsewhere, such as in a hash table. Recall that HttpSession is intended to be used as a *temporary* store for state information between browser invocations.
- ▶ Specify `session=false` in the JSP directive for JSPs that do not need to access the session object.
- ▶ Use time-based write frequency mode. This greatly reduces the amount of I/O, because the persistent store updates are deferred up to a configurable number of seconds. Using this mode, all of the outstanding updates for a Web application are written periodically based on the configured write interval.
- ▶ Use the Schedule sessions cleanup option. When using the End of servlet service write frequency mode, WebSphere does not have to write out the last access time with every HTTP request. This is because WebSphere does not have to synchronize the invalidator thread's deletion with the HTTP request's access.

10.11.3 Multirow persistent sessions: Database persistence

When a session contains multiple objects accessed by different servlets or JSPs in the same Web application, multi-row session support provides a mechanism for improving performance. Multi-row session support stores session data in the persistent session database by Web application and value. Table 10-6 shows a simplified representation of a multi-row database table.

Table 10-6 Simplified multi-row session representation

Session ID	Web application	Property	Small value	Large value
DA32242SSGE2	ShoeStore	ShoeStore.First.Name	Alice	
DA32242SSGE2	ShoeStore	ShoeStore.Last.Name	Smith	
DA32242SSGE2	ShoeStore	ShoeStore.Big.String		A big string....

In this example, if the user visits the ShoeStore application, and the servlet involved needs the user's first name, the servlet retrieves this information through the session API. The session manager brings into the session cache only the value requested. The ShoeStore.Big.String item remains in the persistent session database until the servlet requests it. This saves time by reducing both the data retrieved and the serialization overhead for data the application does not use.

After the session manager retrieves the items from the persistent session database, these items remain in the in-memory session cache. The cache accumulates the values from the persistent session database over time as the various servlets within the Web application request them. With WebSphere's session affinity routing, the user returns to this same cached session instance repeatedly. This reduces the number of reads against the persistent session database, and gives the Web application better performance.

How session data is written to the persistent session database has been made configurable in WebSphere. For information about session updates using single and multi-row session support, see 10.8.6, "Single and multi-row schemas (database persistence)" on page 717. Also see 10.8.7, "Contents written to the persistent store using a database" on page 719.

Even with multi-row session support, Web applications perform best if the overall contents of the session objects remain small. Large values in session objects require more time to retrieve from the persistent session database, generate more network traffic in transit, and occupy more space in the session cache after retrieval.

Multi-row session support provides a good compromise for Web applications requiring larger sessions. However, single-row persistent session management remains the best choice for Web applications with small session objects. Single-row persistent session management requires less storage in the database, and requires fewer database interactions to retrieve a session's contents (all of the values in the session are written or read in one operation). This keeps the session object's memory footprint small, as well as reducing the network traffic between WebSphere and the persistent session database.

Note: Avoid circular references within sessions if using multi-row session support. The multi-row session support does not preserve circular references in retrieved sessions.

10.11.4 Managing your session database connection pool

When using persistent session management, the session manager interacts with the defined database through a WebSphere Application Server data source. Each data source controls a set of database connections known as a connection pool. By default, the data source opens a pool of no more than 10 connections. The maximum pool size represents the number of simultaneous accesses to the persistent session database available to the session manager.

For high-volume Web sites, the default settings for the persistent session data source might not be sufficient. If the number of concurrent session database accesses exceeds the connection pool size; the data source queues the excess requests until a connection becomes available. Data source queuing can impact the overall performance of the Web application (sometimes dramatically).

For best performance, the overhead of the connection pool used by the session manager needs to be balanced against the time that a client can spend waiting for an occupied connection to become available for use. By definition, a connection pool is a *shared* resource, so in general the best performance is realized typically with a connection pool that has significantly fewer connections than the number of simultaneous users.

A large connection pool does not necessarily improve application performance. Each connection represents memory overhead. A large pool decreases the memory available for WebSphere to execute applications. Also, if database connections are limited because of database licensing issues, the administrator must share a limited number of connections among other Web applications requiring database access as well. This is one area where performance tuning tests are required to determine the optimal setting for a given application.

As discussed above, session affinity routing combined with session caching reduces database read activity for session persistence. Likewise, manual update write frequency, time-based write frequency, and multi-row persistent session management reduce unnecessary writes to the persistent database. Incorporating these techniques can also reduce the size of the connection pool required to support session persistence for a given Web application.

Prepared statement caching is a connection pooling mechanism that can be used to further improve session database response times. A cache of previously prepared statements is available on a connection. When a new prepared statement is requested on a connection, the cached prepared statement is returned, if available. This caching reduces the number of costly prepared statements created, which improves response times.

In general, base the prepared statement cache size on the following:

- ▶ The smaller of:
 - Number of concurrent users
 - Connection pool size
- ▶ The number of different prepared statements

With 50 concurrent users, a connection pool size of 10, and each user using two statements, a select and an insert, the prepared statement cache size should be at least $10 \times 2 = 20$ statements. To read more, see the “Prepared statement cache size” article in the *WebSphere Tuning Guide*, included with the Information Center.

10.11.5 Session database tuning

While the session manager implementation in WebSphere provides for a number of parameters that can be tuned to improve performance of applications that utilize HTTP sessions, maximizing performance requires tuning the underlying session persistence table. WebSphere provides a first step by creating an index for the sessions table when creating the table. The index is comprised of the session ID, the property ID for multi-row sessions, and the Web application name.

While most database managers provide a great deal of capability in tuning at the table or tablespace level, creating a separate database or instance provides the most flexibility in tuning. Proper tuning of the instance and database can improve performance by 5% or more over that which can be achieved by simply tuning the table or tablespace.

While the specifics vary, depending on your database and operating system, in general, tune and configure the database as appropriate for a database that experiences a great deal of I/O. The database administrator (DBA) should monitor and tune the database buffer pools, database log size, and write frequency. Additionally, maximizing performance requires striping the database or instance across multiple disk drives and disk controllers, and utilizing any hardware or OS buffering available to reduce disk contention.

10.12 Stateful session bean failover

Stateful session bean failover is supported now in WebSphere Application Server V6. This feature utilizes the functions of the data replication service and workload management.

Each EJB container provides a method for stateful session beans to fail over to other servers. This enables you to specify whether failover occurs for the stateful session beans at the EJB module level or container level. You can also override the parent object's stateful session bean replication settings from the module level.

10.12.1 Enabling stateful session bean failover

Depending on the requirement, you might not want to enable failover for every single stateful session bean installed in the EJB container. You can set or override the EJB container settings at either the application or EJB module level. You can either enable or disable failover at each of these levels. For example, consider the following situations:

- ▶ You want to enable failover for all applications except for a single application. To do this, you enable failover at the EJB container level and override the setting at the application level to disable failover on the single application.
- ▶ You want to enable failover for a single, installed application. To do this, disable failover at the EJB container level and then override the setting at the application level to enable failover on the single application.
- ▶ You want to enable failover for all applications except for a single module of an application. To do this, enable failover at the EJB container level, then override the setting at the module application level to disable failover on the single module.
- ▶ You want to enable failover for a single, installed EJB module. To do this, disable failover at the EJB container level and then override the setting at the EJB module level to enable failover on the single EJB module.

EJB container stateful session bean failover properties

To access stateful session bean failover properties at the EJB container level from the administrative console:

1. Select **Servers** → **Application servers**.
2. Click the application server.
3. In the Container Settings section of the Configuration tab, click **EJB container**.
4. In the General Properties section, check **Enable stateful session bean failover using memory-to-memory replication**.

This check box is disabled until you define a replication domain. This selection has a hyperlink to help you configure the replication settings. If no replication domains are configured, the link takes you to a window where you can create one. If at least one domain is configured, the link takes you to a window where you can select the replication settings to be used by the EJB container. See Figure 10-18 on page 735.

The screenshot shows the 'Configuration' window for the EJB container. It is divided into two main sections: 'General Properties' and 'Additional Properties'. In the 'General Properties' section, the 'Passivation directory' is set to '\$ {USER_INSTALL_ROOT} /ten', the 'Inactive pool cleanup interval' is 30000 milliseconds, and the 'Default data source JNDI name' is '(none)'. The 'Enable stateful session bean failover using memory-to-memory replication' checkbox is disabled, with a note indicating that replication domains are defined but memory-to-memory settings have not been selected. The 'Initial State' is set to 'Started'. The 'Additional Properties' section contains two links: 'EJB cache settings' and 'EJB timer service settings'. At the bottom, there are buttons for 'Apply', 'OK', 'Reset', and 'Cancel'.

General Properties	Additional Properties
* Passivation directory: \$ {USER_INSTALL_ROOT} /ten - Inactive pool cleanup interval: 30000 milliseconds - Default data source JNDI name: (none) ☐ Enable stateful session bean failover using memory-to-memory replication (Replication domains are defined, but the memory to memory settings have not been selected.) - Initial State: Started	EJB cache settings EJB timer service settings

Apply OK Reset Cancel

Figure 10-18 Stateful session bean failover settings at the container level

EJB module stateful session bean failover properties

To access stateful session bean failover properties at the EJB module level from the administrative console:

1. Select **Applications** → **Enterprise applications**.
2. Click the application.
3. In the Enterprise Java Bean Properties section of the Configuration tab, click **Stateful session bean failover settings**.

This enables failover for all stateful session beans in this application. If you want to disable the failover, clear this check box. See Figure 10-19 on page 736.

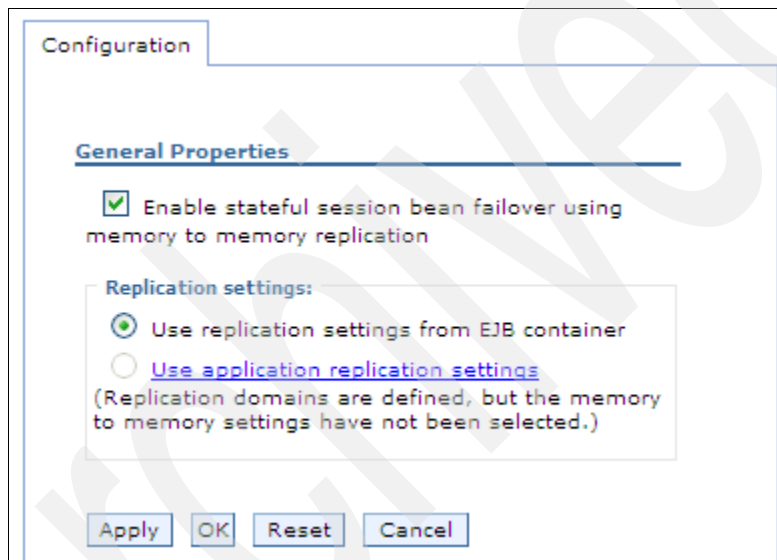


Figure 10-19 Stateful session bean failover settings at the module level

4. In the General Properties section, select your choice of Replication settings:
 - Use replication settings from EJB container

If you select this option, any replication settings defined for this application are ignored.

Important: If you use this radio button, then you must configure memory to memory replication at the EJB container level. Otherwise, the settings on this window are ignored by EJB container during server startup and the EJB container will log a message indicating that stateful session bean failover is not enabled for this application.

- Use application replication settings

If you select this option, you override the EJB container settings. This button is disabled until you define a replication domain. This selection has a hyperlink to help you configure the replication settings. If no replication domains are configured, the link takes you to a window to create one. If at least one domain is configured, the link takes you to a window where you can select the replication settings to be used by the application.

5. Select your choice of replication from:

- **Use replication settings from EJB container**
- **Use application replication settings using memory-to-memory replication**

6. Select **OK**.

Note: The stateful session bean failover settings are available to WebSphere Application Server V6 enterprise applications. They are ignored by WebSphere Application Server V5 enterprise applications.

10.12.2 Stateful session bean failover considerations

The following presents a few considerations when using the stateful session bean failover feature.

Stateful session bean activation policy with failover enabled

WebSphere Application Server V6 allows an application assembler to specify an activation policy to use for stateful session beans. It is important to consider that the only time the EJB container prepares for failover, by replicating the stateful session bean data using DRS, is when the stateful session bean is passivated. If you configure the bean with an activate once policy, the bean is essentially never passivated. If you configure the activate at transaction boundary policy, the bean is passivated whenever the transaction that the bean is enlisted in completes. For stateful session bean failover to be useful, the activate at transaction boundary policy is required.

Rather than forcing you to edit the deployment descriptor of every stateful session bean and reinstall the bean, the EJB container simply ignores the configured activation policy for the bean when you enable failover. The container automatically uses the activate at transaction boundary policy.

Container or bean managed units of work

The relevant units of work in this case are transactions and activity sections. WebSphere Application Server V6 supports stateful session bean failover for:

- ▶ Container managed transactions (CMT)
- ▶ Bean managed transactions (BMT)
- ▶ Container managed activity sessions (CMAS)
- ▶ Bean managed activity sessions (BMAS)

In the container-managed cases, preparation for failover only occurs if a request for an enterprise bean method invocation fails to connect to the server. Also, failover does not take place if the server fails after a request is sent to it and had been acknowledged.

When a failure occurs in the middle of a request or unit of work, WLM cannot safely fail over to another server without some compensation code being executed by the application. When that happens, the application receives a Common Object Request Broker Architecture (CORBA) exception and minor code telling it that transparent failover could not occur because the failure happened during execution of a unit of work. The application should be written to check for the CORBA exception and minor code, and compensate for the failure. After the compensation code executes, the application can retry the requests and, if a path exists to a backup server, WLM routes the new request to a new primary server for the stateful session bean.

The same is true for bean-managed units of work, transactions, or activity sessions. However, bean managed work introduces a new possibility that needs to be considered.

For bean managed units of work, the failover process is not always able to detect that a BMT or BMAS started by a stateful session bean method has not completed. Thus, it is possible that failover to a new server can occur despite the unit of work failing during the middle of a transaction or session. Because the unit of work is implicitly rolled back, WLM behaves as though it is safe to transparently fail over to another server, when in fact some compensation code might be required. When this happens, the EJB container detects this on the new server and initiates an exception. This exception occurs under the following scenario:

1. A method of a stateful session bean using bean-managed transaction or activity session calls begin on a UserTransaction it obtained from the SessionContext. The method does some work in the started unit of work, but does not complete the transaction or session before returning to the caller of the method.

2. During post invocation of the method started in step 1, the EJB container suspends the work started by the method. This is the action required by EJB specification for bean managed units of work when the bean is a stateful session bean.
3. The client starts several other methods on the stateful session bean. Each invocation causes the EJB container to resume the suspended transaction or activity session, dispatch the method invocation, and then suspend the work again before returning to the caller.
4. The client calls a method on the stateful session bean that completes the transaction or session started in step 1.

This scenario depicts a *sticky* bean-managed unit of work. The transaction or activity session sticks around for more than a single stateful session bean method. If an application uses a sticky BMT or BMAS, and the server fails after a sticky unit of work completes and before another sticky unit of work starts, failover is successful. However, if the server fails before a sticky transaction or activity session completes, the failover is not successful. Instead, when the failover process routes the stateful session bean request to a new server, the EJB container detects that the failure occurred during an active, sticky transaction or activity session. At that time, the EJB container initiates an exception.

Essentially, this means that failover for both container-managed and bean-managed units of work is not successful if the transaction or activity session is still active. The only real difference is the exception that occurs.

Application design considerations

Consider the following when designing applications that use the stateful session bean failover process:

- ▶ To avoid the possibility described in the section above, you are encouraged to write your application to configure stateful session beans to use container-managed transactions (CMT) rather than bean-managed transactions (BMT).
 - ▶ If you want immediate failover, and your application creates either an HTTP session or a stateful session bean that stores a reference to another stateful session bean, then the administrator must ensure the HTTP session and stateful session bean are configured to use the same replication domain.
 - ▶ Do not use a local and a remote reference to the same stateful session bean.
- The J2EE 1.4 specification has added additional requirements for Http Sessions that require the Http Session state objects to be able to contain local references to EJBs.

Normally a stateful session bean instance with a given primary key can only exist on a single server at any given moment in time. Failover might cause the bean to be moved from one server to another, but it never exists on more than one server at a time. However, there are some unlikely scenarios that can result in the same bean instance, the same primary key, existing on more than one server concurrently. When that happens, each copy of the bean is unaware of the other, and no synchronization occurs between the two instances to ensure they have the same state data. Thus, your application receives unpredictable results.

Note: To avoid this situation you must remember that with failover enabled, your application should never get both a local (EJBLocalObject) and remote (EJBObject) reference to the same stateful session bean instance.

WebSphere naming implementation

In this chapter, we describe the concepts behind the naming functionality provided as part of IBM WebSphere Application Server:

- ▶ Features
- ▶ WebSphere naming architecture
- ▶ Interoperable Naming Service (INS)
- ▶ Distributed CosNaming
- ▶ Configured bindings
- ▶ Initial contexts
- ▶ Federation of name spaces
- ▶ Foreign cell bindings
- ▶ Interoperability
- ▶ Examples
- ▶ Naming tools
- ▶ Configuration

11.1 Features

The following are features of a WebSphere Application Server V6 name space that remain unchanged from WebSphere Application Server V5:

- ▶ Distributed name space

For additional scalability, the name space for a cell is distributed among the various servers. The deployment manager, node agent, and application server processes all host a name server.

The default initial context for a server is its server root. System artifacts, such as EJB homes and resources, are bound to the server root of the server with which they are associated.

- ▶ Transient and persistent partitions

The name space is partitioned into transient areas and persistent areas. Server roots are transient. System-bound artifacts such as EJB homes and resources are bound under server roots. There is a cell persistent root, which can be used for cell-scoped persistent bindings, and a node persistent root, which can be used to bind objects with a node scope.

- ▶ Federated name space structure

A name space is a collection of all names bound to a particular name server. A name space can contain naming context bindings to contexts located in other servers. If this is the case, the name space is said to be a *federated name space*, because it is a collection of name spaces from multiple servers. The name spaces link together to cooperatively form a single logical name space.

The name space for the WebSphere Application Server V6 cell is federated among the deployment manager, node agents, and application servers of the cell. Every server process hosts a name server. All name servers provide the same logical view of the cell name space, with the various server roots and persistent partitions of the name space being interconnected by means of the single logical name space.

- ▶ Configured bindings

Administrators can configure bindings into the name space. A configured binding is different from a programmatic binding in that the system creates the binding every time a server is started, even if the target context is in a transient partition.

- ▶ Support for CORBA Interoperable Naming Service (INS) object URLs

WebSphere Application Server contains support for CORBA object URLs (corbaloc and corbaname) as JNDI provider URLs and lookup names.

11.2 WebSphere naming architecture

WebSphere Application Server name servers are an implementation of the CORBA CosNaming interface. WebSphere Application Server provides a JNDI implementation that you can use to access CosNaming name servers through the JNDI interface. CosNaming provides the server-side implementation and is where the name space is stored. JNDI essentially provides a client-side wrapper of the name space stored in CosNaming, and interacts with the CosNaming server on behalf of the client.

The model of JNDI over CosNaming has existed in several releases of WebSphere. Since J2EE 1.3, limited CosNaming functionality has been required for interoperability between application servers from different vendors. This level of CosNaming, known as Interoperable Naming Service (INS), was introduced in WebSphere Application Server V5.

The following sections provide a summary of the WebSphere naming architecture, its federated name space, and its support for JNDI.

For an explanation of the WebSphere implementations of INS and Distributed CosNaming, see 11.3, “Interoperable Naming Service (INS)” on page 757 and 11.4, “Distributed CosNaming” on page 759 respectively.

11.2.1 Components

WebSphere application clients use the naming service to obtain references to objects related to those applications, such as EJB homes. These objects are bound into a mostly hierarchical structure, referred to as a *name space*. In this structure, all non-leaf objects are called *contexts*. Leaf objects can be contexts and other types of objects. Naming operations, such as lookups and binds, are performed on contexts. All naming operations begin with obtaining an initial context. You can view the initial context as a starting point in the name space.

The name space structure consists of a set of *name bindings*, each consisting of a name relative to a specific context and the object bound with that name. For example, the name `myApp/myEJB` consists of one non-leaf binding with the name `myApp`, which is a context. The name also includes one leaf binding with the name `myEJB`, relative to `myApp`. The object bound with the name `myEJB` in this example happens to be an EJB home reference. The whole name `myApp/myEJB` is relative to the initial context, which can be viewed as a starting place when performing naming operations.

The name space can be accessed and manipulated through a *name server*. Users of a name server are referred to as naming clients. Naming clients typically use Java Naming and Directory Interface (JNDI) to perform naming

operations. Naming clients can also use the Common Object Request Broker Architecture (CORBA) CosNaming interface.

Figure 11-1 summarizes the naming architecture and its components.

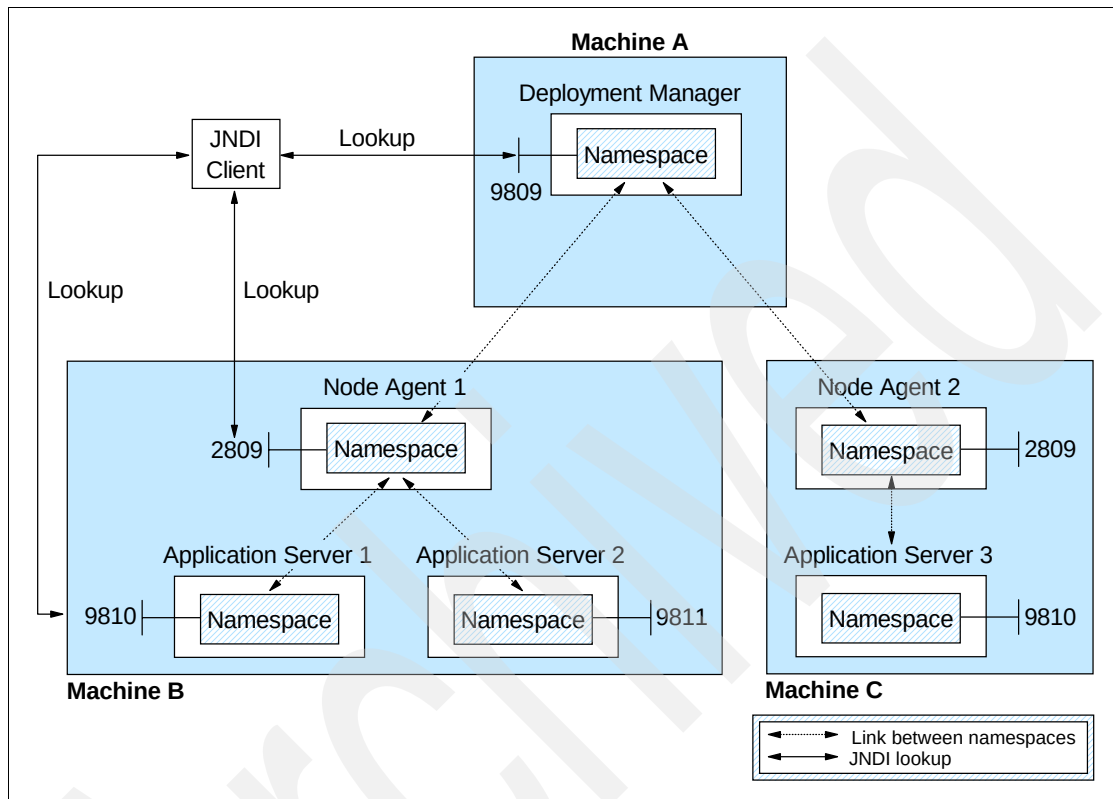


Figure 11-1 Naming topology

Notice that all WebSphere Application Server processes host their own naming service and local name space. Also the name servers in the deployment manager and node agents are listening on their default ports of 9809 and 2809, respectively. The name servers within each application server are listening from a starting default port of 9810.

11.2.2 JNDI support

Each IBM WebSphere Application Server managed process (JVM) includes:

- ▶ A name server providing shared access to its components

- ▶ An implementation of the javax.naming JNDI package, allowing users to access the WebSphere name server through the JNDI naming interface

Note: The JNDI implementation provided by IBM WebSphere Application Server is based on Version 1.2.1 of the JNDI interface, and was tested with Version 1.2.1 of Sun's JNDI SPI (Service Provider Interface).

IBM WebSphere Application Server does not provide implementations for the following Java extension packages:

- ▶ javax.naming.directory
- ▶ javax.naming.ldap

In addition, IBM WebSphere Application Server does not support interfaces defined in the javax.naming.event package.

However, to provide access to LDAP servers, the JDK shipped with IBM WebSphere Application Server supports Sun Microsystems's implementation of:

- ▶ javax.naming.ldap
- ▶ com.sun.jndi.ldap.LdapCtxFactory

11.2.3 JNDI bindings

There are three options available for binding EJB (<ejb-ref>) and resource (<resource-ref>) object names to the name space:

- ▶ Simple name
- ▶ Compound/fully qualified name
- ▶ Corbaname

The binding you can use to look up an object depends on whether or not the application is running within the same application server. The following sections describe each of these in more detail.

Simple name

The simple name binding is guaranteed to succeed if lookup is within the same server or when connected directly to the name space of the server containing the target of the lookup. It can be used in a servlet or EJB, if it is certain that the object is located on the same application server. Here is an example of a simple name:

```
ejb/webbank/Account
```

Lookup names of this form provide a level of indirection such that the name used to look up an object is not dependent on the object's name as it is bound in the

name server's name space. The deployment descriptors for the application provide the mapping between the name and the name server lookup name. The container sets up the name space based on the deployment descriptor information so that the name is correctly mapped to the corresponding object.

Compound name

Applications that do not run in the same server cannot use simple name lookup because the simple name is not local to the application. Instead, an application of this type must look the object up directly from the name server. Each application server contains a name server. System artifacts such as EJB homes are bound relative to the server root context in that name server.

The fully qualified (compound name) JNDI name is always guaranteed to work. Here is an example of a compound name:

```
cell/nodes/node1/servers/server1/ejb/webbank/Account
```

We recommend using compound names for JNDI bindings.

Corbaname

The corbaname binding is always guaranteed to work. However, it requires that you know the correct path to the object at deployment time. Here is an example of a corbaname:

```
corbaname::myhost1:9812/NameServiceServerRoot#ejb/webbank/Account
```

11.2.4 Federated name space

All name servers with a cell are federated into the cell name space. Every application server process contains a name server. All name servers provide the same logical view of the cell name space. The various server roots and persistent partitions of the name space are interconnected by a system name space. You can use the system name space structure to traverse any context in the cell's name space. A logical view of the name space is shown in Figure 11-2 on page 747.



System partition

The root of this structure is the cell root and contains a node root for each node in the cell. You can access other contexts to a specific node from the node root, such as the node persistent root and server roots for servers configured in that node.

Chapter 11. WebSphere naming implementation **747**

Cell and node persistent partitions

The persistent partitions are primarily for the storage of resource configuration, such as data sources, JMS destinations, and so on. This data can be modified by accessing the JNDI APIs directly, or through the administration clients, which access the APIs on the user's behalf. The persistent data is stored to a group of XML files.

There are two persistent partitions in the federated name space:

- ▶ Cell persistent root

This partition is used to register persistent objects that are available to all the nodes and managed processes of a cell. A binding created under the cell persistent root is saved as part of the cell configuration and continues to exist until it is explicitly removed.

Applications that need to create additional persistent object bindings associated with the cell can bind those objects under the cell persistent root.

Note: The cell persistent root is not designed for transient, rapidly changing bindings. Instead, the bindings should be more static in nature, such as part of application setup or configuration, and not created at run time.

To bind objects to the cell persistent root, the deployment manager and all node agents in the cell must be running.

- ▶ Node persistent root

This partition is used to register persistent objects available to the nodes and their managed processes. It is similar to the cell partition except that each node has its own node persistent root. A binding created under a node persistent root is saved as part of that node's configuration and continues to exist until it is explicitly removed.

Applications that need to create additional persistent object bindings associated with a specific node can bind those objects under that particular node's node persistent root.

Note: The node persistent area is not designed for transient, rapidly changing bindings. Instead, the bindings should be more static in nature, such as part of application setup or configuration, and not created at run time.

The node persistent area for a node can be read from any server in the node even if the respective node agent is not running. However, the node agent must be running to update the node persistent area, or for any server outside the node to read from that node persistent partition.

In the federated name space, there is no node root for the deployment manager node because no node agent or application servers run in that node.

Transient partitions

The server root transient partition in Figure 11-2 on page 747 is updateable through APIs, and is meant for information, such as EJB bindings and JNDI names. This name space is transient and bindings are created each time a server process starts. It reads configuration data from the file system, for example, EJB deployment descriptors, to register the necessary objects in this space.

Note: The Naming Service of each managed process listens to configuration changes. This means the local name space is updated automatically when configuration changes occur. For example, there is no need to restart a node agent to update its name space when a new application server is created.

11.2.5 Local name space structure

The structure of the federated name space is hierarchical, with each process' local name space federated and linked using corbaloc URLs that allow transparent name searches both within a single local name space and from one local name space to another.

The contents of the cell, node, and process local name spaces are described in the following sections.

Cell-level name space

The cell-level name space, hosted by the deployment manager, has the structure shown in Example 11-1.

(top) represents the root of the federated name space.

Example 11-1 Cell name space dump (dumpNameSpace -port <dmgr_bootstrap>)

```
1 (top)
2 (top)/clusters                                javax.naming.Context
3 (top)/clusters/MyCluster                      javax.naming.Context
3   Linked to URL: corbaloc::wan:9811,:wan:9812/NameServiceServerRoot
4 (top)/domain                                  javax.naming.Context
4   Linked to context: ITS0Cell
5 (top)/legacyRoot                             javax.naming.Context
5   Linked to context: ITS0Cell/persistent
6 (top)/persistent                             javax.naming.Context
7 (top)/persistent/cell                        javax.naming.Context
7   Linked to context: ITS0Cell
8 (top)/cellname                               java.lang.String
9 (top)/cell                                   javax.naming.Context
9   Linked to context: ITS0Cell
10 (top)/nodes                                 javax.naming.Context
11 (top)/nodes/ITS0CellManager                 javax.naming.Context
12 (top)/nodes/ITS0CellManager/domain          javax.naming.Context
12   Linked to context: ITS0Cell
13 (top)/nodes/ITS0CellManager/servers         javax.naming.Context
14 (top)/nodes/ITS0CellManager/servers/dmgr    javax.naming.Context
15 (top)/nodes/ITS0CellManager/servers/dmgr/tm javax.naming.Context
16 (top)/nodes/ITS0CellManager/servers/dmgr/tm/default
16
com.ibm.ws.asynchbeans.timer.TimerManagerImpl
17 (top)/nodes/ITS0CellManager/servers/dmgr/com.ibm.isc
17                                           javax.naming.Context
18 (top)/nodes/ITS0CellManager/servers/dmgr/com.ibm.isc/PluginRegistry
18
com.ibm.ws.PluginRegistry
19 (top)/nodes/ITS0CellManager/servers/dmgr/services javax.naming.Context
20 (top)/nodes/ITS0CellManager/servers/dmgr/services/cache
20                                           javax.naming.Context
21 (top)/nodes/ITS0CellManager/servers/dmgr/services/cache/basecache
21
com.ibm.websphere.cache.DistributedObjectCache
22 (top)/nodes/ITS0CellManager/servers/dmgr/services/cache/distributedmap
22
com.ibm.websphere.cache.DistributedObjectCache
23 (top)/nodes/ITS0CellManager/servers/dmgr/ejb    javax.naming.Context
24 (top)/nodes/ITS0CellManager/servers/dmgr/ejb/mgmt javax.naming.Context
```

```

25 (top)/nodes/ITSOCellManager/servers/dmgr/ejb/mgmt/MEJB
25
javax.management.j2ee.ManagementHome
26 (top)/nodes/ITSOCellManager/servers/dmgr/cell      javax.naming.Context
26   Linked to context: ITSOCell
27 (top)/nodes/ITSOCellManager/servers/dmgr/servername
27   java.lang.String
28 (top)/nodes/ITSOCellManager/servers/dmgr/thisNode  javax.naming.Context
28   Linked to context: ITSOCell/nodes/ITSOCellManager
29 (top)/nodes/ITSOCellManager/node                  javax.naming.Context
29   Linked to context: ITSOCell/nodes/ITSOCellManager
30 (top)/nodes/ITSOCellManager/cell                  javax.naming.Context
30   Linked to context: ITSOCell
31 (top)/nodes/ITSOCellManager/nodename              java.lang.String
32 (top)/nodes/NodeA                                 javax.naming.Context
33 (top)/nodes/NodeA/nodename                        java.lang.String
34 (top)/nodes/NodeA/persistent                      javax.naming.Context
34   Linked to URL: corbaname::wan:2809/NameServiceNodeRoot#persistent
35 (top)/nodes/NodeA/cell                            javax.naming.Context
35   Linked to context: ITSOCell
36 (top)/nodes/NodeA/domain                          javax.naming.Context
36   Linked to context: ITSOCell
37 (top)/nodes/NodeA/nodeAgent                      javax.naming.Context
37   Linked to URL: corbaloc::wan:2809/NameServiceServerRoot
38 (top)/nodes/NodeA/node                            javax.naming.Context
38   Linked to context: ITSOCell/nodes/NodeA
39 (top)/nodes/NodeA/servers                        javax.naming.Context
40 (top)/nodes/NodeA/servers/MyClusterServer2        javax.naming.Context
40   Linked to URL: corbaloc::wan:9812/NameServiceServerRoot
41 (top)/nodes/NodeA/servers/server1                 javax.naming.Context
41   Linked to URL: corbaloc::wan:9810/NameServiceServerRoot
42 (top)/nodes/NodeA/servers/nodeagent               javax.naming.Context
42   Linked to URL: corbaloc::wan:2809/NameServiceServerRoot
43 (top)/nodes/NodeA/servers/MyClusterServer1        javax.naming.Context
43   Linked to URL: corbaloc::wan:9811/NameServiceServerRoot
44 (top)/deploymentManager                          javax.naming.Context
44   Linked to context: ITSOCell/nodes/ITSOCellManager/servers/dmgr
45 (top)/cells                                       javax.naming.Context

```

The cell-level name space contains:

- ▶ A link to the cell persistent root, /persistent/cell
- ▶ A hierarchy of contexts for nodes and the servers managed by each node
- ▶ A full set of entries for the deployment manager node (ITSOCellManager) and the deployment manager server (dmgr)
- ▶ The objects registered in JNDI by the dmgr server

- ▶ A corbaloc URL link to the local name space of each of the other nodes in the cell
- ▶ A number of cross links for the federated name space:
 - /cells
 - /clusters
 - /legacyRoot

Note: Because of the hierarchical structure of the cell/node/server relationship, the following naming conventions and constraints exist:

1. No two nodes can have the same name. Node names must be unique within a cell.
2. Two application servers on different nodes can have the same name.
3. Two application servers on the same node must have different names. Application server names must be unique within a node.

Node-level name space

A node-level name space, hosted by a node agent, has a structure as shown in Example 11-2.

Example 11-2 Node-level name space (dumpNameSpace -port <NodeA_bootstrap>)

```

1 (top)
2 (top)/clusters                                javax.naming.Context
3 (top)/clusters/MyCluster                      javax.naming.Context
3   Linked to URL: corbaloc::wan:9811,wan:9812/NameServiceServerRoot
4 (top)/domain                                javax.naming.Context
4   Linked to context: ITS0Cell
5 (top)/legacyRoot                            javax.naming.Context
5   Linked to context: ITS0Cell/persistent
6 (top)/persistent                            javax.naming.Context
7 (top)/persistent/cell                       javax.naming.Context
7   Linked to context: ITS0Cell
8 (top)/cellname                              java.lang.String
9 (top)/cell                                javax.naming.Context
9   Linked to context: ITS0Cell
10 (top)/nodes                                javax.naming.Context
11 (top)/nodes/ITS0CellManager                 javax.naming.Context
12 (top)/nodes/ITS0CellManager/domain          javax.naming.Context
12   Linked to context: ITS0Cell
13 (top)/nodes/ITS0CellManager/servers         javax.naming.Context
14 (top)/nodes/ITS0CellManager/servers/dmgr    javax.naming.Context
14   Linked to URL: corbaloc::wan:9809/NameServiceServerRoot
15 (top)/nodes/ITS0CellManager/node            javax.naming.Context
15   Linked to context: ITS0Cell/nodes/ITS0CellManager
16 (top)/nodes/ITS0CellManager/nodename        java.lang.String

```

17 (top)/nodes/ITS0CellManager/cell	javax.naming.Context
17 Linked to context: ITS0Cell	
18 (top)/nodes/NodeA	javax.naming.Context
19 (top)/nodes/NodeA/nodename	java.lang.String
20 (top)/nodes/NodeA/persistent	javax.naming.Context
21 (top)/nodes/NodeA/cell	javax.naming.Context
21 Linked to context: ITS0Cell	
22 (top)/nodes/NodeA/domain	javax.naming.Context
22 Linked to context: ITS0Cell	
23 (top)/nodes/NodeA/nodeAgent	javax.naming.Context
23 Linked to context: ITS0Cell/nodes/NodeA/servers/nodeagent	
24 (top)/nodes/NodeA/node	javax.naming.Context
24 Linked to context: ITS0Cell/nodes/NodeA	
25 (top)/nodes/NodeA/servers	javax.naming.Context
26 (top)/nodes/NodeA/servers/MyClusterServer2	javax.naming.Context
26 Linked to URL: corbaloc::wan:9812/NameServiceServerRoot	
27 (top)/nodes/NodeA/servers/server1	javax.naming.Context
27 Linked to URL: corbaloc::wan:9810/NameServiceServerRoot	
28 (top)/nodes/NodeA/servers/nodeagent	javax.naming.Context
29 (top)/nodes/NodeA/servers/nodeagent/cell	javax.naming.Context
29 Linked to context: ITS0Cell	
30 (top)/nodes/NodeA/servers/nodeagent/servername	java.lang.String
31 (top)/nodes/NodeA/servers/nodeagent/thisNode	javax.naming.Context
31 Linked to context: ITS0Cell/nodes/NodeA	
32 (top)/nodes/NodeA/servers/MyClusterServer1	javax.naming.Context
32 Linked to URL: corbaloc::wan:9811/NameServiceServerRoot	
33 (top)/deploymentManager	javax.naming.Context
33 Linked to URL: corbaloc::wan:9809/NameServiceServerRoot	
34 (top)/cells	javax.naming.Context

The node-level name space contains:

- ▶ A full set of entries for the node and the node agent server (<nodename> or nodeAgent)
- ▶ A corbaloc URL link to the local name space root (NameServiceServerRoot) of each application server managed by the node
- ▶ Links to other nodes in the cell and to the servers on those nodes
- ▶ A corbaloc URL link to the root of the deployment manager local name space (NameServiceServerRoot)
- ▶ A number of cross-links for the federated name space:
 - /cells
 - /clusters
 - /legacyRoot
- ▶ A link to the cell persistent root, /persistent/cell

- A link to the node persistent root, /nodes/<nodename>/persistent

Managed process-level name space

A process-level name space, hosted by a managed process, has the structure shown in Example 11-3 on page 754.

Example 11-3 Managed process-level name space

```

1 (top)
2 (top)/domain                                javax.naming.Context
2   Linked to context: ITSOCe11
3 (top)/cellname                              java.lang.String
4 (top)/nodes                                javax.naming.Context
5 (top)/nodes/ITSOCe11Manager                 javax.naming.Context
6 (top)/nodes/ITSOCe11Manager/servers         javax.naming.Context
7 (top)/nodes/ITSOCe11Manager/servers/dmgr    javax.naming.Context
7   Linked to URL: corbaloc::wan:9809/NameServiceServerRoot
8 (top)/nodes/ITSOCe11Manager/domain          javax.naming.Context
8   Linked to context: ITSOCe11
9 (top)/nodes/ITSOCe11Manager/cell            javax.naming.Context
9   Linked to context: ITSOCe11
10 (top)/nodes/ITSOCe11Manager/nodename       java.lang.String
11 (top)/nodes/ITSOCe11Manager/node           javax.naming.Context
11   Linked to context: ITSOCe11/nodes/ITSOCe11Manager
12 (top)/nodes/NodeA                         javax.naming.Context
13 (top)/nodes/NodeA/nodename                java.lang.String
14 (top)/nodes/NodeA/persistent              javax.naming.Context
15 (top)/nodes/NodeA/cell                    javax.naming.Context
15   Linked to context: ITSOCe11
16 (top)/nodes/NodeA/domain                  javax.naming.Context
16   Linked to context: ITSOCe11
17 (top)/nodes/NodeA/nodeAgent               javax.naming.Context
17   Linked to URL: corbaloc::wan:2809/NameServiceServerRoot
18 (top)/nodes/NodeA/node                    javax.naming.Context
18   Linked to context: ITSOCe11/nodes/NodeA
19 (top)/nodes/NodeA/servers                 javax.naming.Context
20 (top)/nodes/NodeA/servers/server1         javax.naming.Context
21 (top)/nodes/NodeA/servers/server1/servername java.lang.String
22 (top)/nodes/NodeA/servers/server1/services javax.naming.Context
23 (top)/nodes/NodeA/servers/server1/services/cache javax.naming.Context
24 (top)/nodes/NodeA/servers/server1/services/cache/basecache
24                                     com.ibm.websphere.cache.DistributedObjectCache
25 (top)/nodes/NodeA/servers/server1/services/cache/distributedmap
25                                     com.ibm.websphere.cache.DistributedObjectCache
26 (top)/nodes/NodeA/servers/server1/thisNode javax.naming.Context
26   Linked to context: ITSOCe11/nodes/NodeA
27 (top)/nodes/NodeA/servers/server1/com     javax.naming.Context
28 (top)/nodes/NodeA/servers/server1/com/ibm javax.naming.Context

```

```

29 (top)/nodes/NodeA/servers/server1/com/ibm/websphere
29                                     javax.naming.Context
30 (top)/nodes/NodeA/servers/server1/com/ibm/websphere/ejbquery
30                                     javax.naming.Context
31 (top)/nodes/NodeA/servers/server1/com/ibm/websphere/ejbquery/Query
31                                     com.ibm.websphere.ejbquery.QueryHome
32 (top)/nodes/NodeA/servers/server1/Increment
32                                     com.ibm.defaultapplication.IncrementHome
33 (top)/nodes/NodeA/servers/server1/DefaultDataSource
33                                     javax.resource.cci.ConnectionFactory
34 (top)/nodes/NodeA/servers/server1/eis                                     javax.naming.Context
35 (top)/nodes/NodeA/servers/server1/eis/jdbc                             javax.naming.Context
36 (top)/nodes/NodeA/servers/server1/eis/jdbc/PlantsByWebSphereDataSource_CMP
36                                     javax.resource.cci.ConnectionFactory
37 (top)/nodes/NodeA/servers/server1/eis/DefaultDataSource_CMP
37                                     javax.resource.cci.ConnectionFactory
38 (top)/nodes/NodeA/servers/server1/jdbc                             javax.naming.Context
39 (top)/nodes/NodeA/servers/server1/jdbc/PlantsByWebSphereDataSource
39                                     javax.resource.cci.ConnectionFactory
40 (top)/nodes/NodeA/servers/server1/jdbc/DefaultEJBTimerDataSource
40                                     javax.resource.cci.ConnectionFactory
41 (top)/nodes/NodeA/servers/server1/tm                             javax.naming.Context
42 (top)/nodes/NodeA/servers/server1/tm/default
com.ibm.ws.asynchbeans.timer.TimerManagerImpl
43 (top)/nodes/NodeA/servers/server1/plantsby                             javax.naming.Context
44 (top)/nodes/NodeA/servers/server1/plantsby/LoginHome
44                                     com.ibm.websphere.samples.plantsbywebsphereejb.LoginHome
45 (top)/nodes/NodeA/servers/server1/plantsby/MailerHome
45                                     com.ibm.websphere.samples.plantsbywebsphereejb.MailerHome
46 (top)/nodes/NodeA/servers/server1/plantsby/BackOrderHome
46                                     com.ibm.websphere.samples.plantsbywebsphereejb.BackOrderHome
47 (top)/nodes/NodeA/servers/server1/plantsby/SuppliersHome
47                                     com.ibm.websphere.samples.plantsbywebsphereejb.SuppliersHome
48 (top)/nodes/NodeA/servers/server1/plantsby/ResetDBHome
48                                     com.ibm.websphere.samples.plantsbywebsphereejb.ResetDBHome
49 (top)/nodes/NodeA/servers/server1/plantsby/ReportGeneratorHome
49                                     com.ibm.websphere.samples.plantsbywebsphereejb.ReportGeneratorHome
50 (top)/nodes/NodeA/servers/server1/plantsby/CatalogHome
50                                     com.ibm.websphere.samples.plantsbywebsphereejb.CatalogHome
51 (top)/nodes/NodeA/servers/server1/plantsby/ShoppingCartHome
51                                     com.ibm.websphere.samples.plantsbywebsphereejb.ShoppingCartHome
52 (top)/nodes/NodeA/servers/server1/plantsby/SupplierHome
52                                     com.ibm.websphere.samples.plantsbywebsphereejb.SupplierHome
53 (top)/nodes/NodeA/servers/server1/plantsby/BackOrderStockHome
53                                     com.ibm.websphere.samples.plantsbywebsphereejb.BackOrderStockHome
54 (top)/nodes/NodeA/servers/server1/mail                             javax.naming.Context
55 (top)/nodes/NodeA/servers/server1/mail/PlantsByWebSphere
55                                     javax.mail.Session
56 (top)/nodes/NodeA/servers/server1/jta                             javax.naming.Context

```

```

57 (top)/nodes/NodeA/servers/server1/jta/usertransaction
57                                     java.lang.Object
58 (top)/nodes/NodeA/servers/server1/cell
58   Linked to context: ITS0Cell
59 (top)/nodes/NodeA/servers/server1/wm
59                                     javax.naming.Context
60 (top)/nodes/NodeA/servers/server1/wm/default
com.ibm.websphere.asynchbeans.WorkManager
61 (top)/nodes/NodeA/servers/MyClusterServer1
61   Linked to URL: corbaloc::wan:9811/NameServiceServerRoot
62 (top)/nodes/NodeA/servers/MyClusterServer2
62   Linked to URL: corbaloc::wan:9812/NameServiceServerRoot
63 (top)/nodes/NodeA/servers/nodeagent
63   Linked to URL: corbaloc::wan:2809/NameServiceServerRoot
64 (top)/clusters
64                                     javax.naming.Context
65 (top)/clusters/MyCluster
65   Linked to URL: corbaloc::wan:9811,:wan:9812/NameServiceServerRoot
66 (top)/legacyRoot
66                                     javax.naming.Context
66   Linked to context: ITS0Cell/persistent
67 (top)/persistent
67                                     javax.naming.Context
68 (top)/persistent/cell
68   Linked to context: ITS0Cell
69 (top)/cell
69                                     javax.naming.Context
69   Linked to context: ITS0Cell
70 (top)/deploymentManager
70   Linked to URL: corbaloc::wan:9809/NameServiceServerRoot
71 (top)/cells
71                                     javax.naming.Context

```

The process-level name space contains:

- ▶ A full set of entries for objects registered in the local name space of the process.
These entries include resources (JDBC, JMS, and so on) read from the resources.xml of the process, as well as those registered at run time by applications, for example, EJB homes.
- ▶ A corbaloc URL link to the local name space root (NameServiceServerRoot) of the node agent.
- ▶ A corbaloc URL link to the root of the deployment manager local name space (NameServiceServerRoot).
- ▶ A number of cross-links for the federated name space:
 - /cells
 - /clusters
 - /legacyRoot
- ▶ A link to the cell persistent root, /persistent/cell.
- ▶ A link to the node persistent root, /nodes/<nodename>/persistent.

11.3 Interoperable Naming Service (INS)

It is a requirement in J2EE 1.4 to provide a CosNaming service to support the EJB interoperability through the Interoperable Naming Service (INS). The INS allows J2EE application servers to deal with and understand names formulated according to the CORBA 2.3 naming scheme. The main advantage of INS is that it improves interoperability with other application server products, as well as CORBA servers. The naming architecture of WebSphere Application Server is compliant with the Interoperable Naming Service (INS). The requirements of INS CosNaming include:

- ▶ *corbaloc* and *corbaname* URLs must be supported, in addition to the IIOP URL supported in WebSphere Application Server V4.
 - Corbaloc designates an endpoint, such as a host machine.
 - Corbaname designates an object's name.
- ▶ The default bootstrap port must be 2809, as compared to the default of 900 used in earlier versions of IBM WebSphere Application Server.

11.3.1 Bootstrap ports

Every WebSphere Application Server V6 process has a bootstrap server and port assignment.

Each process on a given machine and WebSphere logical node requires unique ports, including the bootstrap port. The default port assignments are:

- ▶ Application server
The default for application server is 9810. Each subsequently created application server will be assigned a unique ascending port number that does not conflict.
- ▶ Network deployment
The default for node agent is 2809. Application servers are each assigned a unique non-default port, either explicitly by the administrator, or automatically determined by the administration tool.

11.3.2 CORBA URLs

CORBA URL syntax, both *corbaloc* and *corbaname*, is supported by IBM WebSphere Application Server.

corbaloc

The *corbaloc* form of the CORBA 2.3 URL has the following syntax:

`corbaloc:<protocol>:<addresslist>/<key>`

Table 11-1 shows the corbaloc options.

Table 11-1 corbaloc options

Setting	Description
protocol	The protocol used for the communication. Currently, the only valid value is iiop.
addresslist	List of one or more addresses (host name and port number). The addresses are separated by commas, and each address has a colon prefix.
key	Defines the type of root to access. See Table 11-5 on page 767 for further information.

The following list illustrates how the corbaloc URL can range from simple to complex, depending upon whether fault tolerance (request retry with second, third, and so on, server) is required:

- ▶ Basic
`corbaloc::myhost`
- ▶ Cell's name space root from a specific server
`corbaloc:iiop:1.2@myhost.raleigh.ibm.com:9344/NameServiceCellRoot`
- ▶ Server name space root with fault tolerance
`corbaloc::myhost1:9333,:myhost2:9333,:myhost2:9334/NameServiceServerRoot`

Note: corbaloc URLs are usually used for the provider URL when retrieving an InitialContext.

corbaname

A corbaname can be useful at times as a lookup name. If, for example, the target object is not a member of the federated name space and cannot be located with a qualified name, a corbaname can be a convenient way to look up the object.

The corbaloc form of the CORBA 2.3 URL has the following syntax:

`corbaname:<protocol>:<addresslist>/<key>#<INS string-formatted-name>`

Table 11-2 shows the corbaname options.

Table 11-2 corbaname options

Setting	Description
protocol	Use this protocol for the communication. Currently, the only valid value is iiop.
addresslist	This is a list of one or more addresses (host name and port number). The addresses are separated by commas, and each address has a colon prefix.
key	Define the type of root to access. See Table 11-5 on page 767 for details.
<INS string-formatted-name >	This is the fully qualified path to entry under the specific root context.

The following examples illustrate how the corbaname URL can range from simple to complex, depending upon whether fault tolerance, request retry with second, third, and so on. server, is required.

► Fully qualified name access

```
corbaname::myhost:9333#cell/nodes/node1/servers/server5/someEjb
```

► Object access through a specific server root

```
corbaname::myhost:9333/NameServiceServerRoot/someEjb
```

Note: corbaname URLs are usually used when performing a direct URL lookup using a previously obtained InitialContext, for example, `ic.lookup("urlstring")`.

11.4 Distributed CosNaming

One of the advantages of the distributed nature of CosNaming in WebSphere Application Server is that it removes the bottleneck of having a single name server for all naming lookups. Each WebSphere process, such as deployment manager, node agent and application server, hosts its own ORB, NameService and local name space. Lookups are made by accessing the NameService in the most convenient process. They are not bottlenecked through a single server process in the cell.

The WebSphere Application Server naming architecture uses CORBA CosNaming as its foundation. The CosNaming architecture has been changed to support a distributed and federated collection of CosNaming servers. Each deployment manager, node agent, and application server is a CosNaming server

and is responsible for managing the names of the objects that are bound locally. Objects are bound into the local context. Lookups start in the local process and end in the process where the target object is located. This reduces the dependency of the WebSphere Application Server network on a single name server for all lookups.

A single, logical name space exists across the cell. The separate name spaces of each server process are linked and federated via context links in the cell name space. It is possible to navigate to specific subcontexts, as every server cluster and non-clustered server has its own context stored in a level of the cell name space.

The contents of the federated name space are mostly transient, built from configuration data read from the XML configuration files on the startup of each server process. Persistent roots are provided at the cell and node level of the name space to provide locations where objects can be persistently bound. These bindings are persisted to XML files on the file system.

Each separate server process has its own bootstrap port, thereby reducing bottlenecks.

11.5 Configured bindings

With the configured bindings feature, you can add objects to the name space using the administrative interfaces. This feature allows an administrator to explicitly add bindings to the cell name space without having to write code. The administrator configures an alias in a persistent name space that refers to a real reference in one of the local name spaces, thus providing an additional level of indirection for names. (The configuration details are covered later in 11.12.1, “Name space bindings” on page 785.)

The functionality is useful in these areas:

- Federation of name spaces

As long as it is CORBA 2.3 compliant, supporting INS, the name space of other WebSphere Application Server V6 or V5 cells, WebSphere Application Server V4 administrative domains, third-party application servers and even CORBA servers can be federated into the cell's name space.

- Interoperability with WebSphere Application Server V4

The default context of WebSphere Application Server V4 clients is the global, or legacy, context. However, WebSphere Application Server V6 processes bind their objects in local, transient name spaces. Therefore, WebSphere Application Server V4 clients looking up and accessing objects in WebSphere

Application Server V6 without requiring changes to the client requires the WebSphere Application Server V6 object to be bound to the legacy name space accessible to the client. Enter configured bindings. An alias can be configured into the legacy name space. When used by the WebSphere Application Server V4 client, the client is transparently redirected to the real object reference in one of the cell's local name spaces.

11.5.1 Types of objects

The following types of objects can be bound using configured bindings:

- ▶ EJB hosted by a server in the cell

The configured binding identifies an EJB home based on its configured JNDI name and the server in which it is deployed.

A possible use of this is to put a binding for an EJB into the cell-scoped name space so that a lookup can be done without knowledge about the server in which the EJB is deployed. This mechanism is useful for allowing WebSphere Application Server V4 clients to look up WebSphere Application Server V6 EJBs without having to redeploy.

- ▶ CORBA object

The configured binding identifies a CORBA object bound somewhere in this or another name space by using a corbaname URL string. Included is also an indicator of whether the object is a CosNaming NamingContext, in which case the binding is a federated link from one name space to another.

- ▶ JNDI name

The configured binding identifies a provider URL and a JNDI name that can be used to look up an object. This can be used to reference a resource or other Java serialized object bound elsewhere in this name space or another name space.

- ▶ String constant

The string constant can be used to bind environment data into the name space.

11.5.2 Types of binding references

There are several different references that can be specified for configured bindings. Valid types are summarized in Table 11-3.

Table 11-3 Types of binding reference

Binding type	Required settings
EJB (EjbNameSpaceBinding)	1. The binding identifier is the name that uniquely identifies this configured binding. 2. The name in name space is relative to the configured root. 3. The JNDI is the name of EJB. 4. Use the server or server cluster where the EJB is deployed.
CORBA (CorbaObjectNameSpaceBinding)	1. The binding identifier is the name that uniquely identifies this configured binding. 2. The name in name space is relative to configured root. 3. Use the corbaname URL. 4. It is an indicator if the target object is a federated context object, or a leaf node object.
Indirect (IndirectLookupNameSpaceBinding)	1. The binding identifier is the name that uniquely identifies this configured binding. 2. The name in name space is relative to configured root. 3. Use the Provider URL. 4. Use the JNDI name of object.
String (StringNameSpaceBinding)	1. The binding identifier is the name that uniquely identifies this configured binding. 2. The name in name space is relative to configured root. 3. Set the constant string value.

The configured bindings can be relative to one of the following context roots:

- ▶ Server root
- ▶ Node persistent root
- ▶ Cell persistent root

11.6 Initial contexts

In WebSphere, an initial context for a name server is associated with a bootstrap host and bootstrap port. These combined values can be viewed as the address of the name server owning the initial context. To get an initial context, you must know the bootstrap host and port for the initial context's name server.

JNDI clients should assume the correct environment is already configured, so there is no need to explicitly set property values and pass them to the `InitialContext` constructor.

However, a JNDI client might need to access a name space other than the one identified in its environment. In this case, it is necessary to explicitly set the `javax.naming.provider.url` (provider URL) property used by the `InitialContext` constructor. A provider URL contains bootstrap server information that the initial context factory can use to obtain an initial context. Any property values passed directly to the `InitialContext` constructor take precedence over settings of those same properties found elsewhere in the environment.

Two provider URL forms can be used with WebSphere's initial context factory:

- ▶ CORBA object URL
- ▶ IIOP URL

CORBA object URLs are more flexible than IIOP URLs and are the recommended URL format to use. CORBA object URLs are part of the OMG CosNaming Interoperable Naming Specification. The IIOP URLs are the JNDI format, but are still supported by the WebSphere initial context factory. The examples in the following sections illustrate the use of these URLs.

Using a CORBA object URL

An example of using a corbaloc URL with a single address to obtain an initial context is shown in Example 11-4.

Example 11-4 Initial context using CORBA object URL

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;

Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc:iiop:myhost.mycompany.com:2809");

Context initialContext = new InitialContext(env);
```

Using a CORBA object URL with multiple addresses

CORBA object URLs can contain more than one bootstrap server address. This feature can be used in WebSphere when attempting to obtain an initial context from a server cluster. The bootstrap server addresses for all servers in the cluster can be specified in the URL. The operation will succeed if at least one of the servers is running, eliminating a single point of failure.

Note: There is no guarantee of any particular order in which the address list will be processed. For example, the second bootstrap server address might be used to obtain the initial context even though the first bootstrap server in the list is available.

An example of using a corbaloc URL with multiple addresses to obtain an initial context is shown in Example 11-5 on page 764.

Example 11-5 Initial context using CORBA object URL with multiple addresses

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;

Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL,
"corbaloc::myhost1:2809,:myhost2:2809,:myhost3:2809");

Context initialContext = new InitialContext(env);
```

Using a CORBA object URL from a non-WebSphere JNDI

To access a WebSphere name server from a non-WebSphere environment, such that the WebSphere initial context factory is not used, a corbaloc URL must be used that has an object key of *NameServiceServerRoot* to identify the server root context.

The server root is where system artifacts such as EJB homes are bound. The default key of NameService can be used when fully qualified names are used for JNDI operations.

Example 11-6 shows a CORBA object type URL from a non-WebSphere JNDI implementation. It assumes full CORBA object URL support by the non-WebSphere JNDI implementation.

Example 11-6 Using a CORBA object URL from non-WebSphere JNDI

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;

Hashtable env = new Hashtable(); env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.somecompany.naming.TheirInitialContextFactory");
env.put(Context.PROVIDER_URL,
"corbaname:iiop:myhost.mycompany.com:2809/NameServiceServerRoot");

Context initialContext = new InitialContext(env);
```

Using an IIOP URL

The IIOP type of URL is a existing format that is not as flexible as CORBA object URLs. However, URLs of this type are still supported by the WebSphere initial context factory.

Example 11-7 shows an IIOP type URL as the provider URL.

Example 11-7 Initial context using an IIOP URL

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;

Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "iiop://myhost.mycompany.com:2809");

Context initialContext = new InitialContext(env);
```

11.6.1 Setting initial root context

Each server contains its own server root context. When bootstrapping to a server, the server root is the default initial JNDI context. Most of the time, this is the desired initial context, because system artifacts such as EJB homes are bound at this point. However, other root contexts exist that might contain bindings of interest. It is possible to specify a provider URL to select other root contexts.

Note: The default is that the name will be resolved based upon the context associated with the server bootstrap to which the client is connected.

The initial root context can be selected using the following settings:

- ▶ CORBA object URL
- ▶ Name space root property

Default initial context

The default initial context depends on the type of client. Table 11-4 summarizes the different categories of clients and the corresponding default initial context.

Table 11-4 Default initial context versus client type

Client type	Description	Default initial context
WebSphere Application Server V6 or V5 JNDI	EJB applications use the JNDI interface to perform name space lookups. WebSphere clients by default use the WebSphere's CosNaming JNDI plug-in implementation.	Server root
WebSphere Application Server V4 JNDI	WebSphere clients running in releases prior to V5 by default use the WebSphere's V4 CosNaming JNDI plug-in implementation.	Cell persistent root (legacy root)
Other JNDI	Some applications might perform name space lookups with a non-WebSphere CosNaming JNDI plug-in implementation.	Cell root
CORBA	Standard CORBA client obtains an initial org.omg.CosNaming.NamingContext reference with the key NamingContext.	Cell root

Selecting initial root context with a CORBA object URL

There are several object keys registered with the bootstrap server that you can use to select the root context to be used as the initial context. To select a particular root context with a CORBA object URL object key, set the object key to the corresponding value. The default object key is NameService. Using JNDI, this will yield the server root context.

Table 11-5 lists the different root contexts and their corresponding object key.

Table 11-5 CORBA object URL root context values

Root context	CORBA object URL object key	Description
Server root	NameServiceServerRoot	Server root for the accessed server
Cell persistent root	NameServiceCellPersistentRoot	The persistent cell root for the accessed server
Cell root	NameServiceCellRoot	The cell root for the accessed server
Node root	NameServiceNodeRoot	The node root for the accessed server

Note: The name server running in the deployment manager process has no node root registered under the NameServiceNodeRoot key, because there is no node agent, nor application servers, running in its node.

Example 11-8 shows the use of a corbaloc URL with the object key set to select the cell persistent root context as the initial context.

Example 11-8 Select cell persistent root context using corbaloc URL

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;

Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL,
"corbaloc:iiop:myhost.mycompany.com:2809/NameServiceCellPersistentRoot");

Context initialContext = new InitialContext(env);
```

Selecting initial root context with name space root property

You can select the initial root context by passing a name space root property setting to the InitialContext constructor. Generally, the object key setting is sufficient.

Sometimes, a property setting might be preferable. For example, the root context property can be set on the Java invocation to make it transparent to the application which server root is being used as the initial context. The default server root property setting is *defaultroot*, which will yield the server root context.

Tip: If a simple name is used, the root context that will be assumed can be set by passing the `com.ibm.websphere.naming.namespaceroot` property to InitialContext.

Table 11-6 Name space root values

Root context	CORBA object URL object key
Server root	bootstrapserverroot
Cell persistent root	cellpersistentroot
Cell root	cellroot
Node root	bootstrapnoderoot

The name space root property is used to select the default root context only if the provider URL does not contain an object key or contains the object key, *NameService*. Otherwise, the property is ignored.

Example 11-9 shows use of the name space root property to select the cell persistent root context as the initial context.

Tip: WebSphere makes available constants that can be used instead of hard-coding the property name and value, for example:

```
env.put(PROPS.NAME_SPACE_ROOT, PROPS.NAME_SPACE_ROOT_CELL_PERSISTENT);
```

Example 11-9 Use of name space root property to select cell persistent root context

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
import com.ibm.websphere.naming.PROPS;
```

```
Hashtable env = new Hashtable();
```

```
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc:iiop:myhost.mycompany.com:2809");
env.put(Props.NAME_SPACE_ROOT, Props.NAME_SPACE_ROOT_CELL_PERSISTENT);
```

```
Context initialContext = new InitialContext(env);
```

11.7 Federation of name spaces

Federating name spaces involves binding contexts from one name space into another name space. In a WebSphere Application Server V6 name space, federated bindings can be created with the following restrictions:

- ▶ Federation is limited to CosNaming name servers. A WebSphere name server is a CORBA CosNaming implementation.
Federated bindings to other CosNaming contexts can be created, but bindings to LDAP name server implementation contexts cannot.
- ▶ If JNDI is used to federate the name space, the WebSphere initial context factory must be used to obtain the reference to the federated context. If any other initial context factory implementation is used, the binding might not be created, or the level of transparency might be reduced.
- ▶ A federated binding to a non-WebSphere naming context has the following functional limitations:
 - JNDI operations are restricted to the use of CORBA objects. For example, EJB homes can be looked up, but non-CORBA objects such as data sources cannot.
 - JNDI caching is not supported for non-WebSphere name spaces. This only affects the performance of lookup operations.
- ▶ Do not federate two WebSphere single server name spaces. If this is done, incorrect behavior can result. If you require federation of WebSphere name spaces, then servers running under IBM WebSphere Application Server Network Deployment are required.

In the example in Figure 11-3, assume that a name space, Namespace 1, contains a context under the name a/b. Also assume that a second name space, Namespace 2, contains a context under the name x/y. If context x/y in Namespace 2 is bound into context a/b in Namespace 1 under the name f2, the two name spaces are federated. Binding f2 is a federated binding because the context associated with that binding comes from another name space. As shown in Figure 11-3, from Namespace 1, a lookup of the name a/b/f2, would return the context bound under the name x/y in Namespace 2. Furthermore, if context x/y contained an EJB home bound under the name ejb1, the EJB home could be

looked up from Namespace1 with the lookup name `a/b/f2/ejb1`. Notice that the name crosses name spaces. This fact is transparent to the naming client.

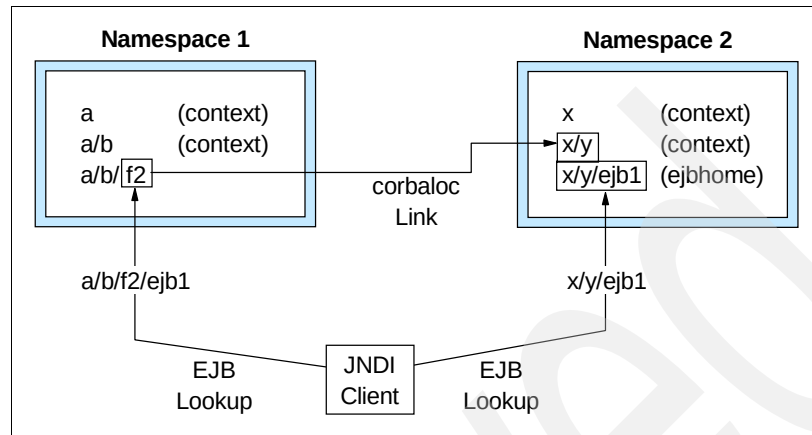


Figure 11-3 JNDI access using federated name spaces

11.8 Foreign cell bindings

New in V6.1: If you have applications in a cell that access other applications in another cell, you can configure a foreign cell name binding for the other cell.

A *foreign cell binding* is a context binding that resolves to the cell root context of another cell. You can configure a foreign cell binding such that applications in your cell can access applications and other resources within the other cell.

Foreign cell bindings consist of one or more bootstrap addresses for a given cell. The bootstrap address consists of the name of the bootstrap host and a port number that it listens on. This concept places the bootstrap addresses in a single location, instead of every reference to the foreign cell in the application deployment data. If the bootstrap information changes, this can be updated in a central location, just once via the administration console. No applications need to be updated, as all objects in the foreign cell are looked up through the central bindings.

Note: The foreign cell and the local cell must have different names.

Figure 11-4 on page 771 depicts an example, where we have a cell “CellA” that has a cell-scoped EJB name space binding configured with a name in the name

space /ejb/AccountHome. Applications running in CellA would look up the home with a JNDI name of cell/persistent/ejb/AccountHome (actually J2EE applications would use a java:comp/env name that maps to that JNDI name through the application deployment descriptors). With a foreign cell binding configured in CellB that points to CellA, applications in CellB can do the same lookup with a JNDI name of cell/cells/CellA/persistent/ejb/AccountHome.

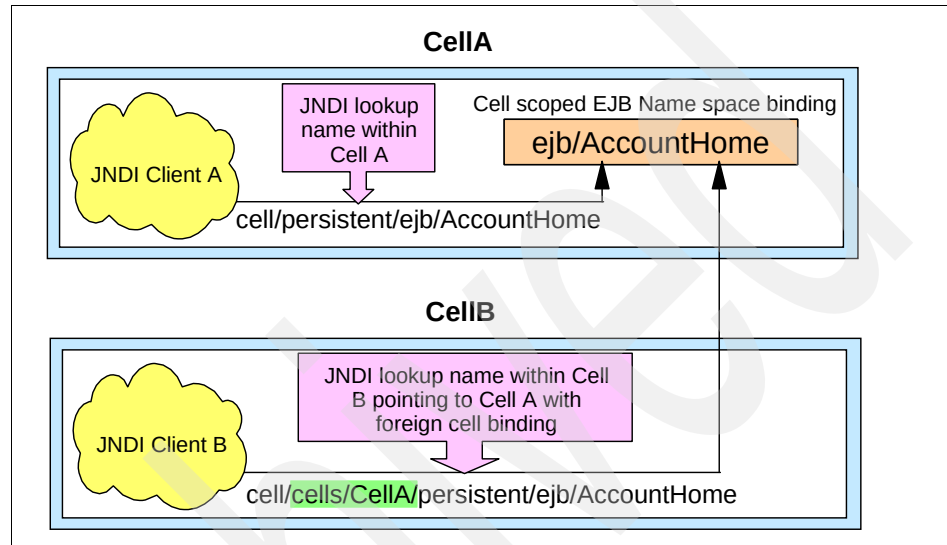


Figure 11-4 JNDI lookup names using a foreign cell binding

11.9 Interoperability

The name space in IBM WebSphere Application Server V5 is the same as in WebSphere Application Server V6. Thus, an EJB client running on IBM WebSphere Application Server V5 accessing EJB applications running on WebSphere Application Server V6 will have no interoperability issues.

WebSphere Application Server V6 provides the following support for interoperating with previous releases of WebSphere and with non-WebSphere JNDI clients:

- ▶ EJB clients running on WebSphere V4.0.x, accessing EJB applications running on WebSphere Application Server V6
- ▶ EJB clients running on WebSphere Application Server V6, accessing EJB applications running on WebSphere V4.0.x servers
- ▶ EJB clients running in an environment other than WebSphere, accessing EJB applications running on WebSphere Application Server V6 servers

11.9.1 WebSphere V4.0 EJB clients

Applications migrated from previous WebSphere releases can still have clients running in a previous release. The default initial JNDI context for EJB clients running on previous versions of WebSphere is the cell persistent root (the legacy root). However, the home for an EJB deployed in V6.0 is bound to the server root context. For the EJB lookup name for down-level clients to remain unchanged, configure a binding for the EJB home under the cell persistent root.

The following options enable interoperability with WebSphere Application Server V4 clients:

- ▶ Set the client's default initial context to legacyRoot. This option is equivalent to the cell persistent root of WebSphere Application Server V6.
- ▶ Redeploy the clients using the Application Server Toolkit so that the JNDI names can be fixed to reflect the real, fully qualified names in the WebSphere Application Server V6 name space.
- ▶ Use aliases for the names the clients look up. These transparently redirect to the correct object in the WebSphere Application Server V6 name space. This option uses configured bindings.

Options for EJB lookup

The following options support EJB lookup from a WebSphere Application Server V4 client to a WebSphere Application Server V6 hosted EJB:

- ▶ Redeploy the WebSphere Application Server V4 client.
Update the <ejb-ref> to reflect the WebSphere Application Server V6 compatible JNDI name.
- ▶ In WebSphere Application Server V6, configure EjbNamespaceBinding:
 - a. Use the same JNDI name looked up by the WebSphere Application Server V4 client.
 - b. Identify the JNDI name and server, or cluster, of the target EJB.
 - c. Configure the binding in the cell persistent root.

Options for resources bound in external name space

Options for resources bound in external name spaces include the following:

- ▶ Redeploy the WebSphere Application Server V4 client.
Update the <resource-ref> to reflect the WebSphere Application Server V6 compatible JNDI name.
- ▶ In WebSphere Application Server V6, run the program to bind the resource into the WebSphere Application Server V6 cell persistent root.

- ▶ In WebSphere Application Server V6, configure IndirectLookupNameSpaceBinding by doing the following:
 - a. Use the same JNDI name looked up by the WebSphere Application Server V4 client.
 - b. Specify the provider URL and JNDI name of the name space where the resource is already bound (a WebSphere Application Server V4 name space).
 - c. Configure the binding in the cell persistent root.

11.9.2 WebSphere V4.0 server

The default initial context for a WebSphere V4.0 server is the correct context. WebSphere Application Server V6 clients simply look up the JNDI name under which the EJB home is bound.

11.9.3 EJB clients hosted by non-WebSphere environment

When an EJB application running in WebSphere Application Server V6 is accessed by a non-WebSphere EJB client, the JNDI initial context factory is presumed to be a non-WebSphere implementation. In this case, the default initial context is the cell root. If the JNDI service provider being used supports CORBA object URLs, use the corbaname format shown in Example 11-10 to look up the EJB home.

Example 11-10 corbaname format for EJB home lookup

```
initialContext.lookup("corbaname:iiop:myHost:2809#cell/clusters/myCluster/myEJB");
```

According to the URL in Example 11-10, the bootstrap host and server (node agent) port are myHost and 2809. The EJB is installed in a server cluster named myCluster. The EJB is bound in that cluster under the name myEJB.

Note: The server name could also be the name of a non-clustered server. This form of lookup works in the following situations:

- ▶ With any name server bootstrap host and port configured in the same cell
- ▶ If the bootstrap host and port belong to a member of the cluster itself

To avoid a single point of failure, the bootstrap server host and port for each cluster member could be listed in the URL, as shown in Example 11-11.

Example 11-11 corbaname format with multiple addresses for EJB home lookup

```
initialContext.lookup("corbaname:iiop:host1:9810,host2:9810#cell/clusters/myCluster/myEJB");
```

The name prefix `cell/clusters/<clustername>/` is not necessary if bootstrapping to the cluster itself, but it always works. The prefix is required, however, when looking up EJBs in other clusters. The server binding for the prefix used to access another cluster is implemented in a way that avoids a single point of failure during a lookup.

If the JNDI initial context factory you use does not support CORBA object URLs, the initial context can be obtained from the server, and the lookup can be performed on the initial context, as shown in Example 11-12.

Example 11-12 corbaname format with multiple addresses for EJB home lookup

```
Hashtable env = new Hashtable();  
env.put(CONTEXT.PROVIDER_URL, "iiop://myHost:2809");  
Context ic = new InitialContext(env);  
Object o = ic.lookup("cell/clusters/myCluster/myEJB");
```

This form of lookup works from any server in the same cell as the EJB home being looked up. However, this approach does not allow multiple hosts and ports to be specified in the provider URL and does not incorporate the availability advantages of a `corbaloc` or `corbaname` URL with multiple hosts and ports belonging to the server cluster members.

11.10 Examples

The following examples highlight a number of different server topologies and the effect the topologies have on the use of the Naming Service:

- ▶ Single server
- ▶ Single server with a non-default port
- ▶ Two single servers on the same box
- ▶ Two Network Deployment application servers on the same box
- ▶ WebSphere Application Server V4 client

11.10.1 Single server

In the single-server environment, the naming functionality works in exactly the same way as in WebSphere Application Server V4. There is only one server and only one root context and, therefore, no ambiguity in the location of a named object. This is illustrated by the example in Figure 11-5.

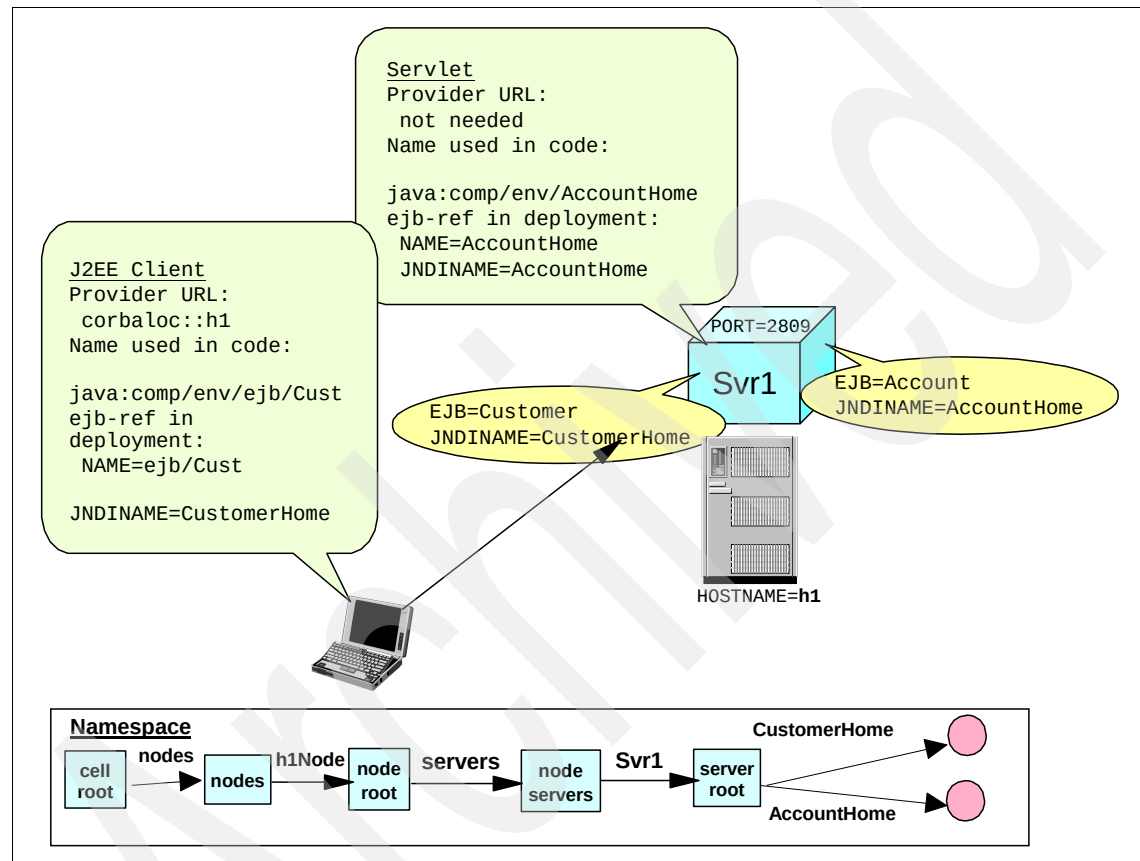


Figure 11-5 Single server

By accessing the server root directly, the J2EE or servlet client does not need to traverse the cell name space (cell root → servers → server root → object).

Note: Even in a single-server case, clients can still use the fully qualified JNDI name to look up an object. This removes any dependency on the particular topology. However, there is a small performance degradation.

In a single-server, the server root acts as the default bootstrap, and should be assigned port 2809. Clients external to the server process using the provider URL do not need a port number.

If the named object is looked up by a client running in the same process, then a provider URL, and corbaloc, is not needed. By default, the lookup is performed against the local process name space. Table 11-7 illustrates the Provider URL.

Table 11-7 Lookup settings required for a single server

Component	Provider URL	JNDI name
Servlet (same process)	<i>Not needed</i>	CustomerHome
J2EE client (external process)	corbaloc::<hostname>	CustomerHome

11.10.2 Two single servers on the same box

When more than one instance of the application server runs on a single machine, then you must configure each server's bootstrap to run on a different port. In this case, you can have a J2EE component in one server looking up objects in the other server. This is illustrated by the example in Figure 11-6.

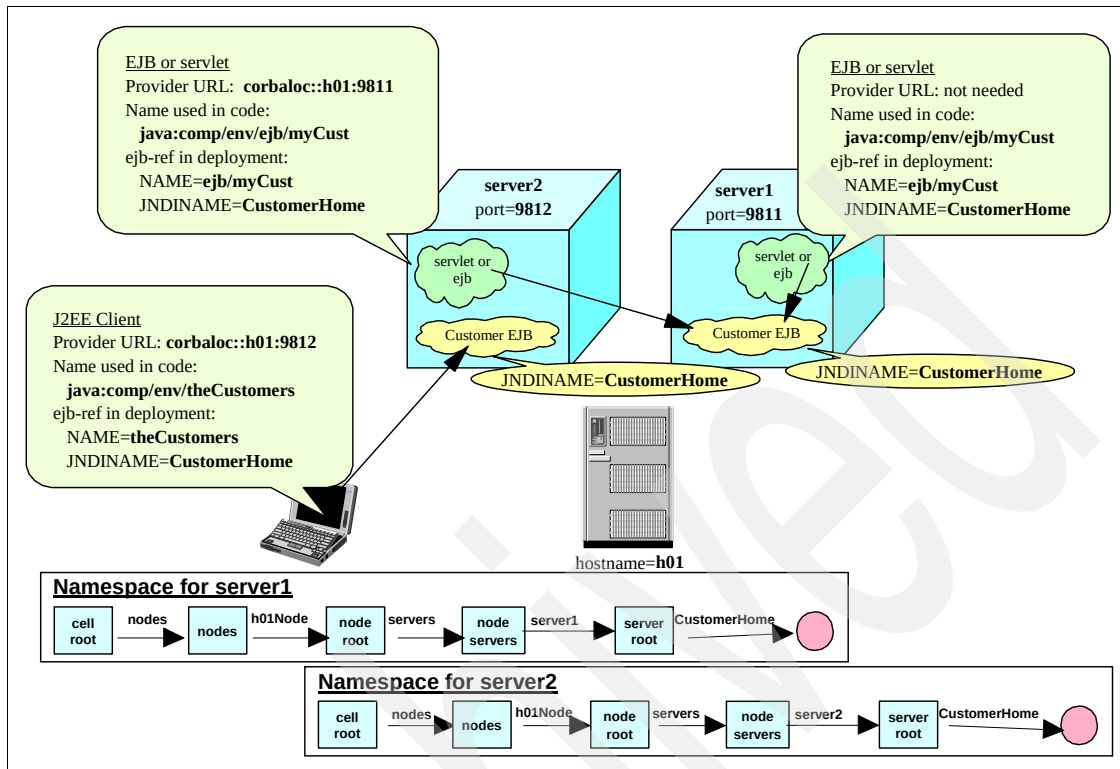


Figure 11-6 Two single servers on the same box

Because each application server name space is separate, the different objects can use the same name, CustomerHome. There is no name collision. The fully qualified JNDI name can be used to uniquely identify the name registered in one server from the name in another. If objects are registered under the name CustomerHome on two servers, look up the name using:

cell/nodes/<nodename>/servers/<server1>/CustomerHome
 cell/nodes/<nodename>/servers/<server2>/CustomerHome

Table 11-8 illustrates the required Provider URL settings.

Table 11-8 Lookup settings for two single servers on the same box

Component	Provider URL	JNDI name
Servlet (same process)	<i>Not needed</i>	CustomerHome
Servlet (external process)	corbaloc::<hostname>:<port#>	CustomerHome
J2EE client (external process)	corbaloc::<hostname>:<port#>	CustomerHome

11.10.3 Network Deployment application servers on the same box

The configuration becomes more complex when we move from an stand-alone server environment to a Network Deployment distributed server environment. In this topology, there can be separate application servers as well as a node agent process, all of which have a bootstrap port and host a local name space:

- ▶ The node agent is the default bootstrap for the node, and has its bootstrap port configured on 2809.
- ▶ The application servers are not the default bootstrap, and, therefore, each is configured to use a non-default bootstrap port.

This concept is illustrated by the example in Figure 11-7 on page 778.

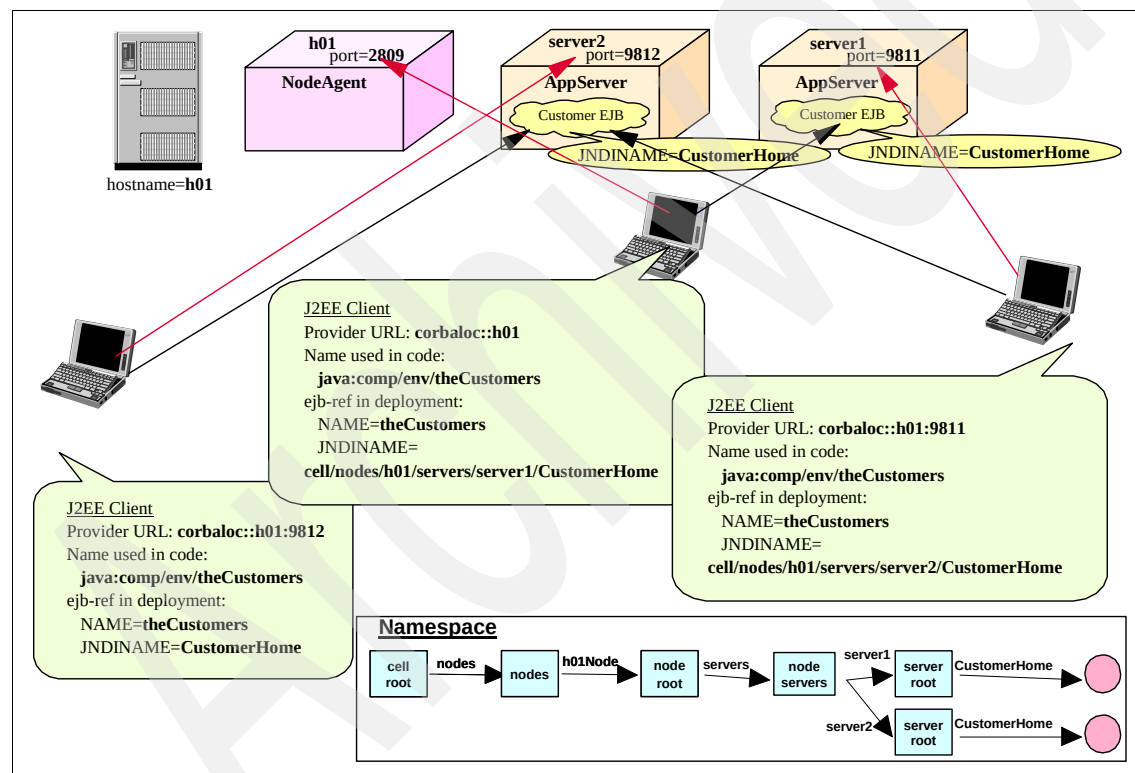


Figure 11-7 Two network deployment application servers on the same box

Unless a client uses a specific application server in its provider URL, the lookup is performed on the node agent. For the lookup to succeed, the bindings have to specify the fully qualified name of the object:

`cell/nodes/<nodename>/servers/<servername>/<name of object>`

That is, the client needs to specify where the object is located. This is a big difference from the behavior in WebSphere Application Server V4, where all named objects were registered in a single global name space.

Tip: When you need server clusters for high availability, bootstrap to a server cluster so that the initial context has failover support. Lookups that resolve to other clusters from that bootstrap cluster also have failover support from the name server implementation. The provider URL should have the bootstrap address of each cluster member to avoid a single point of failure when obtaining the initial context.

In a distributed server environment, choose a bootstrap server that has a stable bootstrap address, such as a designated cluster, server, or node agent.

Table 11-9 illustrates the Provider URL settings required.

Table 11-9 Lookup settings for two Network Deployment servers on the same box

Component	Provider URL	JNDI name
Servlet (same process)	<i>Not needed</i>	CustomerHome
Servlet (external process accessing local name space to access local object)	<i>Not needed</i>	cell/nodes/<nodename>/servers/<server2>/CustomerHome
Servlet (external process accessing other appserver's name space to access object on that appserver)	corbaloc::<appserver2 hostname>:<port#>	CustomerHome
	<i>(or) Not needed</i>	cell/persistent/CustomerHome2 ¹
J2EE client (external process accessing appserver1 with object located on appserver1)	corbaloc::<appserver hostname>:<port#>	CustomerHome
	<i>(or) Not needed</i>	cell/persistent/CustomerHome1 ¹
J2EE client (external process accessing node agent)	corbaloc::<node agent hostname>	cell/nodes/<nodename>/servers/<server2>/CustomerHome
	<i>(or) Not needed</i>	cell/persistent/CustomerHome2 ¹

Component	Provider URL	JNDI name
J2EE client (external process accessing appserver1 with object located on appserver2)	corbaloc::<appserver1 hostname>:<port#>	cell/nodes/<nodename>/servers/<server2>/CustomerHome
	<i>(or) Not needed</i>	cell/persistent/CustomerHome ¹
¹ You must manually configure indirect JNDI references to the respective EJB in the cell/persistent name space.		

11.10.4 WebSphere Application Server V4 client

In WebSphere Application Server V4, there is no need to specify a path to a named object, because all objects are registered in a single global name space. Although convenient, this causes naming conflicts because no two objects can be registered across all application servers with the same names.

The use of configured bindings, aliases, in the cell persistent root provides a mechanism by which the V4 naming structure can be mapped to the fully qualified names of V6. This is illustrated in Figure 11-8.

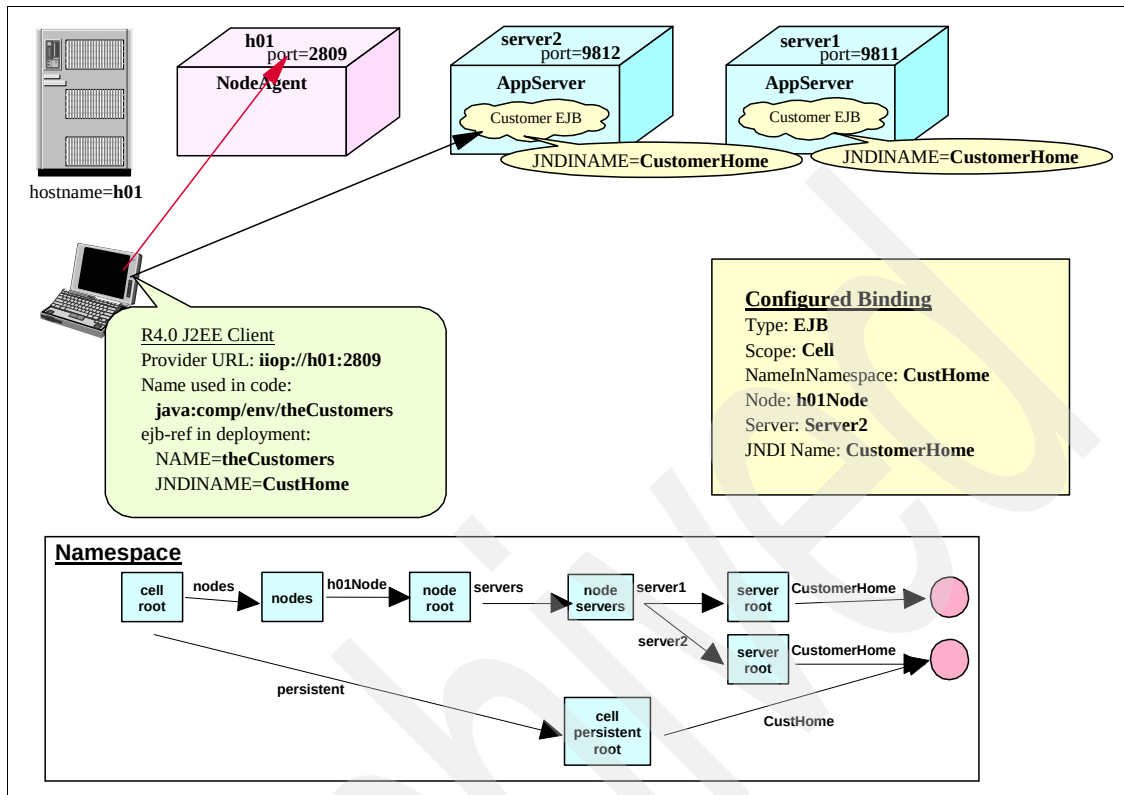


Figure 11-8 WebSphere Application Server V4 client

Table 11-10 illustrates the Provider URL settings required.

Table 11-10 Lookup settings for WebSphere Application Server V4 client interoperability

Component	Provider URL	JNDI name
V4 client	<code>iiop://<hostname>:2809</code>	CustHome

CustHome is the name registered in the cell-level persistent root (the legacy root), for `cell/nodes/<nodename>/servers/<servername>/CustomerHome`.

The WebSphere Application Server V4 client accesses the JNDI alias registered in the cell persistent root of the WebSphere Application Server V6 cell. The WebSphere Application Server V6 run time *transparently* redirects the client to the JNDI entry located in a specific local name space hosted by one of the name servers of the cell.

11.11 Naming tools

IBM WebSphere Application Server provides the following tools for the support of the naming architecture.

11.11.1 dumpNameSpace

Run the **dumpNameSpace** command against any bootstrap port to get a listing of the names bound with that provider URL.

The output of the command:

- ▶ Does not present a full logical view of the name space
- ▶ Shows CORBA URLs where the name space transitions to another server

The tool indicates that certain names point to contexts external to the current server and its name space. The links show the transitions necessary to perform a lookup from one name space to another.

Tip: An invocation of the **dumpNameSpace** command cannot generate a dump of the entire name space, only the objects bound to the bootstrap server and links to other local name spaces that compose the federated name space. Use the correct host name and port number for the server to be dumped.

Syntax

To run the **dumpNameSpace** command, type the following:

```
dumpNameSpace [options]
```

All arguments are optional. Table 11-11 on page 782 shows the available options.

Table 11-11 Options for dumpNameSpace

Option	Description
-host <hostname>	This option is the host name of bootstrap server. If it is not defined, then the default is localhost.
-port <portnumber>	This option is the bootstrap server port number. If it is not defined, then the default is 2809.
-factory <factory>	This option is the initial context factory to be used to get initial context. The default of com.ibm.websphere.naming.WsnInitialContextFactory is okay for most use.

Option	Description
-root [cell server node host legacy tree default]	WebSphere V5.0 or later ▶ cell: dumpNameSpace default. Dump the tree starting at the cell root context. ▶ server: Dump the tree starting at the server root context. ▶ node: Dump the tree starting at the node root context. (Synonymous with "host") WebSphere V4.0 ▶ legacy: dumpNameSpace default. Dump the tree starting at the legacy root context. ▶ host: Dump the tree starting at the bootstrap host root context (Synonymous with node) ▶ tree: Dump the tree starting at the tree root context. All WebSphere and other name servers ▶ default: Dump the tree starting at the initial context that JNDI returns by default for that server type. This is the only -root choice that is compatible with WebSphere servers prior to V4.0 and with non-WebSphere name servers.
-url <url>	This option is the value for the java.naming.provider.url property used to get the initial JNDI context. This option can be used in place of the -host, -port, and -root options. If the -url option is specified, the -host, -port, and -root options are ignored.
-startAt <context>	This option is the path from the requested root context to the top level context where the dump should begin. Recursively dumps subcontexts below this point. Defaults to empty string, that is, root context requested with the -root option.
-format <format>	▶ jndi: Display name components as atomic strings. ▶ ins: Display name components parsed against INS rules (id.kind). The default format is jndi.
-report <length>	▶ short: Dumps the binding name and bound object type, essentially what JNDI Context.list() provides. ▶ long: Dumps the binding name, bound object type, local object type, and string representation of the local object. In other words, IORs, string values, and so on, are printed. The default report option is short.
-traceString <tracespec>	Trace string of the same format used with servers, with output going to the file DumpNameSpaceTrace.out.
-help or -?	Prints a usage statement.

Finding the bootstrap address

To find the bootstrap address for node agents, servers, and the cell, do the following:

- ▶ For application servers, click **Servers** → **Application Servers**. Click the server to open the configuration. Select **Ports** from the Communications section, then **BOOTSTRAP_ADDRESS**.
- ▶ For node agents, click **System Administration** → **Node Agents**. Select the node agent to open the configuration. Select **Ports** from the Additional Properties section, then **BOOTSTRAP_ADDRESS**.
- ▶ For the cell, click **System Administration** → **Deployment Manager**. Select **Ports** from the Additional Properties section, then **BOOTSTRAP_ADDRESS**.

To find the `dumpNameSpace` usage, see Example 11-13.

Example 11-13 dumpNameSpace usage

```
$ cd c:\ibm\was60\AppServer\bin
```

Get help on options:

```
$ dumpNameSpace -?
```

Dump server on localhost:2809 from cell root:

```
$ dumpNameSpace
```

Dump server on localhost:2806 from cell root:

```
$ dumpNameSpace -port 2806
```

Dump server on yourhost:2811 from cell root:

```
$ dumpNameSpace -port 2811 -host yourhost
```

Dump server on localhost:9810 from server root:

```
$ dumpNameSpace -root server*
```

Dump server at corbaloc

```
dumpNameSpace -url corbaloc:iiop:yourhost:901
```

11.12 Configuration

This section discusses how to configure a name binding for an enterprise bean, a CORBA CosNaming naming context or CORBA leaf node object, an object that can be looked up using JNDI, or a constant string value using the administrative console.

11.12.1 Name space bindings

The configured bindings feature allows objects to be added to the name space using the administrative console. An administrator can now explicitly add bindings to the cell name space without having to write code. With this feature, an administrator can configure an alias in a persistent name space for a reference in one of the local name spaces.

Name space bindings can be created for the following four object types:

- ▶ String
- ▶ EJB
- ▶ CORBA
- ▶ Indirect

As an example, look at Figure 11-8. In this scenario, an alias is configured to allow an application using the WebSphere V4 naming style to access an EJB while running on WebSphere V6. Because the V4 application code does not specify a path to the named object, a binding is added to the cell persistent root to redirect the client to the JNDI entry in the local name space.

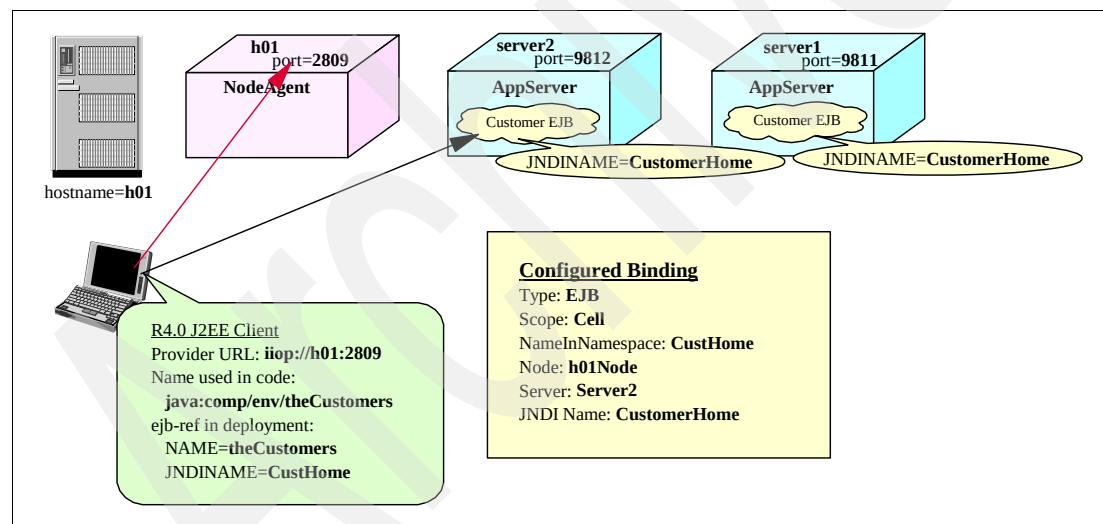


Figure 11-9 WebSphere Application Server V4 client

To create the binding, do the following:

1. Select **Environment** → **Naming** → **Name Space Bindings**.
2. Set the scope to cell.
3. Click **New**. See Figure 11-10.

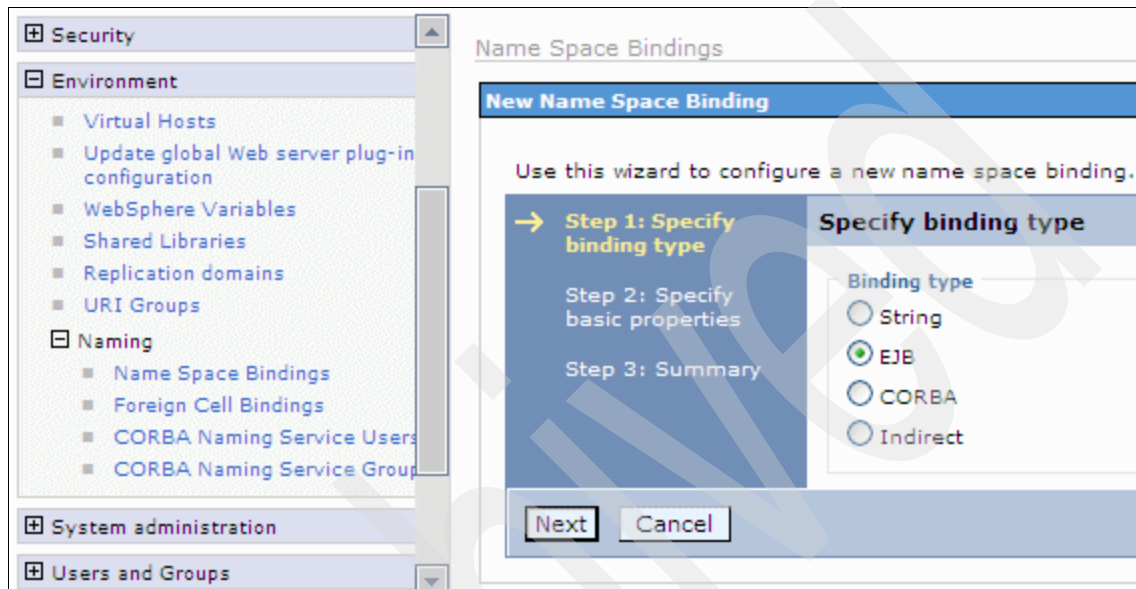


Figure 11-10 Name Space Binding window

4. Choose **EJB** and click **Next**.
5. Enter the values shown in Figure 11-11.
 - Binding identifier is a unique identifier for the binding.
 - Name in Name Space matches the JNDI name used in the application code.
 - Enterprise Bean Location is the cluster or node where the EJB resides.
 - Server is the name of the server where the EJB resides.
 - JNDI Name is the JNDI name of the deployed EJB. Use the name in the enterprise beans bindings, not the java:comp name.

New Name Space Binding

Use this wizard to configure a new name space binding.

Step 1: Specify binding type

→ Step 2: Specify basic properties

Step 3: Summary

Specify basic properties

Binding type
EJB

* Binding identifier
customer

* Name in name space
CustHome

Enterprise Bean Location

☐ Server cluster

☒ Single server (Node is required)

* Node
h01Node

* Server
server2

* JNDI name
CustomerHome

Previous Next Cancel

Figure 11-11 Defining an EJB name space binding

6. Click **Next**.
7. Click **Finish** and save your changes.

Note: Name space bindings can be configured at the cell, node, and server scope:

- ▶ Bindings configured at the cell scope are included in the local run time name space of all application servers in that cell.
- ▶ Bindings configured at the node scope area included in the local run time name space of all application servers in that node.
- ▶ Bindings configured at the server scope are included in the local run time name space of only that application server.

11.12.2 Foreign cell bindings

To create the binding, do the following:

1. Select **Environment** → **Naming** → **Foreign Cell Bindings**. See Figure 11-12.

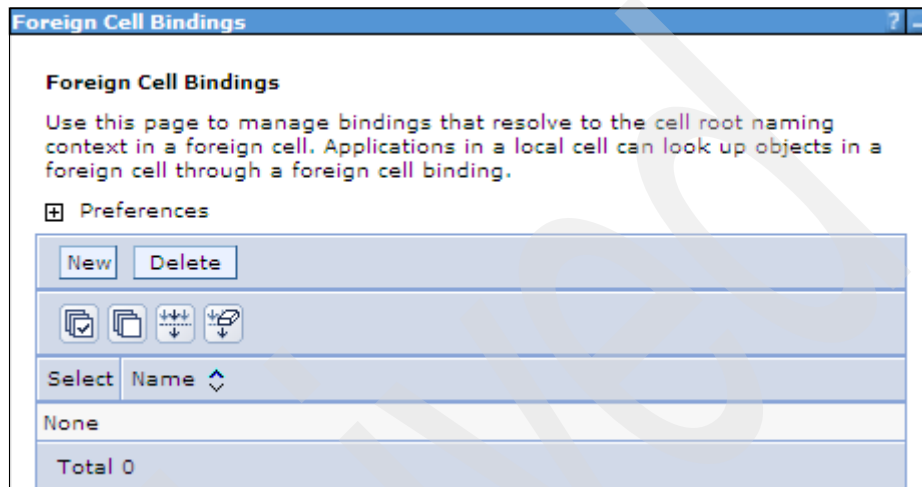


Figure 11-12 Add new foreign cell binding

2. Click **New**. Supply the name of the foreign cell. This must be different to the name of the local cell.
3. Click **Next**. See Figure 11-13.

Wizard to configure new foreign cell bindings

Use this wizard to configure one or more foreign cell bindings. Configure a foreign cell binding for each cell that is accessed by applications in the local cell.

Step 1: Specify foreign cell name

→ **Step 2: Specify bootstrap addresses**

Step 3: Summary

Specify bootstrap addresses

Specify the foreign cell name and create bootstrap endpoint bindings.

Host name

Port

Apply

Edit **Delete**

☒ ☐

Select	Host name	Port
☐	h01Host	2809

Previous **Next** **Cancel**

Figure 11-13 Configuring bootstrap addresses for a foreign cell binding

4. Enter the host name of the deployment manager, node agent, server, or cluster member in the foreign cell that is used for bootstrapping.
5. Enter the bootstrap port number that is used in conjunction with the previously specified host name.
6. Click **Apply**. Your settings will appear in the list at the bottom of the window. You may configure multiple bootstrap addresses for the foreign cell binding.
7. When you are finished, click **OK** and save your changes.

11.12.3 CORBA naming service users and groups

The J2EE role-based authorization concept has been extended to protect the WebSphere CosNaming service. CosNaming security offers increased granularity of security control over CosNaming functions, which affects the content of the WebSphere name space. There are generally two ways in which

client programs will make a CosNaming call. The first is through the JNDI interfaces. The second is CORBA clients invoking CosNaming methods directly.

Note: The authorization policy is only enforced when administrative security is enabled. Before enabling security, you should design your entire security solution. See *WebSphere Application Server V6.1 Security Handbook*, SG24-6316 for information about designing and implementing WebSphere security.

You can design authorization based on users and groups of users defined to the active user registry. Design the authorization by assigning an authority level to one of the following:

- ▶ User
- ▶ Group
- ▶ ALL_AUTHENTICATED (special subject that acts as a group)
This means any user who authenticates by entering a valid user ID and password.
- ▶ EVERYONE (special subject that acts as a group)
All users are authorized. No authentication is necessary.

The roles now have authority level from low to high as follows:

- ▶ Users assigned the CosNamingRead role are allowed to do queries of the WebSphere Name Space, such as through the JNDI lookup method. The special subject “Everyone” is the default policy for this role.
- ▶ Users assigned to the CosNamingWrite role are allowed to do write operations, such as JNDI bind, rebind, or unbind, plus CosNamingRead operations. The special subject All_Authenticated is the default policy for this role.
- ▶ Users assigned to the CosNamingCreate role are allowed to create new objects in the Name Space through such operations as JNDI createSubcontext, plus CosNamingWrite operations. The special subject, All_Authenticated, is the default policy for this role.
- ▶ Users assigned to the CosNamingDelete role are able to destroy objects in the Name Space, for example, using the JNDI destroySubcontext method, as well as CosNamingCreate operations.

By default, you have the following:

- ▶ The ALL_AUTHENTICATED group has the following role privileges: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete.

- The EVERYONE group has CosNamingRead privileges only.

Working with the CORBA naming service authorization is straightforward.

Working with CORBA naming service users

To work with users, do the following:

1. Select **Environment** → **Naming** → **CORBA Naming Service Users**. See Figure 11-14.



Figure 11-14 Add CORBA naming service users

2. Click **Add**.

Enter a case-sensitive user ID and select an authorization level. The user must be a valid user in the active user registry. If you have not activated administrative security, the local operating system user registry will be used.

Note: Before these settings take effect, you will have to enable and configure WebSphere administrative security.

To specify multiple roles, hold the Ctrl key while you click the applicable roles. See Figure 11-15.

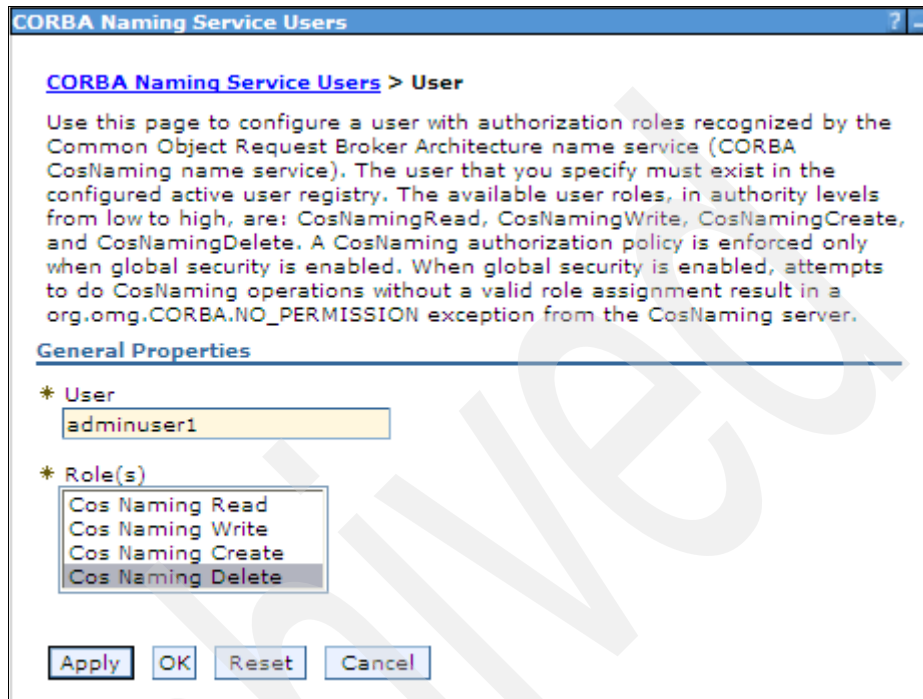


Figure 11-15 Assign an authorization level

3. Click **Apply**.
4. Click **OK** and save your changes.

Working with CORBA naming service groups

To work with groups, do the following:

1. Select **Environment** → **Naming** → **CORBA Naming Service Groups**. Notice that the default settings are defined. Figure 11-16 on page 793 shows the initial settings.

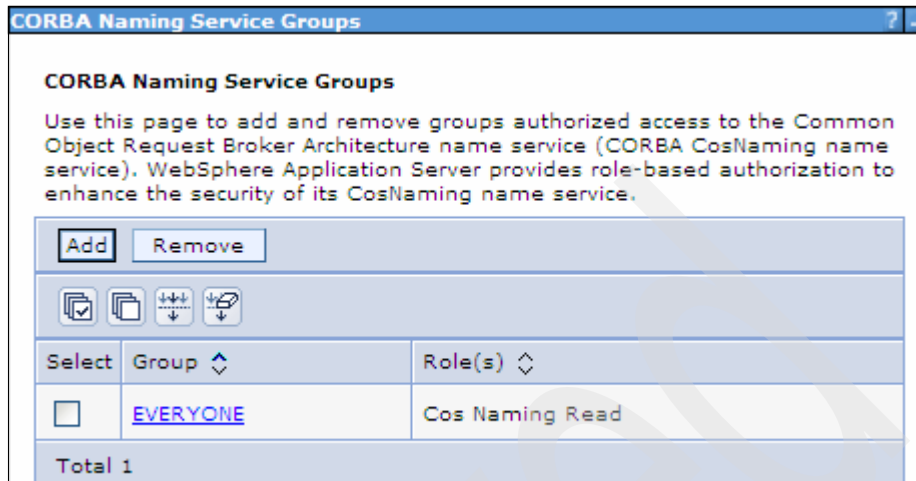


Figure 11-16 Default settings for CORBA naming service groups

Note: The special group EVERYONE is already defined and has roles assigned. To change the roles assigned to the special group, click the group name link.

2. To add a new group, click **Add**. See Figure 11-17 on page 794.

CORBA Naming Service Groups ?

CORBA Naming Service Groups > Group

Use this page to configure a group with authorization roles recognized by the Common Object Request Broker Architecture name service (CORBA CosNaming name service). The available group roles, in authority levels from low to high, are: CosNamingRead, CosNamingWrite, CosNamingCreate, and CosNamingDelete. A CosNaming authorization policy is enforced only when global security is enabled. When global security is enabled, attempts to do CosNaming operations without a valid role assignment result in a org.omg.CORBA.NO_PERMISSION exception from the CosNaming server.

General Properties

Group

☒ Enter group name

Group name
Administrators

☐ Select from special subjects

Special subjects
EVERYONE

* Role(s)

Cos Naming Read
Cos Naming Write
Cos Naming Create
Cos Naming Delete

Apply OK Reset Cancel

Figure 11-17 Assign an authorization level

3. Select the **Specify Group** button, enter a case-sensitive group name and select an authorization level, or role. The group must be a valid user in the active user registry. If you have not activated administrative security, the local operating system user registry will be used. Remember, before these settings take effect, you will have to enable and configure WebSphere administrative security.

To specify multiple roles, hold the Ctrl key while you click the applicable roles.

4. Click **Apply**.
5. Click **OK** and save your changes.

Understanding class loaders

Understanding how the Java and WebSphere class loaders work is critical to packaging and deploying J2EE applications. Failure to set up the class loaders properly most likely results in a cascade of the infamous class loading exceptions (such as `ClassNotFoundException`) when trying to start your application.

This chapter explains class loaders and how to customize the behavior of the WebSphere class loaders to suit your particular application's requirements. The chapter concludes with an example designed to illustrate these concepts.

This chapter includes the following topics:

- ▶ A brief introduction to Java class loaders
- ▶ WebSphere class loaders overview
- ▶ Configuring WebSphere for class loaders
- ▶ Class loader viewer
- ▶ Learning class loaders by example
- ▶ Additional class loader diagnostics

12.1 A brief introduction to Java class loaders

Class loaders enable the Java virtual machine (JVM) to load classes. Given the name of a class, the class loader locates the definition of this class. Each Java class must be loaded by a class loader.

When you start a JVM, you use three class loaders: the bootstrap class loader, the extensions class loader, and the application class loader.

- ▶ The *bootstrap* class loader is responsible for loading only the core Java libraries, that is `vm.jar`, `core.jar`, and so on, in the `<JAVA_HOME>/jre/lib` directory. This class loader, which is part of the core JVM, is written in native code.
- ▶ The *extensions* class loader is responsible for loading the code in the extensions directories (`<JAVA_HOME>/jre/lib/ext` or any other directory specified by the `java.ext.dirs` system property). This class loader is implemented by the `sun.misc.Launcher$ExtClassLoader` class.
- ▶ The *application class loader* is responsible for loading the code that is found on `java.class.path`, which ultimately maps to the system `CLASSPATH` variable. This class loader is implemented by the `sun.misc.Launcher$AppClassLoader` class.

The parent-delegation model is a key concept to understand when dealing with class loaders. It states that a class loader delegates class loading to its parent before trying to load the class itself. The parent class loader can be either another custom class loader or the bootstrap class loader. But what is very important is that a class loader can only delegate requests to its parent class loader, never to its child class loaders (it can go up the hierarchy but never down).

The extensions class loader is the parent for the application class loader. The bootstrap class loader is the parent for the extensions class loader. The class loaders hierarchy is shown in Figure 12-1 on page 797.

If the application class loader needs to load a class, it first delegates to the extensions class loader, which, in turn, delegates to the bootstrap class loader. If the parent class loader cannot load the class, the child class loader tries to find the class in its own repository. In this manner, a class loader is only responsible for loading classes that its ancestors cannot load.

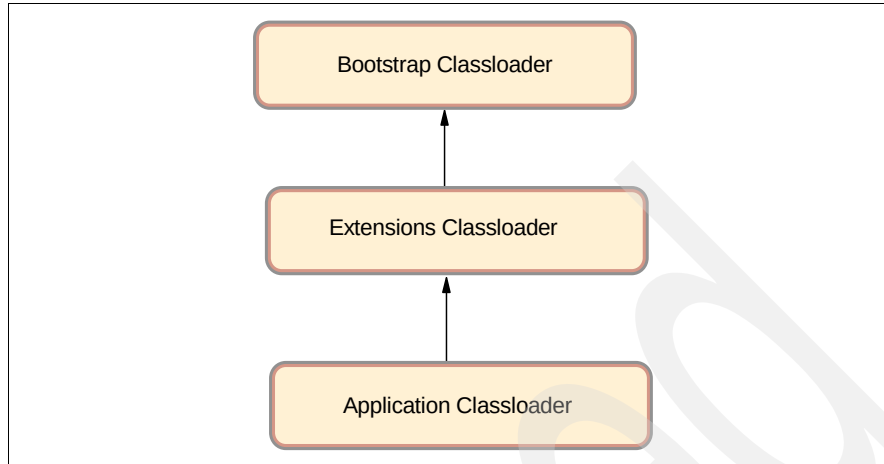


Figure 12-1 Java class loaders hierarchy

This behavior can lead to some interesting problems if a class is loaded from a class loader that is not on a leaf node in the class loader tree. Consider Example 12-1. A class called `WhichClassLoader1` loads a class called `WhichClassLoader2`, in turn invoking a class called `WhichClassLoader3`.

Example 12-1 `WhichClassLoader1` and `WhichClassLoader2` source code

```
public class WhichClassLoader1 {

    public static void main(String[] args) throws javax.naming.NamingException
    {
        // Get classpath values
        String bootClassPath = System.getProperty("sun.boot.class.path");
        String extClassPath = System.getProperty("java.ext.dirs");
        String appClassPath = System.getProperty("java.class.path");

        // Print them out
        System.out.println("Bootstrap classpath =" + bootClassPath + "\n");
        System.out.println("Extensions classpath =" + extClassPath + "\n");
        System.out.println("Application classpath=" + appClassPath + "\n");

        // Load classes
        Object obj = new Object();
        WhichClassLoader1 wc11 = new WhichClassLoader1();
        WhichClassLoader2 wc12 = new WhichClassLoader2();

        // Who loaded what?
        System.out.println("Object was loaded by "
            + obj.getClass().getClassLoader());
        System.out.println("WCL1 was loaded by "
```

```

        + wcl1.getClass().getClassLoader());
System.out.println("WCL2 was loaded by "
        + wcl2.getClass().getClassLoader());

    wcl2.getTheClass();
}
}
=====
public class WhichClassLoader2 {

    // This method is invoked from WhichClassLoader1
    public void getTheClass() {
        WhichClassLoader3 wcl3 = new WhichClassLoader3();
        System.out.println("WCL3 was loaded by "
            + wcl3.getClass().getClassLoader());
    }
}

```

If all WhichClassLoaderX classes are put on the application class path, the three classes are loaded by the application class loader, and this sample runs just fine.

Now suppose you package the WhichClassLoader2.class file in a JAR file that you store under <JAVA_HOME>/jre/lib/ext directory. You see the output in Example 12-2.

Example 12-2 NoClassDefFoundError exception trace

Bootstrap classpath

```

=C:\WebSphere\AppServer\java\jre\lib\vm.jar;C:\WebSphere\AppServer\java\jre\lib\
\core.jar;C:\WebSphere\AppServer\java\jre\lib\charsets.jar;C:\WebSphere\AppServ
er\java\jre\lib\graphics.jar;C:\WebSphere\AppServer\java\jre\lib\security.jar;C
:\WebSphere\AppServer\java\jre\lib\ibmpkcs.jar;C:\WebSphere\AppServer\java\jre\
lib\ibmorb.jar;C:\WebSphere\AppServer\java\jre\lib\ibmcfw.jar;C:\WebSphere\AppS
erver\java\jre\lib\ibmorbapi.jar;C:\WebSphere\AppServer\java\jre\lib\ibmjcefw.j
ar;C:\WebSphere\AppServer\java\jre\lib\ibmjgssprovider.jar;C:\WebSphere\AppServ
er\java\jre\lib\ibmjsseprovider2.jar;C:\WebSphere\AppServer\java\jre\lib\ibmjaa
slm.jar;C:\WebSphere\AppServer\java\jre\lib\ibmjaasactivelm.jar;C:\WebSphere\Ap
pServer\java\jre\lib\ibmcertpathprovider.jar;C:\WebSphere\AppServer\java\jre\li
b\server.jar;C:\WebSphere\AppServer\java\jre\lib\xml.jar

```

Extensions classpath =C:\WebSphere\AppServer\java\jre\lib\ext
Application classpath=.

```

Exception in thread "main" java.lang.NoClassDefFoundError: WhichClassLoader3
    at java.lang.J9VMInternals.verifyImpl(Native Method)
    at java.lang.J9VMInternals.verify(J9VMInternals.java:59)
    at java.lang.J9VMInternals.initialize(J9VMInternals.java:120)
    at WhichClassLoader1.main(WhichClassLoader1.java:17)

```

As you can see, the program fails with a `NoClassDefFoundError` exception, which might sound strange because `WhichClassLoader3` is on the application class path. The problem is that it is *now* on the wrong class path.

What happened was that the `WhichClassLoader2` class was loaded by the extensions class loader. In fact, the application class loader delegated the load of the `WhichClassLoader2` class to the extensions class loader, which in turn delegated the request to the bootstrap class loader. Because the bootstrap class loader could not find the class, the class loading control was returned to the extensions class loader. The extensions class loader found the class on its class path and loaded it.

Now, when a class has been loaded by a class loader, any new classes that the class needs reuse the same class loader to load them (or goes up the hierarchy according to the parent-delegation model). So when the `WhichClassLoader2` class needed to access the `WhichClassLoader3` class, it is the extensions class loader that first gets the request to load it. The extensions class loader first delegates the request to the Bootstrap class path, which cannot find the class, and then tries to load it itself but does not find it either because `WhichClassLoader3` is not on the extensions class path but on the application classpath. And because the extensions class loader cannot delegate the request to the application class loader (a delegate request can only go up the hierarchy, never down), a `NoClassDefFoundError` exception is thrown.

Note: Remember that developers very often also load property files through the class loader mechanism using the following syntax:

```
Properties p = new Properties();  
p.load(MyClass.class.getClassLoader().getResourceAsStream("myApp.properties"  
));
```

This means, if the class `MyClass` is loaded by the extensions class loader and the `myApp.properties` file is only seen by the application class loader, the loading of the property file fails.

12.2 WebSphere class loaders overview

Note: Keep in mind when reading the following discussion that each JVM has its own setup of class loaders. In a WebSphere environment hosting multiple application servers (JVMs), this means the class loaders for the JVMs are completely separated even if they are running on the same physical machine.

Also note that the Java Virtual Machine (JVM) uses class loaders called the extensions and application class loaders. As you will see, the WebSphere run time also uses class loaders called extensions and application class loader, but despite their names, they are not the same as the JVM ones.

WebSphere provides several custom delegated class loaders, as shown in Figure 12-2 on page 800.

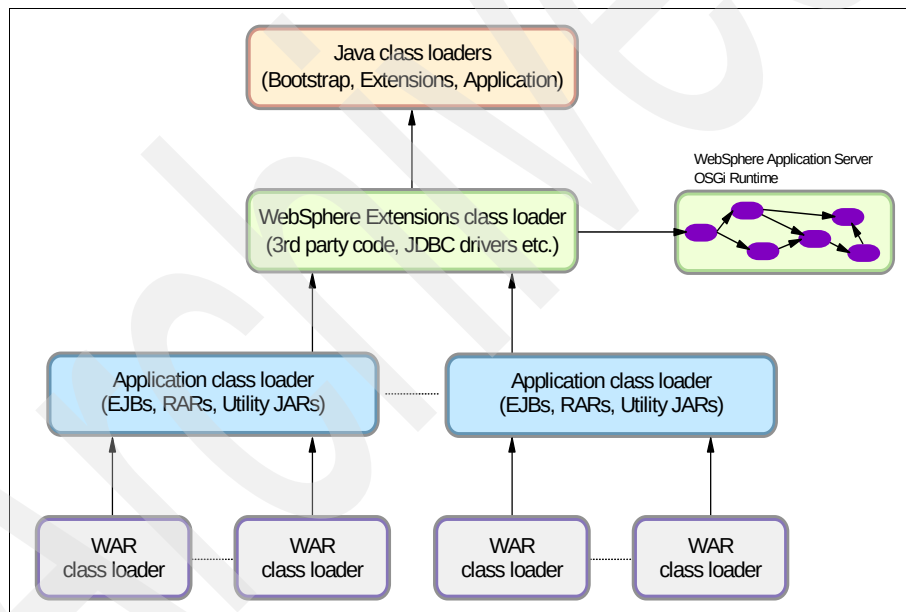


Figure 12-2 WebSphere class loaders hierarchy

The top box represents the Java (bootstrap, extensions, and application) class loaders. WebSphere loads just enough here to get itself bootstrapped and initialize the WebSphere extensions class loader.

12.2.1 WebSphere extensions class loader

New in V6.1: The WebSphere extensions class loader is where WebSphere itself is loaded. In previous versions of WebSphere, the run time was loaded by this single class loader. However, beginning with WebSphere Application Server V6.1, WebSphere is now packaged as a set of OSGi bundles. Each OSGi bundle is loaded separately by its own class loader. This network of OSGi class loaders is then connected to the extensions class loader and the rest of the class loader hierarchy through an OSGi gateway class loader.

Despite this architectural change in the internals of how WebSphere loads its own classes, there is no behavioral change as far as your applications are concerned. They still have the same visibility, and the same class loading options still exist for your application.

New in V6.1: In previous versions of WebSphere Application Server the WebSphere run time classes files were stored in the `classes`, `lib`, `lib\ext`, and `installedChannels` directories in the `<was_home>` directory. Because of the OSGi packaging, these directories no longer exist and the run time classes are now stored in the `<was_home>\plugins` directory.

The class path used by the extensions class loader is retrieved from the `ws.ext.dirs` system property, which is initially derived from the `WAS_EXT_DIRS` environment variable set in the `setupCmdLine` script file. The default value of `ws.ext.dirs` is displayed in Example 12-3.

Example 12-3 Default value of `ws.ext.dirs`

```
SET
WAS_EXT_DIRS=%JAVA_HOME%\lib;%WAS_HOME%\classes;%WAS_HOME%\lib;%WAS_HOME%\installedChannels;%WAS_HOME%\lib\ext;%WAS_HOME%\web\help;%ITP_LOC%\plugins\com.ibm.e
tools.ejbdeploy\runtime
```

Each directory listed in the `ws.ext.dirs` environment variable is added to the WebSphere extensions class loaders class path and every JAR file and ZIP file in the directory is added to the class path.

As you can see, even though the `classes`, `lib`, `lib\ext`, and `installedChannels` directories no longer exist in the `<was_home>` directory, the `setupCmdLine` script still adds them to the extensions class path. This means that if you previously have added your own JAR files to, for example, the `<was_home>\lib` directory, you could create this directory and add your JAR files to it and they would still be loaded by the extensions class loader. However, this is not recommended and you should really try to migrate away from such a setup.

On the other hand, if you have developed Java applications that rely on the WebSphere JAR files that were previously in the `<was_home>\lib` directory, you will need to modify your application to retain compatibility. WebSphere Application Server V6.1 provides two thin client libraries designed specifically for such applications: one administrative client library and one Web services client library. These thin client libraries can be found in the `<was_home>\runtimes` directory:

- ▶ `com.ibm.ws.admin.client_6.1.0.jar`
- ▶ `com.ibm.ws.webservices.thinclient_6.1.0.jar`

These libraries provide everything your application might need for connecting to and working with WebSphere.

New in V6.1: WebSphere Application Server V6.1 gives you the ability to restrict access to internal WebSphere classes so that your applications do not make unsupported calls to WebSphere classes not published in the official WebSphere Application Server API. This setting is a per-server (JVM) setting called Access to internal server classes.

The default setting is Allow, meaning that your applications can make unrestricted calls to non-public internal WebSphere classes. This is not recommended and may be prohibited in future releases. Therefore, as an administrator, it is a good idea to switch this setting to Restrict to see if your applications still work. If they depend on non-public WebSphere internal classes you will receive a `ClassNotFoundException`, and in that case you can switch back to Allow. Your developers should then try to migrate their applications so that they do not make unsupported calls to the WebSphere internal classes in order to retain compatibility with future WebSphere Application Server releases.

12.2.2 Application and Web module class loaders

J2EE applications consist of five primary elements: Web modules, EJB modules, application client modules, resource adapters (RAR files), and utility JARs. Utility JARs contain code used by both EJBs and servlets. Utility frameworks such as `log4j` are good examples of a utility JAR.

EJB modules, utility JARs, resource adapter files, and shared libraries associated with an application are always grouped together into the same class loader. This class loader is called the application class loader. Depending on the class loader policy, this class loader can be shared by multiple applications (EARs), or be unique for each application, which is the default.

By default, Web modules receive their own class loader, a WAR class loader, to load the contents of the WEB-INF/classes and WEB-INF/lib directories. You can modify the default behavior by changing the application's WAR class loader policy. The default is to Class loader for each WAR file in the application (this setting was called Module in previous releases). If the WAR class loader policy is set to Single class loader for application (called Application in previous releases), the Web module contents are loaded by the application class loader in addition to the EJBs, RARs, utility JARs, and shared libraries. The application class loader is the parent of the WAR class loader.

The application and the WAR class loaders are reloadable class loaders. They monitor changes in the application code to automatically reload modified classes. You can modify this behavior at deployment time.

12.2.3 Handling JNI code

Because a JVM only has a single address space and native code can only be loaded once per address space, the JVM specification states that native code may only be loaded by one class loader in a JVM.

This may cause a problem if, for example, you have an application (EAR file) with two Web modules that both need to load the same native code through a Java Native Interface (JNI). Only the Web module that first loads the library will succeed.

To solve this problem, you can break out just the few lines of Java code that load the native code into a class on its own and place this file on WebSphere's application class loader (in a utility JAR). However, if you would deploy multiple such applications (EAR files) to the same application server (JVM), you would have to place the class file on the WebSphere extensions class loader instead to ensure the native code is only loaded once per JVM.

If the native code is placed on a reloadable class loader (such as the application class loader or the WAR class loader), it is important that the native code can properly unload itself should the Java code have to reload. WebSphere has no control over the native code and if it does not unload and load properly the application may fail.

If one native library depends on another one, things become even more complicated. Search for Dependent native library in the Information Center for more details.

12.3 Configuring WebSphere for class loaders

In the previous topic, you learned about WebSphere class loaders and how they work together to load classes. There are settings in WebSphere Application Server that allow you to influence WebSphere class loader behavior. This section discusses these options.

12.3.1 Class loader policies

For each application server in the system, the class loader policy can be set to Single or Multiple.

When the application server class loader policy is set to Single, a single application class loader is used to load all EJBs, utility JARs, and shared libraries within the application server (JVM). If the WAR class loader policy then has been set to Single class loader for application (or Application), the Web module contents for this particular application are also loaded by this single class loader.

When the application server class loader policy is set to Multiple, the default, each application will receive its own class loader for loading EJBs, utility JARs, and shared libraries. Depending on whether the WAR class loader loading policy is set to Class loader for each WAR file in application (or Module) or Single class loader for application (or Application), the Web module might or might not receive its own class loader.

Here is an example to illustrate. You have two applications, Application1 and Application2, running in the same application server. Each application has one EJB module, one utility JAR, and two Web modules. If the application server has its class loader policy set to Multiple, the default, and the class loader policy for all the Web modules are set to use a class loader for each WAR file in application (Module), also the default, the result is as shown in Figure 12-3.

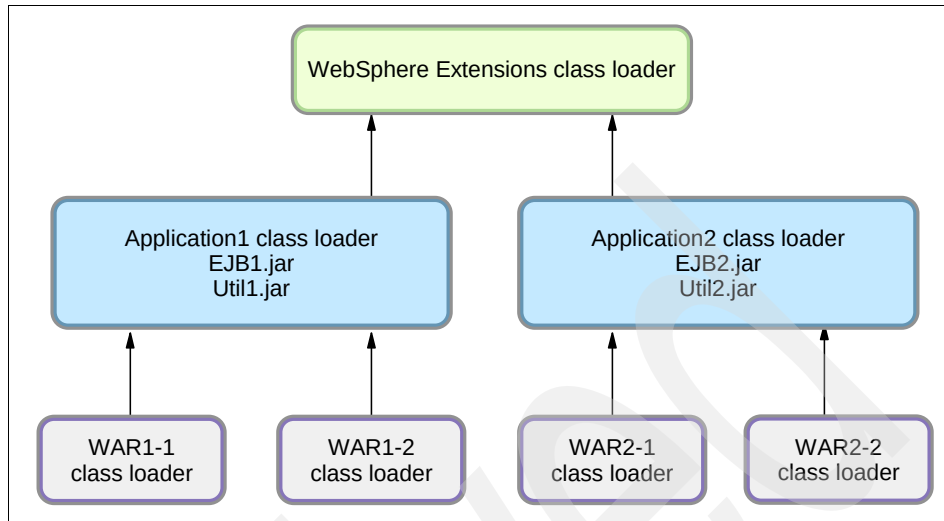


Figure 12-3 Class loader policies: Example 1

Each application is completely separated from the other and each Web module is also completely separated from the other one in the same application. WebSphere's default class loader policies results in total isolation between the applications and the modules.

If we now change the class loader policy for the WAR2-2 module from Class loader for each WAR file in application (Module) to Single class loader for application (Application), the result is shown in Figure 12-4 on page 806.

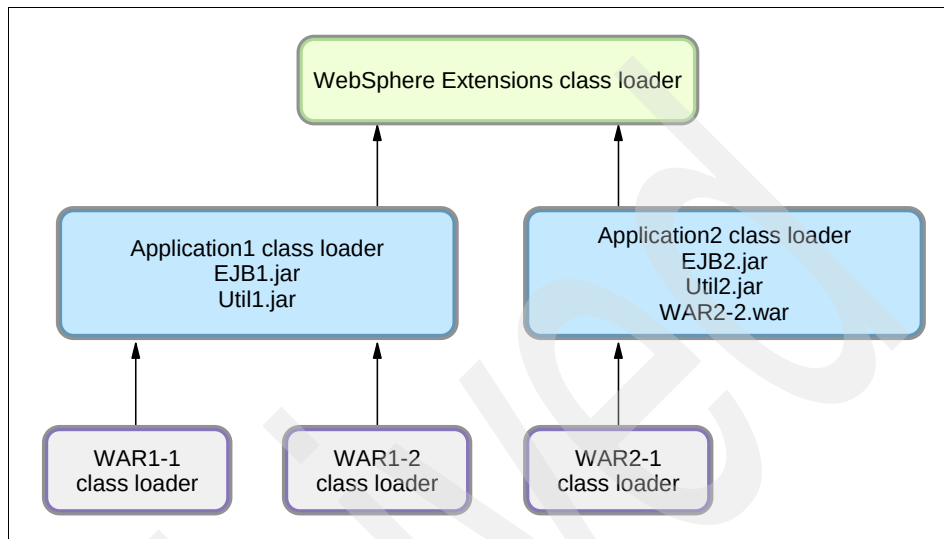


Figure 12-4 Class loader policies: Example 2

Web module WAR2-2 is loaded by Application2's class loader and classes, for example, in Util2.jar, are able to see classes in WAR2-2's /WEB-INF/classes and /WEB-INF/lib directories.

As a last example, if we change the class loader policy for the application server from Multiple to Single and also change the class loader policy for WAR2-1 from Class loader for each WAR file in application (Module) to Single class loader for application (Application), the result is as shown in Figure 12-5 on page 807.

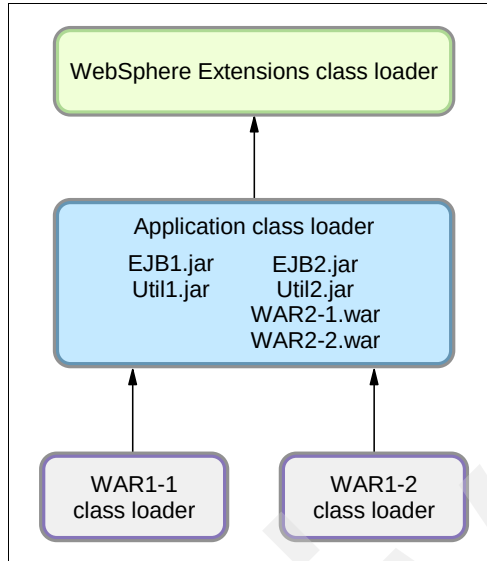


Figure 12-5 Class loader policies: Example 3

There is now only a single application class loader loading classes for both Application1 and Application2. Classes in Util1.jar can see classes in EJB2.jar, Util2.jar, WAR2-1.war and WAR2-2.war. The classes loaded by the application class loader still cannot, however, see the classes in the WAR1-1 and WAR1-2 modules, because a class loader can only find classes by going up the hierarchy, never down.

12.3.2 Class loading/delegation mode

WebSphere's application class loader and WAR class loader both have a setting called the class loader order. This setting determines whether they should follow the normal Java class loader delegation mechanism, as described in 12.1, "A brief introduction to Java class loaders" on page 796, or override it.

There are two possible options for the class loading mode:

- ▶ Classes loaded with parent class loader first
- ▶ Classes loaded with application class loader first

In previous WebSphere releases, these settings were called PARENT_FIRST and PARENT_LAST, respectively.

The default value for class loading mode is Classes loaded with parent class loader first (PARENT_FIRST). This mode causes the class loader to first delegate the loading of classes to its parent class loader before attempting to

load the class from its local class path. This is the default policy for standard Java class loaders.

If the class loading policy is set to Classes loaded with application class loader first (PARENT_LAST), the class loader attempts to load classes from its local class path before delegating the class loading to its parent. This policy allows an application class loader to override and provide its own version of a class that exists in the parent class loader.

Note: The administrative console is a bit confusing at this point. On the settings page for a Web module, the two options for class loader order are Classes loaded with parent class loader first and Classes loaded with application class loader first. However, in this context, the “application class loader” really refers to the WAR class loader, so the option Classes loaded with application class loader first should really be called Classes loaded with WAR class loader first.

Assume you have an application, similar to Application1 in the previous examples, and it uses the popular log4j package to perform logging from both the EJB module and the two Web modules. Also assume that each module has its own, unique, log4j.properties file packaged into the module. It is tempting to configure log4j as a utility JAR so you only have a single copy of it in your EAR file.

However, if you do that, you might be surprised to see that all modules, including the Web modules, load the log4j.properties file from the EJB module. The reason is that when a Web module initializes the log4j package, the log4j classes are loaded by the application class loader. Log4j is configured as a utility JAR. Log4j then looks for a log4j.properties file on its class path and finds it in the EJB module.

Even if you do not use log4j for logging from the EJB module and the EJB module does not, therefore, contain a log4j.properties file, log4j does not find the log4j.properties file in any of the Web modules anyway. The reason is that a class loader can only find classes by going up the hierarchy, never down.

To solve this problem, you can either:

- Create a separate file, for example, Resource.jar, configure it as a utility JAR, move all log4j.properties files from the modules into this file, and make their names unique (like war1-1_log4j.properties, war1-2_log4j.properties, and ejb1_log4j.properties). When initializing log4j from each module, tell it to load the proper configuration file for the module instead of the default (log4j.properties).

- Keep the log4j.properties for the Web modules in their original place (/WEB-INF/classes), add log4j.jar to both Web modules (/WEB-INF/lib) and set the class loading mode for the Web modules to Classes loaded with application class loader first (PARENT_LAST). When initializing log4j from a Web module, it loads the log4j.jar from the module itself and log4j would find the log4j.properties on its local classpath, the Web module itself. When the EJB module initializes log4j, it loads from the application class loader and it finds the log4j.properties file on the same class path, the one in the EJB1.jar file.
- Merge, if possible, all log4j.properties files into one and place it on the Application class loader, in a Resource.jar file, for example).

Singletons: The Singleton pattern is used to ensure that a class is instantiated only once. However, *once* only means *once for each class loader*. If you have a Singleton being instantiated in two separated Web modules, two separate instances of this class will be created, one for each WAR class loader. So in a multi-class loader environment, special care must be taken when implementing Singletons.

12.3.3 Shared libraries

Shared libraries are files used by multiple applications. Examples of shared libraries are commonly used frameworks like Apache Struts or log4j. You use shared libraries typically to point to a set of JARs and associate those JARs to an application, a Web module, or the class loader of an application server. Shared libraries are especially useful when you have different versions of the same framework you want to associate to different applications.

Shared libraries are defined using the administration tools. They consist of a symbolic name, a Java class path, and a native path for loading JNI libraries. They can be defined at the cell, node, server, or cluster level. However, simply defining a library does not cause the library to be loaded. You must associate the library to an application, a Web module, or the class loader of an application server for the classes represented by the shared library to be loaded. Associating the library to the class loader of an application server makes the library available to all applications on the server.

Note: If you associate a shared library to an application, do not associate the same library to the class loader of an application server.

You can associate the shared library to an application in one of two ways:

- ▶ You can use the administration tools. The library is added using the Shared libraries references link under the References section for the enterprise application.
- ▶ You can use the manifest file of the application and the shared library. The shared library contains a manifest file that identifies it as an extension. The dependency to the library is declared in the application's manifest file by listing the library extension name in an extension list.

For more information about this method, search for installed optional packages in the Information Center.

Shared files are associated with the class loader of an application server using the administrative tools. The settings are found in the **Server Infrastructure** section. Expand the **Java and Process Management**. Select **Class loader** and then click the **New** button to define a new class loader. Once you have defined a new class loader, you can modify it and, using the Shared library references link, you can associate it to the shared libraries you need.

See “Step 4: Sharing utility JARs using shared libraries” on page 820 for more details.

12.4 Class loader viewer

New in V6.0.2: WebSphere Application Server V6.0.2 introduced a new utility called the Class Loader Viewer. When activated, this utility can help you diagnose class loading problems by showing you the different class loaders involved, their settings, and the classes loaded by each of them.

If the Class Loader Viewer Service is not enabled, the Class Loader Viewer only displays the hierarchy of class loaders and their classpaths, but not the classes actually loaded by each of the class loaders. This also means that the search capability of the Class Loader Viewer is lost.

To enable the Class Loader Viewer Service, select **Servers** → **Application Servers** → **<server name>** and then click the **Class Loader Viewer Service** under the **Additional Properties** link. Then select **Enable service at server startup**. You will need to restart the application server for the setting to take effect.

In the next section, we give an example of how to work with the different class loader settings and will then use the Class Loader Viewer also to illustrate the different results.

12.5 Learning class loaders by example

We have now described all the different options for influencing class loader behavior. In this section, we take an example and use all the different options we have discussed to this point so that you can better evaluate the best solution for your applications.

We have created a very simple application, with one servlet and one EJB. Both call a class, `VersionChecker`, shown in Example 12-4. This class can print which class loader was used to load the class. The `VersionChecker` class also has an internal value that can be printed to check which version of the class we are using. This will be used later to demonstrate the use of multiple versions of the same utility JAR.

Example 12-4 VersionChecker class source code

```
package com.itso.classloaders;

public class VersionChecker {
    static final public String classVersion = "v1.0";

    public String getInfo() {
        return ("VersionChecker is " + classVersion +
            ". Loaded by " + this.getClass().getClassLoader());
    }
}
```

Once installed, the application can be invoked through `http://localhost:9080/ClassLoaderExampleWeb/ExampleServlet`. This invokes the `ExampleServlet` which calls `VersionChecker` and then displays the sample information in Example 12-5.

Example 12-5 Invoking ExampleServlet

`VersionChecker` is v1.0.

Loaded by `com.ibm.ws.classloader.CompoundClassLoader@71827182`

Local ClassPath:

`C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcggl8Node02Cell1\ClassLoaderExample.ear\ClassLoaderExampleWeb.war\WEB-INF\classes;C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcggl8Node02Cell1\ClassLoaderExample.ear\ClassLoaderExampleWeb.war\WEB-INF\lib\VersionCheckerV1.jar;C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcggl8Node02Cell1\ClassLoaderExample.ear\ClassLoaderExampleWeb.war`

Delegation Mode: `PARENT_FIRST`

The `VersionCheckerV1.jar` file contains the `VersionChecker` class file that returns Version number 1.0. For all the following tests, we have, unless otherwise noted, left the class loader policies and loading modes to their defaults. In other words, we have one class loader for the application and one for the WAR file. Both have their delegation modes set to Classes loaded with parent class loader first (`PARENT_FIRST`). We assume the application has been deployed to an application server called `AppSrv02`.

12.5.1 Step 1: Simple Web module packaging

Start with the following assumption: our utility class is only used by a servlet. We have placed the `VersionCheckerV1.jar` file under the `WEB-INF/lib` directory of the Web module.

Tip: You place JAR files used by a single Web module, or a JAR file that *only* this Web module should see under `WEB-INF/lib`.

When we run the application in such a configuration, we obtain the results shown in Example 12-6.

VersionChecker called from Servlet

VersionChecker is v1.0.

Loaded by com.ibm.ws.classloader.CompoundClassLoader@71827182

Local ClassPath:

C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcgg1d8Node02Cell1\ClassLoaderExample.ear\ClassLoaderExampleWeb.war\WEB-INF\classes;C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcgg1d8Node02Cell1\ClassLoaderExample.ear\ClassLoaderExampleWeb.war\WEB-INF\lib\VersionCheckerV1.jar;C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcgg1d8Node02Cell1\ClassLoaderExample.ear\ClassLoaderExampleWeb.war

Delegation Mode: PARENT_FIRST

There are a few things we can learn from this trace:

1. The type of the WAR class loader is
com.ibm.ws.classloader.CompoundClassLoader.
2. It searches classes in the following order:

ClassLoaderExampleWeb.war\WEB-INF\classes
ClassLoaderExampleWeb.war\WEB-INF\lib\VersionCheckerV1.jar
ClassLoaderExampleWeb.war

The WEB-INF/classes folder holds unpacked resources (such as servlet classes, plain Java classes, and property files), while the WEB-INF/lib holds resources packaged as JAR files. You can choose to package your Java code in JAR files and place them in the lib directory or you can put them unpacked in the classes directory. They will both be on the same classpath. Because our sample application was developed and exported from the Application Server Toolkit, our servlet goes into the classes folder, because the toolkit does not package the Java classes in a JAR file when exporting an application.

The root of the WAR file is the next place where you can put code or properties, but you really should not do that because that folder is the document root for the Web server (if the File Serving Servlet capabilities are enabled, which they are by default) so anything that is in that folder is accessible from a browser. According to the J2EE specification, though, the WEB-INF folder is protected, which is why the classes and lib folders are under WEB-INF.

The class loader class path is dynamically built at application startup.

We can now also use the Class Loader Viewer to display the class loader. In the administrative console, select **Troubleshooting** → **Class Loader Viewer**. Then expand **server1** → **Applications** → **ClassloaderExample** → **Web modules** and click the **ClassloaderExampleWeb.war**, as shown in Figure 12-6.

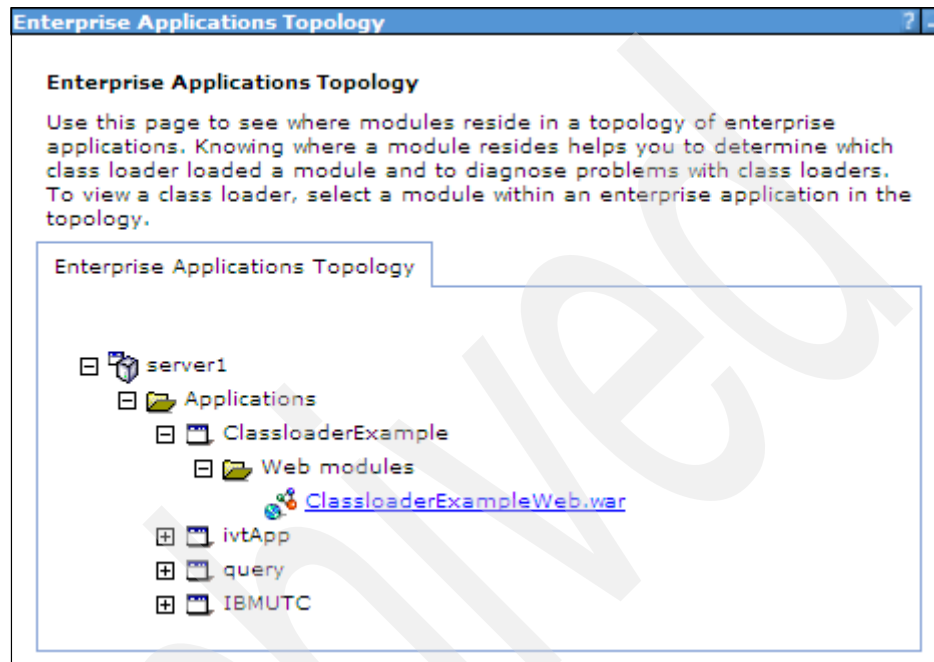


Figure 12-6 Class Loader Viewer showing applications tree

When the Web module is expanded, the Class Loader Viewer shows the hierarchy of class loaders all the way from the JDK Extensions and JDK application class loaders at the top to the WAR class loader at the bottom, called the compound class loader. See Figure 12-7.

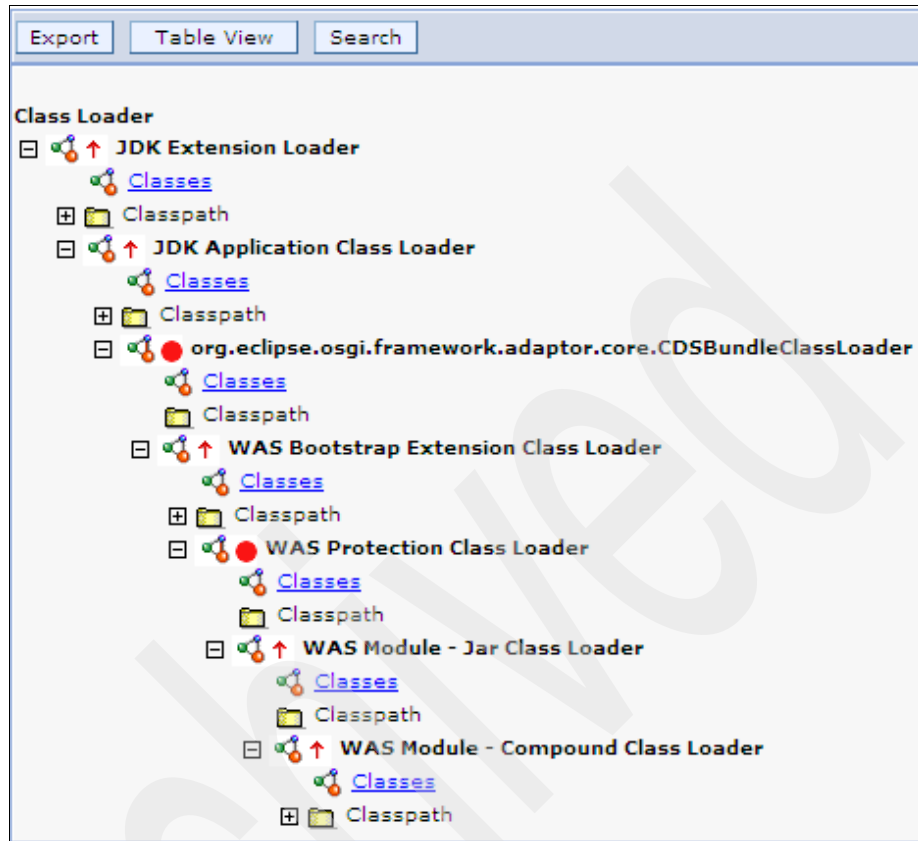


Figure 12-7 Class Loader Viewer showing class loader hierarchy

If you expand the classpath for the WAS Module - Compound Class Loader, you see the same information as our VersionChecker class prints. See Example 12-7.

Example 12-7 Class Loader Viewer showing class path for WAR class loader

```

file:/C:/WebSphere/AppServer/profiles/AppSrv02/installedApps/kcgg1d8Nod
e02Cell/ClassloaderExample.ear/ClassloaderExampleWeb.war/WEB-INF/classe
s
file:/C:/WebSphere/AppServer/profiles/AppSrv02/installedApps/kcgg1d8Nod
e02Cell/ClassloaderExample.ear/ClassloaderExampleWeb.war/WEB-INF/lib/Ve
rsionCheckerV1.jar
file:/C:/WebSphere/AppServer/profiles/AppSrv02/installedApps/kcgg1d8Nod
e02Cell/ClassloaderExample.ear/ClassloaderExampleWeb.war

```

Note: For the Class Loader Viewer to display the classes loaded, the Class Loader Viewer Service must be enabled as described in “Class loader viewer” on page 810.

The Class Loader Viewer also has a table view that displays all the class loaders and the classes loaded by each of them on a single page. The table view also displays the Delegation mode. True means that classes are loaded with parent class loader first (PARENT_FIRST), while false means that classes are loaded with application class loader first (PARENT_LAST), or the WAR class loader in the case of a Web module. See Figure 12-8.

WAS Module - Compound Class Loader	
Delegation	true
Classpath	file:/C:/WebSphere/AppServer/profiles/AppSrv02/installedApps/k
	INF/classes
	file:/C:/WebSphere/AppServer/profiles/AppSrv02/installedApps/k
Classes	file:/C:/WebSphere/AppServer/profiles/AppSrv02/installedApps/k
	com.itso.classloaders.ExampleServlet
	com.itso.classloaders.VersionChecker
	com.sun.faces.config.ConfigureListener
	java.io.PrintWriter
	java.lang.Class
	java.lang.Object
	java.lang.StringBuffer
	java.lang.StringBuilder
	javax.servlet.Servlet
	javax.servlet.http.HttpServlet
	javax.servlet.http.HttpServletResponse

Figure 12-8 Class Loader Viewer table view

As you can see, the WAR class loader has loaded our example servlet and the VersionChecker class, just as expected.

The Class Loader Viewer also has a search feature where you can search for classes, JAR files, folders, and so on. This can be particularly useful if you do not know which of the class loaders loaded a class you are interested in. The search feature is case sensitive but allows wild cards, so a search for `*VersionChecker*` finds our VersionChecker class.

12.5.2 Step 2: Adding an EJB module and Utility jar

Next, we decided to add an EJB to our application, which also depends on our VersionChecker JAR file. For this task, we added a VersionCheckerV2.jar file to the root of our EAR. The VersionChecker class in this JAR file returns Version 2.0. To make it available as a utility JAR on the extensions class loader, we added a reference to it in the EJB module's manifest file, as shown in Example 12-8.

Example 12-8 Updated MANIFEST.MF for EJB module

```
Manifest-Version: 1.0
Class-Path: VersionCheckerV2.jar
```

The result is that we now have a Web module with a servlet in the WEB-INF/classes folder and the VersionCheckerV1.jar file in the WEB-INF/lib folder. We also have an EJB module that references the VersionCheckerV2.jar Utility JAR in the root of the EAR. Which version of the VersionChecker class file would you expect the Web module to load? Version 1.0 from the WEB-INF/lib or version 2.0 from the Utility JAR?

The test results are then as shown in Example 12-9.

Example 12-9 Class loader: Example 2

VersionChecker called from Servlet
VersionChecker is **v2.0**.

Loaded by com.ibm.ws.classloader.**CompoundClassLoader@26282628**

Local ClassPath:
C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcg1d8Node02Ce11\ClassLoaderExample.ear\ClassLoaderExampleEJB.jar;C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcg1d8Node02Ce11\ClassLoaderExample.ear\VersionCheckerV2.jar

Delegation Mode: PARENT_FIRST

VersionChecker called from EJB
VersionChecker is **v2.0**.

Loaded by com.ibm.ws.classloader.**CompoundClassLoader@26282628**

Local ClassPath:
C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcg1d8Node02Ce1

```
1\ClassLoaderExample.ear\ClassLoaderExampleEJB.jar;C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcg1d8Node02Cell1\ClassLoaderExample.ear\VersionCheckerV2.jar
```

Delegation Mode: PARENT_FIRST

As you can see, the VersionChecker is Version 2.0 when called both from the EJB module and the Web module. The reason is, of course, that the WAR class loader delegates the request to its parent class loader instead of loading it itself, so the Utility JAR is loaded by the same class loader regardless of if it was called from the servlet or the EJB.

12.5.3 Step 3: Changing the WAR class loader delegation mode

What if we now wanted the Web module to use the VersionCheckerV1.jar file from the WEB-INF/lib folder? For that task, we would have to change the class loader delegation from parent first to parent last.

Set the delegation mode to PARENT_LAST, using the following steps:

1. Select the **Enterprise Applications** entry in the navigation area.
2. Select the **ClassLoaderExample** application.
3. Select **Manage modules** under the Modules section.
4. Select the **ClassLoaderExampleWeb** module.
5. Change the Class loader order to Classes loaded with application class loader first (PARENT_LAST). Remember, this entry should really be called Classes loaded with WAR class loader first, as noted in “Class loading/delegation mode” on page 807.
6. Click **OK**.
7. Save the configuration.
8. Restart the application.

The VersionCheckerV1 in WEB-INF/lib returns a class version of 1.0. You can see in Example 12-10 on page 819 that this is the version now used by the WAR file.

VersionChecker called from Servlet

VersionChecker is **v1.0**.

Loaded by com.ibm.ws.classloader.**CompoundClassLoader@4d404d40**

Local ClassPath:

C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcggl8Node02Cell1\ClassLoaderExample.ear\ClassLoaderExampleWeb.war\WEB-INF\classes;C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcggl8Node02Cell1\ClassLoaderExample.ear\ClassLoaderExampleWeb.war\WEB-INF\lib\VersionCheckerV1.jar;C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcggl8Node02Cell1\ClassLoaderExample.ear\ClassLoaderExampleWeb.war

Delegation Mode: **PARENT_LAST**

VersionChecker called from EJB

VersionChecker is **v2.0**.

Loaded by com.ibm.ws.classloader.**CompoundClassLoader@37f437f4**

Local ClassPath:

C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcggl8Node02Cell1\ClassLoaderExample.ear\ClassLoaderExampleEJB.jar;C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcggl8Node02Cell1\ClassLoaderExample.ear\VersionCheckerV2.jar

Delegation Mode: **PARENT_FIRST**

Tip: Use this to specify that a Web module should use a specific version of a library, such as Struts, or to override classes coming with the WebSphere runtime. Put the common version at the top of the hierarchy, and the specialized version in WEB-INF/lib.

The J2EE specification does not provide a standard option to specify the delegation mode in the EAR file, but by using a WebSphere Extended EAR file, you can specify this so you do not have to change it every time you redeploy your application.

If you use the search feature of the Class Loader Viewer to search for *VersionChecker*, you would see the two entries in Figure 12-9.

Class Loader	Classes
WAS Module - Compound Class Loader	com.itso.classloaders.VersionChecker
WAS Module - Jar Class Loader	com.itso.classloaders.VersionChecker

Figure 12-9 Class Loader Viewer search feature

The screen is too wide to capture, but the code source is shown in Example 12-11.

Example 12-11 Class Loader Viewer search feature

WAS Module Compound Class Loader (WAR class loader):

file: / C: / WebSphere / AppServer / profiles / AppSrv02 /
installedApps / kcggl8Node02Cell / ClassloaderExample.ear /
ClassloaderExampleWeb.war / WEB-INF / lib / VersionCheckerV1.jar

WAS Module Jar Class Loader (Application class loader):

file: / C: / WebSphere / AppServer / profiles / AppSrv02 /
installedApps / kcggl8Node02Cell / ClassloaderExample.ear /
VersionCheckerV2.jar

12.5.4 Step 4: Sharing utility JARs using shared libraries

In this situation, the VersionCheckerV2.jar file is used by a single application. What if you wanted to share it among multiple applications? Of course, you could package it within each EAR file. But changes to this utility JAR file require redeploying all applications again. To avoid this, you can externalize global utility JARs using a shared library.

Shared libraries can be defined at the cell, node, application server, and cluster levels. Once you have defined a shared library, you must associate it to the class loader of an application server, an application, or an individual Web module. Depending on the target the shared library is assigned to, WebSphere will use the appropriate class loader to load the shared library.

You can define as many shared libraries as you want. You can also associate multiple shared libraries with an application, Web module, or application server.

Using shared libraries at the application level

To define a shared library named `VersionCheckerV2_SharedLib` and associate it to our `ClassLoaderTest` application, do the following:

1. In the administrative console, select **Environment** → **Shared Libraries**.
2. Select the scope at which you want this shared library to be defined, such as `Cell`, and click **New**.
3. Specify the properties shown in Figure 12-10 on page 821.

Shared Libraries ?

[Shared Libraries > New](#)

Use this page to define a container-wide shared library that can be used by deployed applications.

Configuration

General Properties

* **Scope**
cells:kcgg1d8Node02Cell

* **Name**
VersionCheckerV2_SharedLib

Description

* **Classpath**
C:\henrik\VersionCheckerV2.jar

Native Library Path

Apply **OK** **Reset** **Cancel**

Figure 12-10 Shared library configuration

- Name: Enter VersionCheckerV2_SharedLib.
 - Class path: Enter the list of entries on the class path. Press Enter between each entry. We highly recommend that if you need to provide an absolute path that you use WebSphere variables, such as %FRAMEWORK_JARS%/VersionCheckerV2.jar. Make sure that you declare this variable at the same scope as the shared library for cell, node, server, or cluster.
 - Native library path: Enter a list of DLLs and .so files for use by the JNI code.
4. Click **OK**.
 5. Select **Applications** → **Enterprise Applications**.
 6. Select the **ClassloadersExample** application.
 7. In References, select **Shared library references**.
 8. Select **ClassLoaderExample** in the Application row.
 9. Click **Reference shared libraries**.
 10. Select the **VersionCheckerV2_SharedLib** and click the >> button to move it to the Selected column, as shown in Figure 12-11.

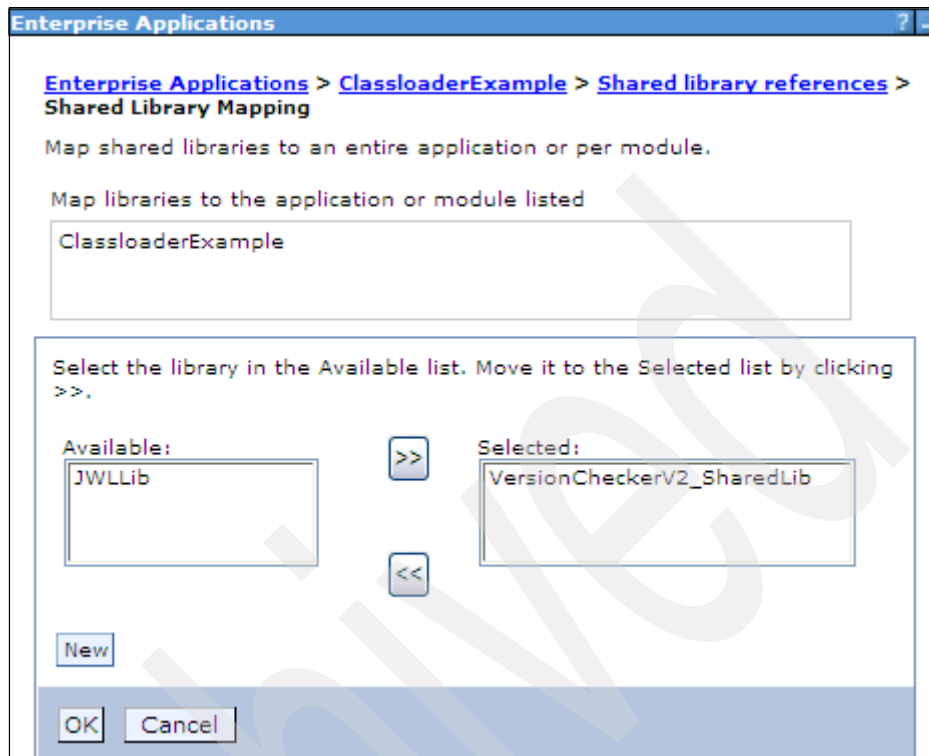


Figure 12-11 Assigning a shared library

11. Click **OK**.

12. The shared library configuration window for the ClassloaderExample application should now look like Figure 12-12.

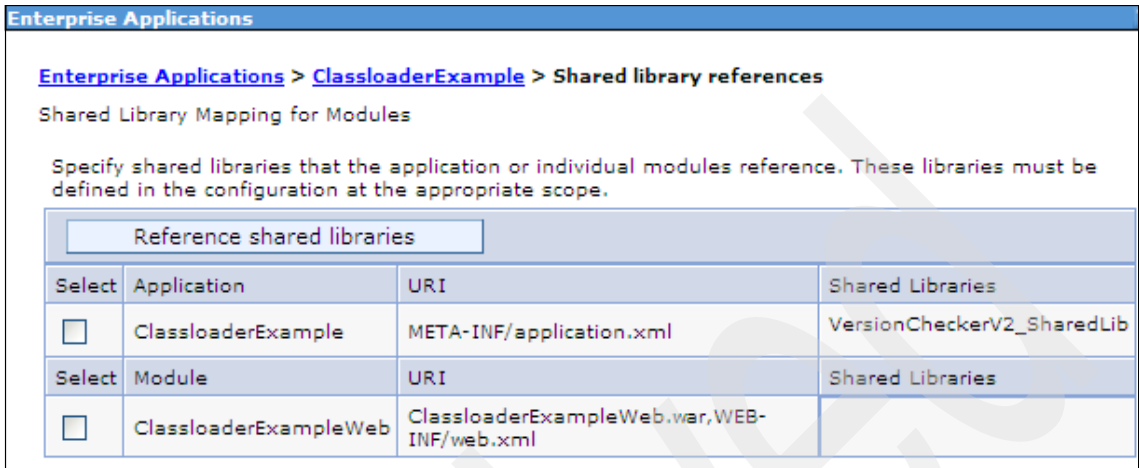


Figure 12-12 Shared library assigned to ClassloaderExample application

13. Click **OK** and then save the configuration.

If we now remove the VersionCheckerV2.jar file from the root of the EAR file, and remove the reference to it from the EJB module's manifest file, and restart the application server, we see the results in Example 12-12. Remember the class loader order for the Web module is still Classes loaded with application class loader first (PARENT_LAST).

Example 12-12 Class loader: Example 5

VersionChecker called from Servlet

VersionChecker is v1.0.

Loaded by com.ibm.ws.classloader.CompoundClassLoader@2e602e60

Local ClassPath:

C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcggl8Node02Cell1\ClassloaderExample.ear\ClassloaderExampleWeb.war\WEB-INF\classes;C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcggl8Node02Cell1\ClassloaderExample.ear\ClassloaderExampleWeb.war\WEB-INF\lib\VersionCheckerV1.jar;C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcggl8Node02Cell1\ClassloaderExample.ear\ClassloaderExampleWeb.war

Delegation Mode: **PARENT_LAST**

VersionChecker called from EJB

VersionChecker is **v2.0**.

Loaded by com.ibm.ws.classloader.**CompoundClassLoader@19141914**

Local ClassPath:

C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcg1d8Node02Cell1\ClassLoaderExample.ear\ClassLoaderExampleEJB.jar;C:\henrik\VersionCheckerV2.jar

Delegation Mode: PARENT_FIRST

As expected, because of the delegation mode for the Web module, the VersionCheckerV1.jar file was loaded when the servlet needed the VersionChecker class. When the EJB needed the VersionChecker class, it was loaded from the shared library, which points to C:\henrik\VersionCheckerV2.jar.

If we would like the Web module to also use the shared library, we would just restore the class loader order to the default, Classes loaded with parent class loader first, for the Web module.

Using shared libraries at the application server level

A shared library can also be associated to an application server. All applications deployed on this server see the code listed on that shared library. To associate a shared library to an application server, you must first create an additional class loader for the application server as follows:

1. Select an application server.
2. In the Server Infrastructure section, expand the **Java and Process Management**. Select **Class loader**.
3. Choose **New**, and select a class loader order for this class loader, Classes loaded with parent class loader first (PARENT_FIRST) or Classes loaded with application class loader first (PARENT_LAST). Click **Apply**.
4. Click the class loader that is created.
5. Click **Shared library references**.
6. Click **Add**, and select the library you want to associate to this application server. Repeat this operation to associate multiple libraries to this class loader. For our example, we selected the **VersionCheckerV2_SharedLib** entry.
7. Click **OK**.

8. Save the configuration.
9. Restart the application server for the changes to take effect.

Because we have now attached the VersionCheckerV2 shared library to the class loader of the application server, we obtain the results in Example 12-13.

Example 12-13 Class loader: Example 6

VersionChecker called from Servlet

VersionChecker is **v1.0**.

Loaded by com.ibm.ws.classloader.**CompoundClassLoader@40c240c2**

Local ClassPath:

C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcg1d8Node02Cell1\ClassLoaderExample.ear\ClassLoaderExampleWeb.war\WEB-INF\classes;C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcg1d8Node02Cell1\ClassLoaderExample.ear\ClassLoaderExampleWeb.war\WEB-INF\lib\VersionCheckerV1.jar;C:\WebSphere\AppServer\profiles\AppSrv02\installedApps\kcg1d8Node02Cell1\ClassLoaderExample.ear\ClassLoaderExampleWeb.war

VersionChecker called from EJB

VersionChecker is **v2.0**.

Loaded by com.ibm.ws.classloader.**ExtJarClassLoader@7dee7dee**

Local ClassPath: **C:\henrik\VersionCheckerV2.jar**

Delegation Mode: PARENT_FIRST

The new class loader we defined is called the ExtJarClassLoader and it has loaded the VersionCheckerV2.jar file when requested by the EJB module.

The WAR class loader still continues to load its own version due to the delegation mode.

12.6 Additional class loader diagnostics

JVM Version 5.0 provides some additional options that can be useful when troubleshooting class loading problems. These options are set as command line arguments for the JVM. Select **Servers** → **Application Servers** → **<server_name>** and then expand the **Java and process management** section under the Server Infrastructure heading. Click the **Process Definition** link and then the **Java Virtual Machine** link. Enter the options below in the **Generic JVM arguments** field:

- ▶ -verbose:dynload

This option provides detailed information about classes being loaded. Information includes the class name and package, the JAR file name if the class is packaged in a JAR file, the size of the class, and the time it takes to load the class. The information is written to the native_stderr.log file. An example output looks like:

```
<Loaded com/itso/classloaders/VersionChecker>
< Class size 817; ROM size 728; debug size 0>
< Read time 0 usec; Load time 40 usec; Translate time 89 usec>
```

- ▶ -Dibm.cl.verbose=<name>

This option allows you to trace the way the class loaders find and load a given class. The name is the full name of the class, including the package name. By specifying -Dibm.cl.verbose=com.itso.classloaders.VersionChecker, the following output is printed to the SystemOut.log file when the VersionChecker class is loaded (note that the output has been truncated to fit):

```
[8/4/06 10:29:48:639 EDT] 00000024 SystemOut      0 ExtClassLoader
attempting to find com.itso.classloaders.VersionChecker
[8/4/06 10:29:48:639 EDT] 00000024 SystemOut      0 ExtClassLoader
using classpath
C:\WebSphere\AppServer\java\jre\lib\ext\CmpCrmf.jar;...C:\WebSphere
\AppServer\java\jre\lib\ext\whichclassloader2.jar;C:\WebSphere\AppSe
rver\java\jre\lib\ext\whichclassloader3.jar
[8/4/06 10:29:48:639 EDT] 00000024 SystemOut      0 ExtClassLoader
could not find com/itso/classloaders/VersionChecker.class in
C:\WebSphere\AppServer\java\jre\lib\ext\CmpCrmf.jar
...
...
[8/4/06 10:29:48:639 EDT] 00000024 SystemOut      0 ExtClassLoader
could not find com.itso.classloaders.VersionChecker
[8/4/06 10:29:48:639 EDT] 00000024 SystemOut      0 AppClassLoader
attempting to find com.itso.classloaders.VersionChecker
```

```
[8/4/06 10:29:48:639 EDT] 00000024 SystemOut      0 AppClassLoader
using classpath
C:\WebSphere\AppServer\profiles\AppSrv02\properties;....
[8/4/06 10:29:48:649 EDT] 00000024 SystemOut      0 AppClassLoader
could not find com/itso/classloaders/VersionChecker.class in
C:\WebSphere\AppServer\profiles\AppSrv02\properties
...
...
[8/4/06 10:29:48:649 EDT] 00000024 SystemOut      0 AppClassLoader
could not find com.itso.classloaders.VersionChecker
[8/4/06 10:29:48:649 EDT] 00000024 SystemOut      0
com.ibm.ws.bootstrap.ExtClassLoader attempting to find
com.itso.classloaders.VersionChecker
[8/4/06 10:29:48:649 EDT] 00000024 SystemOut      0
com.ibm.ws.bootstrap.ExtClassLoader using classpath
C:\WebSphere\AppServer\java\lib;....
...
...
com.ibm.ws.bootstrap.ExtClassLoader could not find
com/itso/classloaders/VersionChecker.class in
C:\WebSphere\AppServer\java\lib
...
...
```


Packaging applications

In this chapter, we show you how to perform some common tasks involved in packaging a J2EE application. For this purpose, we will use the Plants by WebSphere sample application that ships with WebSphere Application Server and the Application Server Toolkit.

This chapter includes the following topics:

- ▶ Plants by WebSphere sample application
- ▶ Packaging using the Application Server Toolkit
- ▶ Setting application bindings
- ▶ IBM EJB extensions: EJB caching options
- ▶ IBM EJB extensions: EJB access intents
- ▶ IBM EJB extensions: inheritance relationships
- ▶ IBM Web module extensions
- ▶ IBM EAR extensions: Sharing session context
- ▶ Exporting the PlantsByWebSphere EAR file
- ▶ WebSphere Enhanced EAR
- ▶ Packaging recommendations

13.1 Plants by WebSphere sample application

Plants by WebSphere is an Internet storefront that specializes in the sale of plants and gardening tools. This Sample application uses many of the Java 2 Platform, Enterprise Edition (J2EE) and WebSphere Application Server functions, including enterprise beans, servlets, and JavaServer Pages (JSP) technology.

Using the Plants by WebSphere storefront, customers can open accounts, browse for items to purchase, view product details, and place orders. The Plants by WebSphere application uses container-managed persistence (CMP), container-managed relationships (CMR), stateless session beans, a stateful session bean, JSP pages, and servlets.

We will not go into details on how Plants by WebSphere application works. For more detailed information, refer to the WebSphere Information Center. If you have installed the sample applications on your system, you can also find information at:

<http://localhost:9080/WSsamples>

When Plants by WebSphere is installed and configured properly, it can be invoked at:

<http://localhost:9080/PlantsByWebSphere>

The Plants by WebSphere EAR file consists of the modules show in Figure 13-1.

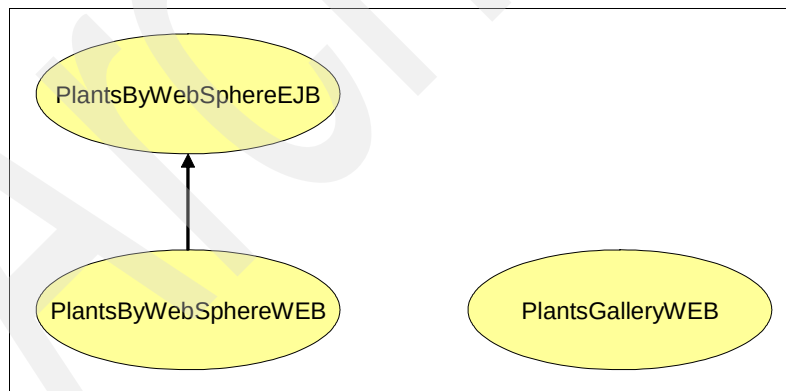


Figure 13-1 Plants By WebSphere modules

The Plants by WebSphere EAR file has one EJB module, and two Web modules. The PlantsByWebSphereWEB is the Web module where the servlets, JSPs, and

images are kept. The PlantsGalleryWEB module contains documentation for the sample, but no executables or Java code.

13.1.1 Plants by WebSphere resources used

To run successfully, Plants by WebSphere requires the following resources to be configured:

- ▶ JAAS authentication alias
 - Scope: Cell level
 - Name: <cellname>/samples
 - User ID: samples
 - Password: samples
- ▶ JDBC provider
 - Scope: Server level
 - Name: Samples Derby JDBC Provider (XA)
 - Implementation class: org.apache.derby.jdbc.EmbeddedXADataSource
- ▶ Data source
 - Scope: Server level
 - JDBC provider: Samples Derby JDBC Provider (XA)
 - Name: PLANTSDB
 - JNDI name: jdbc/PlantsByWebSphereDataSource
 - Database name:
\${APP_INSTALL_ROOT}/\${CELL}/PlantsByWebSphere.ear/Database/PLANTSDB
- ▶ Mail session:
 - Scope: Server level
 - Name: PlantsByWebSphere Mail Session
 - JNDI name: mail/PlantsByWebSphere
 - Type: javax.mail.Session

These are the resources configured when you install the samples during installation of WebSphere. Because we will change the Plants by WebSphere application to use an IBM DB2 database instead of the default Derby (Cloudscape) database, our resources will look a little bit different.

13.2 Packaging using the Application Server Toolkit

New in V6.1: The Application Server Toolkit shipped with V6.1 is now a full-blown integrated development environment (IDE). It can be used to build, test, and deploy J2EE applications on a WebSphere Application Server V6.1 environment (but not on any previous release). It has support for all J2EE artifacts supported by WebSphere Application Server V6.1, such as servlets, JSPs, EJBs, XML, Web services, and so on, and includes support for developing Java 5.0 applications.

To illustrate packaging techniques, we will import the source code for the Plants by WebSphere into the Application Server Toolkit, build it, and go through the various aspects of packaging an application for deployment. The application can then be exported as an EAR file for deployment.

The samples must be selected for installation during the WebSphere install. If you have installed the samples, the source code can be found in `<was_home>\samples\src\PlantsByWebSphere`.

13.2.1 Import source code

To work with the Plants by WebSphere sample application in the Application Server Toolkit, we first need to create an Enterprise Application project and then import the source code.

1. Select **Start** → **Programs** → **IBM WebSphere** → **Application Server Toolkit V6.1** → **Application Server Toolkit**.
2. When asked for a default location for a workspace, browse to a suitable directory and then click **OK**. Do not check the **Use this as the default and do not ask again** check box.
3. When the toolkit has launched, close the Welcome page by clicking the X in the Welcome tab, as in Figure 13-2.

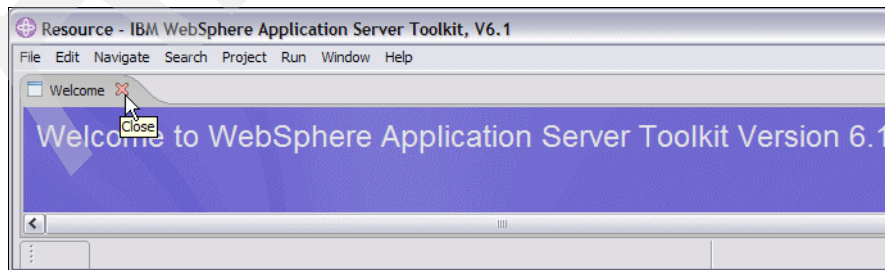


Figure 13-2 Welcome page

4. Select **File** → **New** → **Other**.
5. Expand **J2EE**, select **Enterprise Application Project**, and then click **Next**.
6. In the Project name field, enter PlantsByWebSphere, and then click **Next**.
7. On the Select Project Facets page, keep the defaults (EAR, WebSphere Application Co-existence, and WebSphere Application Extended), and click **Next**.
8. On the J2EE Modules to Add to the EAR page, click **New Module**.
9. On the New J2EE Module page, deselect the **Application client module** and **Connector module**. Keep the PlantsByWebSphereEJB and PlantsByWebSphereWeb projects as is. Then click **Finish**.
10. Back on the J2EE Modules to Add to the EAR, click **Finish**.
11. When the projects have been created, click **Yes** on the dialog box asking you to switch to the J2EE perspective.

You should now see the J2EE perspective with the Project Explorer view, as shown in Figure 13-3 on page 834.

Note that we did not create a project for the PlantsGalleryWeb module, because that only holds documentation and we do not need that for our example.

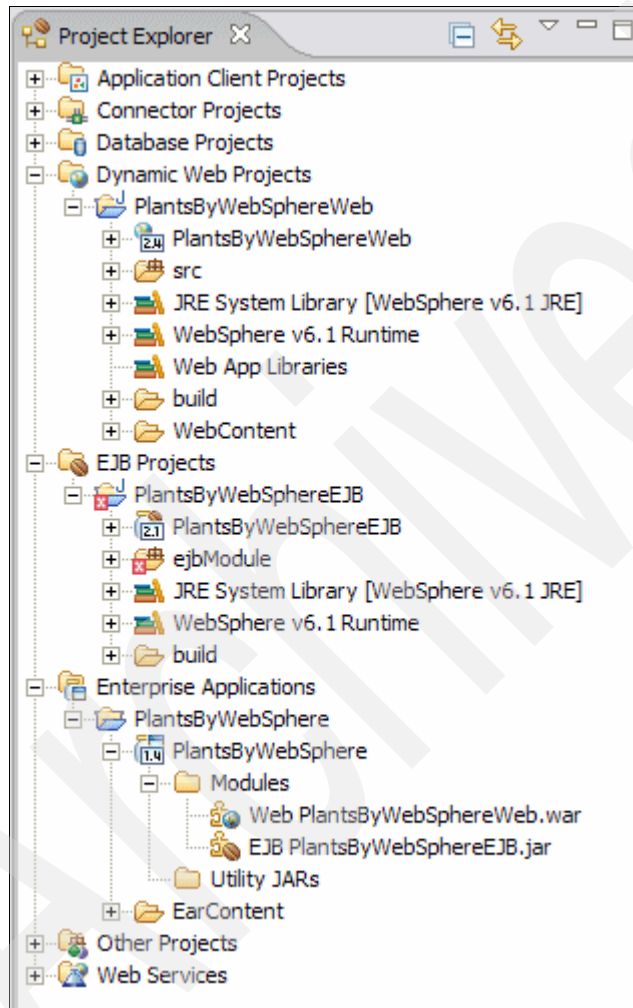


Figure 13-3 Project Explorer view with PlantsByWebSphere projects created

When the necessary projects are created, it is time to import the source code.

The defaults for Web modules is to keep source code in the src folder and the compiled Java classes in the build folder. For EJB projects, the source code is kept in the ejbModule folder and the compiled classes in the build folder.

We will start with the EJB source code first:

1. Select **File** → **Import**.
2. Select **File system** from the list, and then click **Next**.
3. For the From directory field, click **Browse** and select the <was_home>\samples\src\PlantsByWebSphere\PlantsByWebSphereEJB\ejbModule directory, and then click **OK**.
4. Select the **ejbModule** in the left pane.
5. For the Into folder field, click **Browse** and select the **PlantsByWebSphereEJB/ejbModule** folder, and click **OK**.
6. Keep the other options as is (Create selected folders only is selected only) and then click **Finish**. See Figure 13-4.

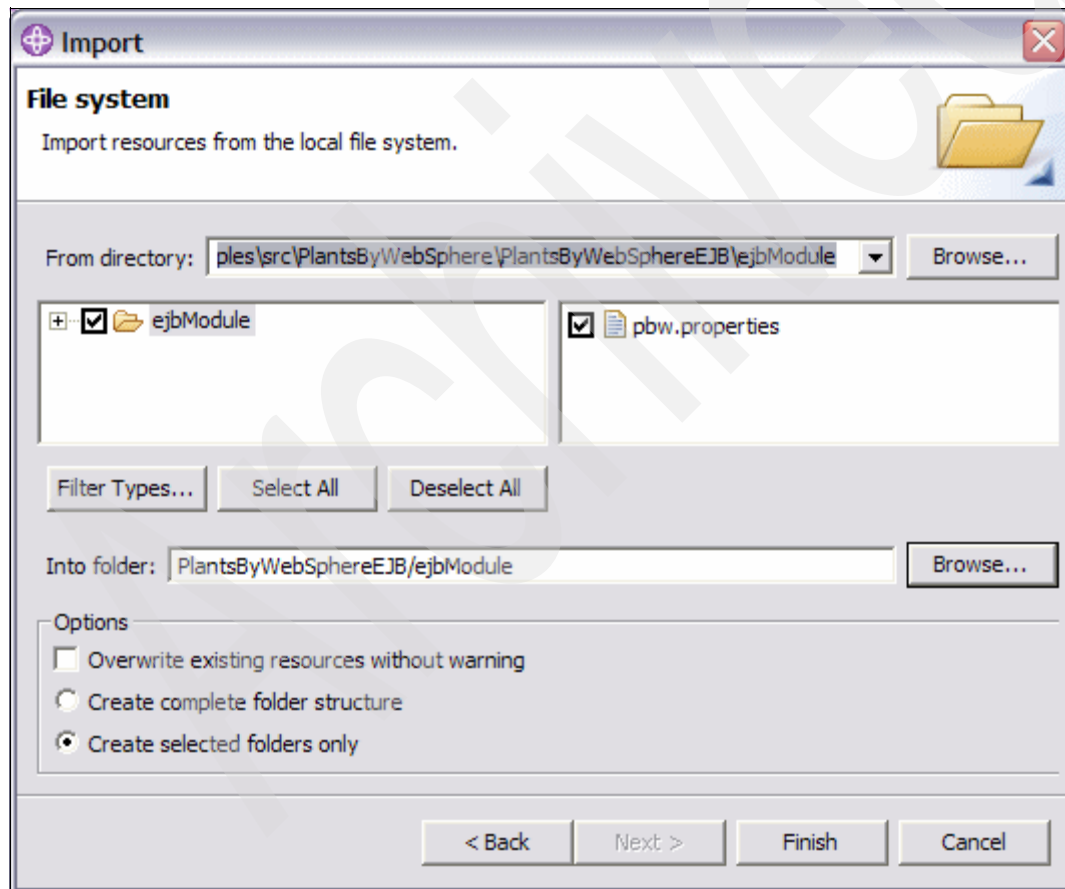


Figure 13-4 Importing EJB module source code

7. When the question about overwriting existing resources appear, click **Yes To All**.
8. Repeat the process to import the Web module source code from `<was_home>\samples\src\PlantsByWebSphere\PlantsByWebSphereWEB\JavaSource` into the `PlantsByWebSphereWeb/src` folder, and the Web content from `<was_home>\samples\src\PlantsByWebSphere\PlantsByWebSphereWEB\WebContent` into the `PlantsByWebSphereWeb/WebContent` folder

Note: When you import source code or an EAR file, the workspace rebuilds the project. The build process runs as a separate background thread and can take a minute or two, depending on the application. In the lower right corner of the Application Server Toolkit window, a progress indicator tells you what is happening. When rebuild is complete, any error messages will appear in the Problems view.

If you click the **Problems** tab, you see that the workspace has a lot of problems. This is because the Web module depends on classes in the EJB module, but because we imported the source code, this meta data was not included, so we now need to set up the J2EE dependencies for the Web module to find its required classes.

1. In the Project Explorer, right-click the **PlantsByWebSphereWeb** project and select **Properties**.
2. Select the **J2EE Module Dependencies** section.
3. Select the **PlantsByWebSphereEJB.jar** option in the dialog. When selected, the Manifest Class-Path is updated so that the Web project references the EJB project. See Figure 13-5. Click **OK** when done.

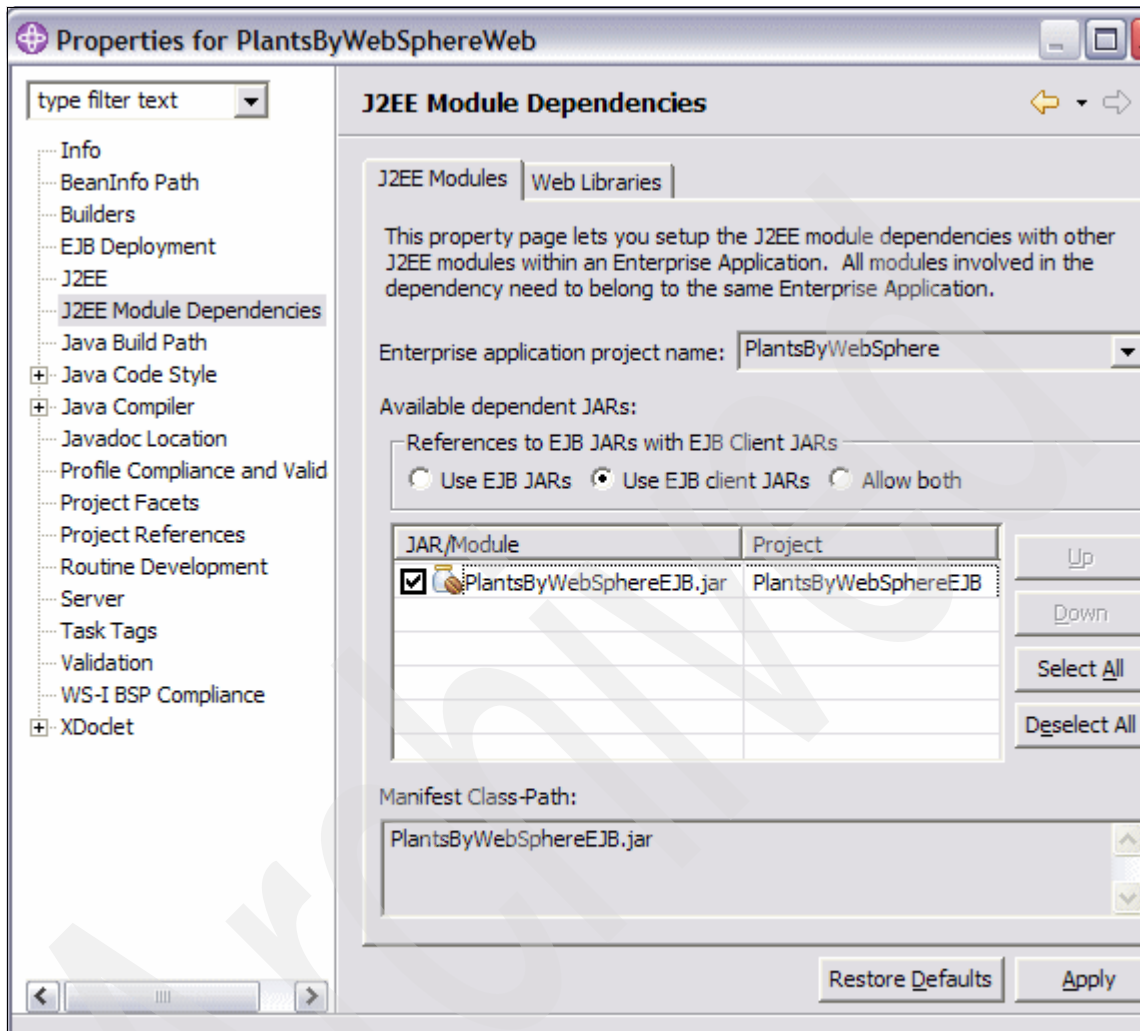


Figure 13-5 Configuring J2EE Module Dependencies

When the class path has been set up correctly and the project rebuilt, all errors in the Problems view should be gone. Still, there are a lot of warning messages.

None of these warnings are critical, though, so you do not need to bother about them. However, if you do want to get rid of them, you can select **Window** → **Preferences** → **Java** → **Compiler** → **Errors/Warnings**. Then expand the Unnecessary code section and change the settings for **Local variable never read** and **Unused local or private members** settings to **Ignore**.

The Application Server Toolkit will still warn you about a `serialVersionUID` field missing from several classes. This is because the toolkit now warns if a class is `Serializable` (implements `java.io.Serializable` interface) but does not have a line like:

```
private static final long serialVersionUID = 12345678L;
```

To fix this warning, you would add the above line to all serializable classes where it is missing. We will not do that here because the warning is not a severe one, so instead expand the **Potential programming problems** section and change the setting for **Serializable class without serialVersionUID** to **Ignore**. After a full rebuild of the workspace, there should be no warnings in the Problems view.

Note: The `serialVersionUID` is used for “version control” of serializable classes. Whenever you make a change to the class that is incompatible with the previous version, you should also change this number. Best practices dictates you should run the **serialver** utility (found in the JDK bin directory) to generate the number, but it is perfectly fine just to pick a number out of thin air. If the `serialVersionUID` line is not present, there is a small performance hit, because the JVM then needs to generate a number on the fly when the class is used.

Tip: When using the Application Server Toolkit, keep in mind the following:

- ▶ To perform a complete rebuild of your project(s), select **Project** → **Clean** and then select either to clean current or all projects. This will remove all build problems from the Problems view and perform a complete re-build of the selected projects. This sometimes removes errors and warnings in the Problems view.
- ▶ As you update and save modules in the toolkit, the contents of the modules are automatically validated and problems are listed in the Tasks view. You can also manually invoke validation of modules by selecting any module and choosing **Run Validation** from the context menu. To verify the settings for validation, select **Window** → **Preferences** and click **Validation**.

13.2.2 Working with deployment descriptors

Information describing a J2EE application and how to deploy it into a J2EE container is stored in XML files called deployment descriptors. An EAR file normally contains multiple deployment descriptors, depending on the modules it contains. Figure 13-6 shows a schematic overview of a J2EE EAR file. In this figure, the various deployment descriptors are designated with DD after their name.

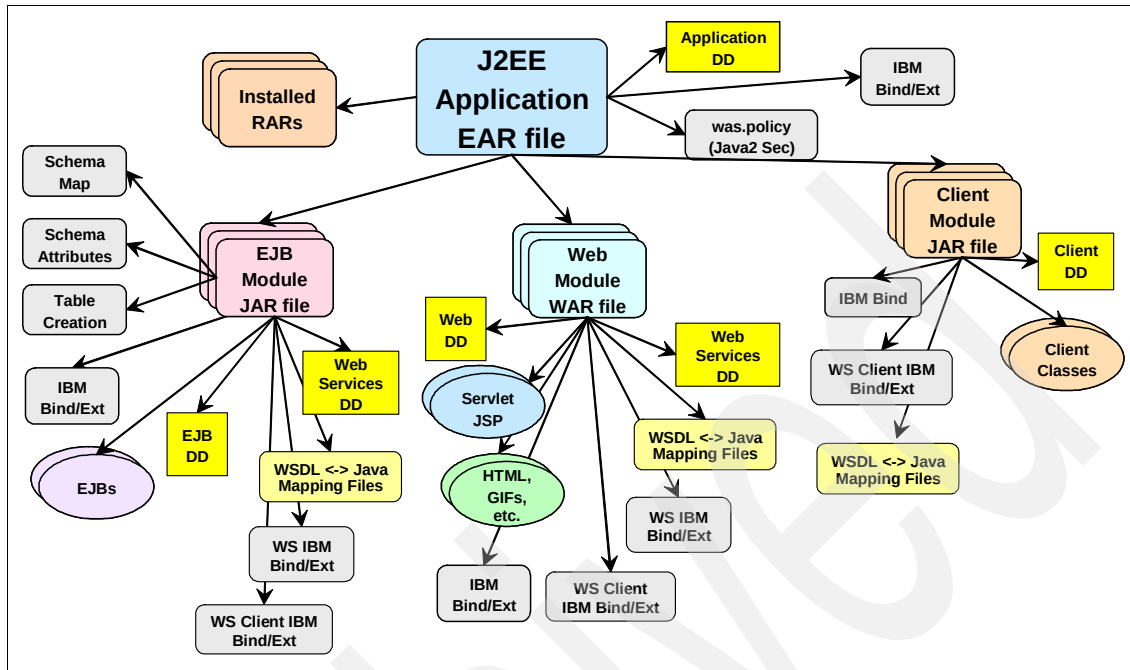


Figure 13-6 J2EE EAR file structure

The deployment descriptor of the EAR file itself is stored in the META-INF directory in the root of the EAR and is called application.xml. It contains information about the modules that make up the application.

The deployment descriptors for each module are stored in the META-INF directory of the module and are called web.xml (for Web modules), ejb-jar.xml (for EJB modules), ra.xml (for resource adapter modules), and application-client.xml (for application client modules). These files describe the contents of a module and allow the J2EE container to configure things like servlet mappings, JNDI names, and so forth.

Classpath information specifying which other modules and utility JARs are needed for a particular module to run is stored in the manifest.mf file also in the META-INF directory of the modules.

In addition to the standard J2EE deployment descriptors, EAR files produced by the Application Server Toolkit can also include additional WebSphere-specific information used when deploying applications to WebSphere environments. This supplemental information is stored in files called ibm-xxx-xxx-xxx.xmi, also in the META-INF directory of the respective modules. Examples of information in the

IBM-specific files are IBM extensions like servlet reloading and EJB access intents.

WebSphere Application Server V6.0 and V6.1 can also store deployment-related information (such as data sources, class loader settings, and so on) as part of an Enhanced EAR file. This information is stored in an `ibmconfig` subdirectory of the EAR file's `META-INF` directory.

The Application Server Toolkit has easy-to-use editors for working with all deployment descriptors. The information that goes into the different files are shown on one page in the GUI, eliminating the need to be concerned about what information is put into what file. However, if you are interested, you can click the Source tab of the deployment descriptor editor to see the text version of what is stored in that descriptor. For example, if you open the EJB deployment descriptor, you will see settings that are stored across multiple deployment descriptors for the EJB module, including:

- ▶ The EJB deployment descriptor, `ejb-jar.xml`
- ▶ The extensions deployment descriptor, `ibm-ejb-jar-ext.xmi`
- ▶ The bindings file, `ibm-ejb-jar-bnd.xmi` files
- ▶ The access intent settings, `ibm-ejb-access-bean.xmi`

To work with a deployment descriptor, do the following:

1. Open the J2EE perspective.
2. In the J2EE Project Explorer view, expand the project category (Enterprise Applications, EJB Projects, Dynamic Web Projects, and so on), and then expand the module you want to work with.
3. Double-click the deployment descriptor to open the editor for it. The deployment descriptor is module name with a version number in front. The version number refers to the J2EE specification level for the module. See Figure 13-7.

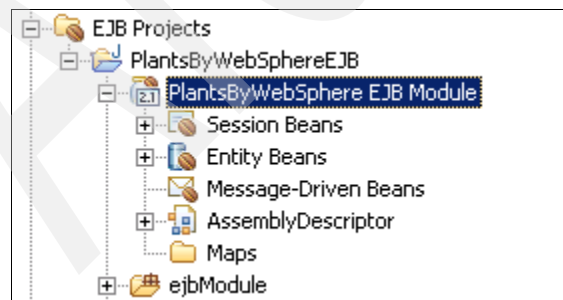


Figure 13-7 EJB deployment descriptor

4. Figure 13-8 on page 841 shows the deployment descriptor for the PlantsByWebSphere EJB module open with the deployment descriptor editor.

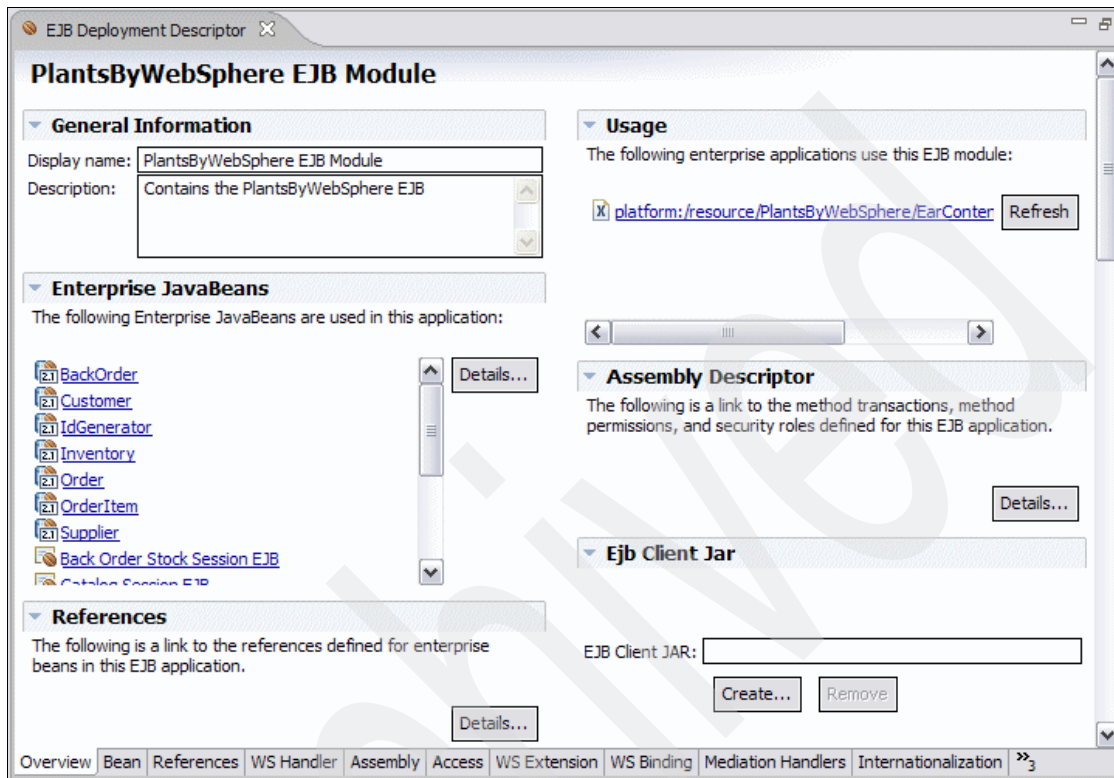


Figure 13-8 EJB deployment descriptor

The editor shows you information stored in all the relevant deployment descriptor files on the appropriate tabs. The descriptor files are kept in the META-INF directory of the module you are editing.

When you have made changes to a deployment descriptor, save it by pressing Ctrl+S and then close it.

13.3 Setting application bindings

At packaging time, you create references to resources. For an application to run, you need to bind these references to the real resources, such as JDBC data sources, created from the administrative console. This needs to be done for EJB references and resource references. You also need to define the enterprise bean's JNDI names, and security roles.

Bindings can be defined at development or deployment time. Most likely, developers will deliver a preconfigured EAR file that will then be modified at deployment time by the deployment team to suit the target environment.

All binding definitions are stored in the `ibm-xxx-bnd.xml` files, where `xxx` can be `ejb-jar`, `Web`, `application`, or `application-client`.

In the next steps, you define the following bindings using the Application Server Toolkit:

- ▶ EJB JNDI names
- ▶ EJB references
- ▶ Data source for entity beans

All sections below assume that you have started the Application Server Toolkit and imported the Plants by WebSphere source, as described in “Import source code” on page 832.

13.3.1 Defining EJB JNDI names

For each session and entity bean, you must specify a JNDI name. This name is used to bind the EJB home object to an entry in the global JNDI name space. The bind happens automatically when the application server starts.

You can bind your EJBs anywhere you want in the JNDI name space, but best practice dictates they should be bound under the `ejb` subcontext, such as `ejb/Order` for the Order EJB. However, the Plants by WebSphere binds its EJBs to the `plantsby` subcontext in the JNDI root, so this is what we will show you here.

For clarity, however, when developing your own application, we recommend that you place all enterprise bean JNDI names for an application in a separate subcontext, such as `ejb/PlantsBy`. You can find the JNDI names for the Plants by WebSphere session and entity EJBs in Table 13-1.

Table 13-1 Plants by WebSphere enterprise bean JNDI names

EJB Name	JNDI Name
BackOrder entity bean	plantsby/BackOrderHome
Customer entity bean	plantsby/CustomerHome
IdGenerator entity bean	plantsby/IdGeneratorHome
Inventory entity bean	plantsby/InventoryHome
Order entity bean	plantsby/OrderHome
OrderItem entity bean	plantsby/OrderItemHome
Supplier entity bean	plantsby/SupplierHome
Back Order Stock Session EJB	plantsby/BackOrderStockHome
Catalog Session EJB	plantsby/CatalogHome
Login Session EJB	plantsby/LoginHome
Mailer Session EJB	plantsby/MailerHome
Report Generator Session EJB	plantsby/ReportGeneratorHome
Reset Database Session EJB	plantsby/ResetDBHome
Shopping Cart Session EJB	plantsby/ShoppingCartHome
Suppliers	plantsby/SuppliersHome

Use this table and the instructions below to define a JNDI name for each Plants by WebSphere enterprise bean:

1. In the Project Explorer view, expand the EJB Projects section.
2. Expand the **PlantsByWebSphereEJB** project and then double-click the Deployment Descriptor (**2.1 PlantsByWebSphere EJB Module**).

3. Click the **Bean** tab, and then click the **BackOrder EJB**. The EJB deployment descriptor is shown in Figure 13-9.
4. Look for the WebSphere Bindings section in the editor. In the JNDI name field, enter plantsby/BackOrderHome.

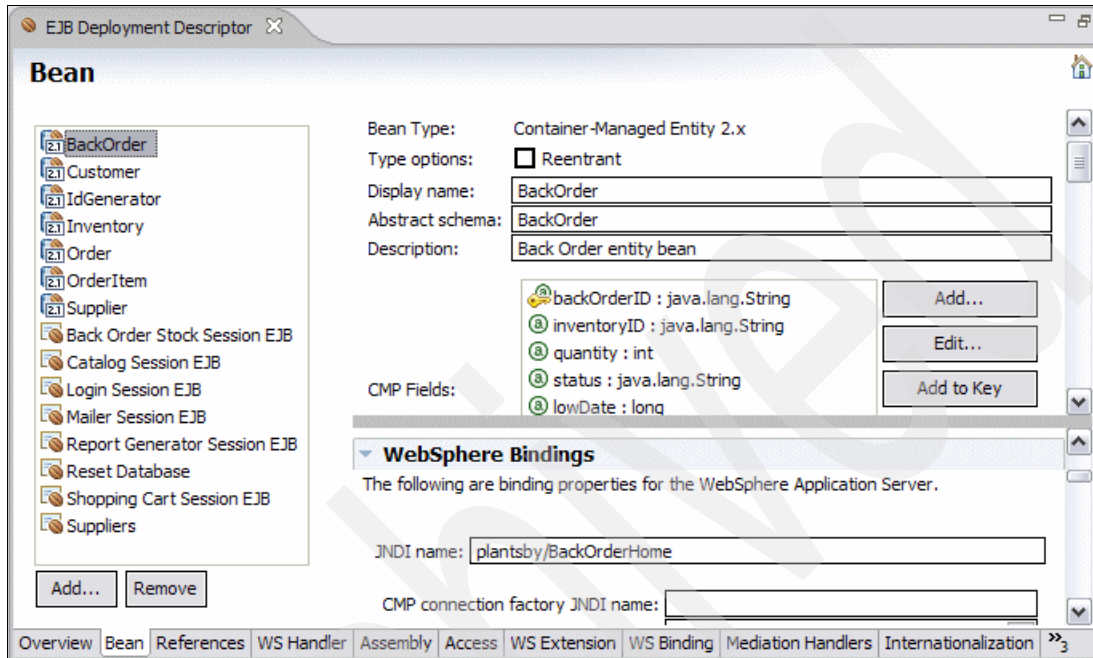


Figure 13-9 Defining EJB JNDI names

5. Repeat these steps for each of the session and entity enterprise beans in the EJB module.
6. Save the deployment descriptor.

13.3.2 Binding EJB and resource references

An EJB client can define *EJB references*, logical names, or nicknames, used by the client to find the EJB homes. When using references, the client can hard code the name of the reference. Then, during deployment, the reference is mapped to the real name in the JNDI namespace to which the EJB is bound. A reference to an EJB specifies either the local or remote home of the EJB.

For remote home interfaces, using EJB references is an option, but also a best practice. For local home interfaces, however, it is a must, because using an EJB reference is the only way an EJB client can look up a local home interface.

J2EE applications also use other kinds of references, such as resource references to look up data sources.

Table 13-2 on page 845 lists some of the EJB references used by the Plants by WebSphere sample.

Table 13-2 EJB and resource references: JNDI names list

EJB name	Reference name	JNDI Name
BackOrder	ejb/Inventory	plantsby/InventoryHome
BackOrder	ejb/IdGenerator	plantsby/IdGeneratorHome
Inventory	ejb/BackOrderStock	plantsby/BackOrderStockHome
Order	ejb/OrderItem	plantsby/OrderItemHome
Order	ejb/IdGenerator	plantsby/IdGeneratorHome
Login Session EJB	ejb/Customer	plantsby/CustomerHome

For example, to log on a user, the Login Session EJB uses the Customer EJB to find the user in a database. The Login Session EJB looks up the `ejb/Customer` entry in the JNDI name space. This is then mapped to `plantsby/CustomerHome`, which is the JNDI name the Customer EJB is bound to.

So the EJB reference is an alias used to find the target EJB. When using EJB references, the name of the reference must be prefixed with `java:comp/env`. So the Login Session EJB would do a JNDI lookup with the name `java:comp/env/ejb/Customer` to get the EJB reference for Customer.

Follow these steps to bind an EJB to a JNDI name:

1. In the EJB Deployment Descriptor for the `PlantsByWebSphereEJB` module (see Figure 13-9 on page 844), click the **References** tab.

2. Click the plus sign next to the Login Session EJB and select the **ejb/Customer** reference, as shown in Figure 13-10.

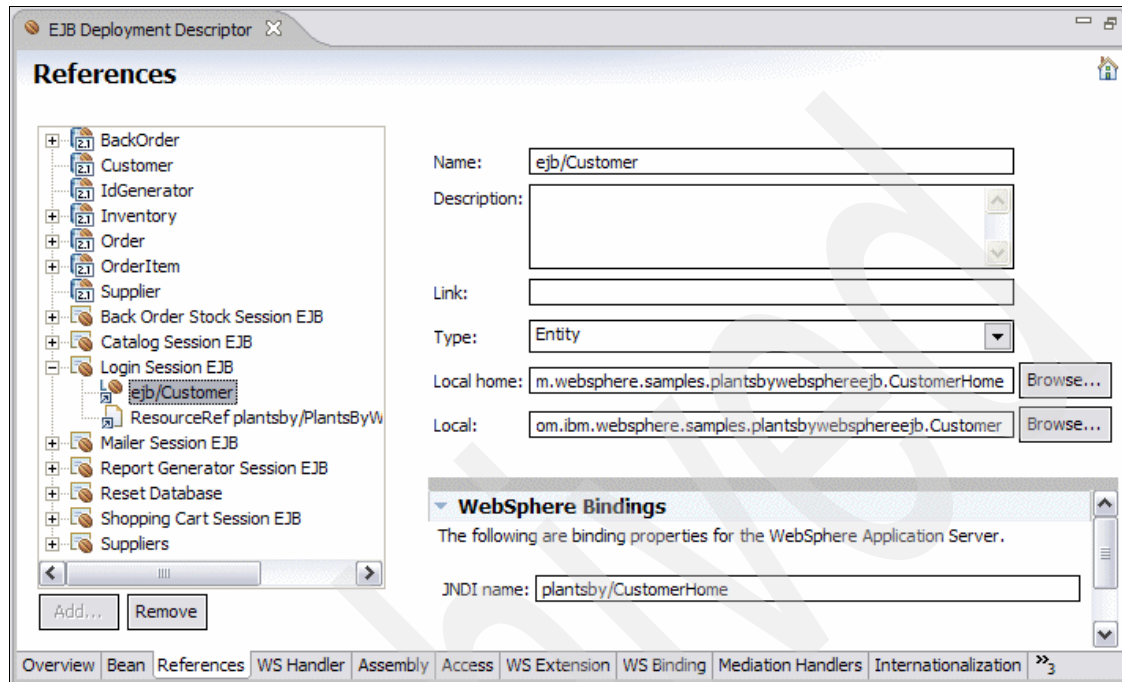


Figure 13-10 Setting EJB References bindings

3. In the WebSphere bindings section, specify plantsby/CustomerHome.
4. Save the deployment descriptor.

Note: EJB reference bindings can be defined or overridden at deployment time in the administrative console for all modules except for application clients, for which you must use the Application Server Toolkit.

13.3.3 Defining data sources for entity beans

The entity beans in the Plants by WebSphere application are container-managed (CMP) EJBs. The EJB container handles the persistence of the EJB attributes in the underlying persistent store. You must specify which data store to use. This is done by binding an EJB module or an individual EJB to a data source. If you bind the EJB module to a data source, all EJBs in that module use the same data source for persistence. If you specify the data source at the EJB level, then this data source is used instead.

For the Plants by WebSphere application, the data source is bound at the EJB module level. The data source configured for the EJB must match the data source configured in the WebSphere environment. The JNDI name for this data source is eis/jdbc/PlantsByWebSphereDataSource_CMP. When a data source is defined in WebSphere, it can be marked as for use by container-manager persistence. If this option is selected the data source is then also mapped into the JNDI name space with a name like this.

To bind the Plants by WebSphere EJB module to this data source, follow these steps:

1. In the EJB Deployment Descriptor for the PlantsByWebSphereEJB module (see Figure 13-9 on page 844), click the **Overview** tab.
2. In the Overview tab, scroll down and find the WebSphere bindings section, as in Figure 13-11.

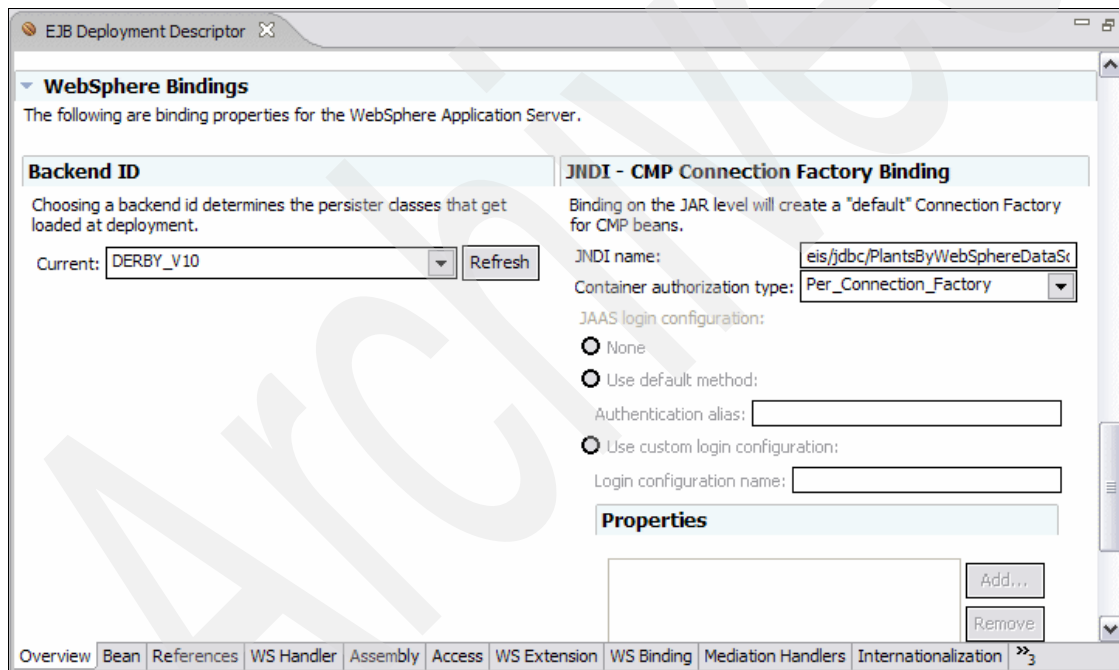


Figure 13-11 Specifying the default CMP data source for the EJB module

Check the following fields:

New in V6.1: The embedded Cloudscape has a Cloudscape v10.1.x code base, referred to as Derby. Derby is a product of the Apache Software Foundation (ASF) open source relational database project. The new Cloudscape includes Derby without any modification to the underlying source code. Learn more about Derby code at the Apache Derby Web site: <http://db.apache.org/derby/>

- Backend ID

In EJB 2.x, mapping and schema files make up a back end for EJB 2.x projects. The Plants by WebSphere sample ships with a Cloudscape back end defined. Leave this for now, though in “Creating a new database mapping and schema” on page 848, we will create a new back end for DB2 and change this binding to point it.

- JNDI name

Enter `eis/jdbc/PlantsByWebSphereDataSource_CMP` in the JNDI name field. This is the value the application uses to access the database. Note that this is the same, regardless of which back-end ID is used.

- Container authorization type

Select **Per_Connection_Factory** for the Container authorization type.

3. Save the deployment descriptor.

Tip: An EJB JAR can contain database mappings and EJB deployment code for multiple databases. Because we imported the source code for the Plants by WebSphere sample, there is currently no deployed code, but we will generate it for DB2 UDB. You can set which back-end ID will be used at run time in the WebSphere bindings section. This choice can also be overridden at deployment time.

Creating a new database mapping and schema

The Plants by WebSphere sample is configured to run against a Cloudscape database. However, for the purpose of showing how to create a new database back end and deployed code, we will configure it so it can also run against a DB2 database.

Creating the database mapping

First, you need to create a database mapping using the EJB project. To do this, perform the following steps:

1. Expand the **EJB Projects** section in the Project Navigator view and then expand the **PlantsByWebSphereEJB** project.
2. Right-click the deployment descriptor and select **EJB to RDB Mapping** → **Generate Map**.
3. Select **Create a new backend folder** and click **Next**.
4. Select **Top-down** and click **Next**.
5. On the Top-down mapping options page, select **DB2 UDB V8.2**, or the corresponding DB2 product and version you are using as the target database. Enter database name **PLANTS** and leave **NULLID** as the schema name. See Figure 13-12 on page 849.

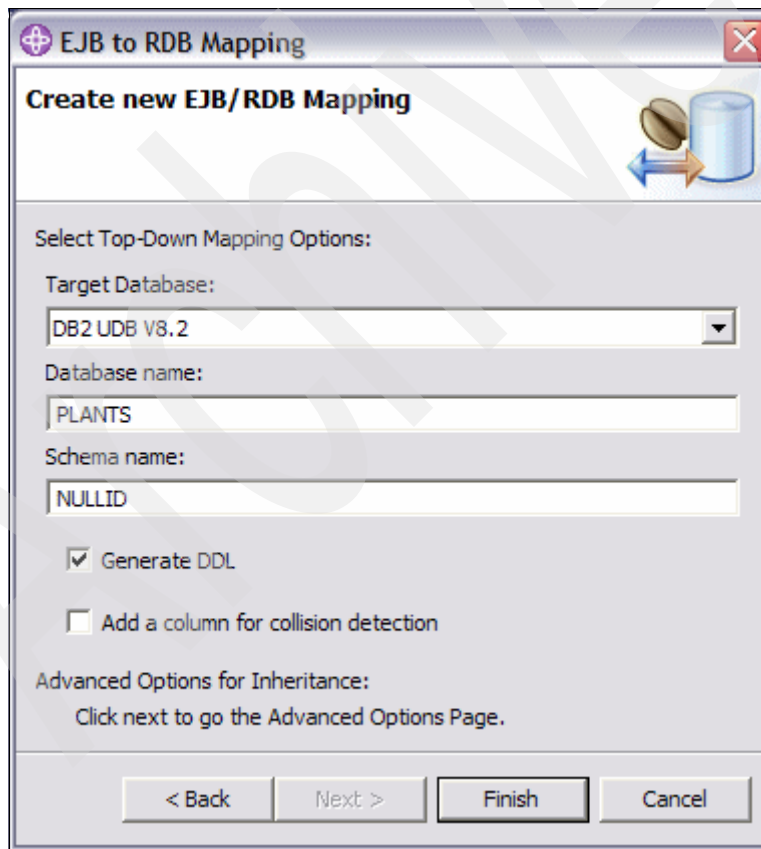


Figure 13-12 Generating a DB2 mapping

6. Click **Finish**.

The Application Server Toolkit database mapping editor (Map.mapxmi editor) opens, allowing you to make adjustments to the mapping between the fields of the entity EJBs and the database columns. We do not need to do that, so close the editor.

A Table.ddl file containing the script to set up the DB2 tables is created in the PlantsByWebSphereEJB\ejbModule\META-INF\backends\DB2UDBNT_V82_1 directory of the Application Server Toolkit workspace. You will need this script when creating the DB2 PLANTS database before deploying Plants by WebSphere.

The sample application stores images of the products you can buy in the INVENTORY table in the database. The field of the EJB that holds the image data is declared as a byte array (byte[]) and the Application Server Toolkit maps this to a DB2 type of VARCHAR (1000) FOR BIT DATA (as you can see if you open the database mapping editor on Map.mapxmi). We did not get this data type to work, so we changed it to a 100 KB large BLOB instead. To do this, you need to open the Database Table Editor. If you are familiar with the Application Server Toolkit or Rational Application Developer, you should pay attention, because in the Application Server Toolkit V6.1, it is not accessed in the same way.

7. Double-click the PLANTS.dbm file in the PlantsByWebSphereEJB\ejbModule\META-INF\backends\DB2UDBNT_V82_1 folder.
8. Click the plus sign that appears in front of PLANTS.dbm and expand the tree all the way down to the INVENTORY table and then expand the INVENTORY table as well.
9. Select the IMGBYTES column.
10. In the Properties view, click the **Type** tab, as shown Figure 13-13.

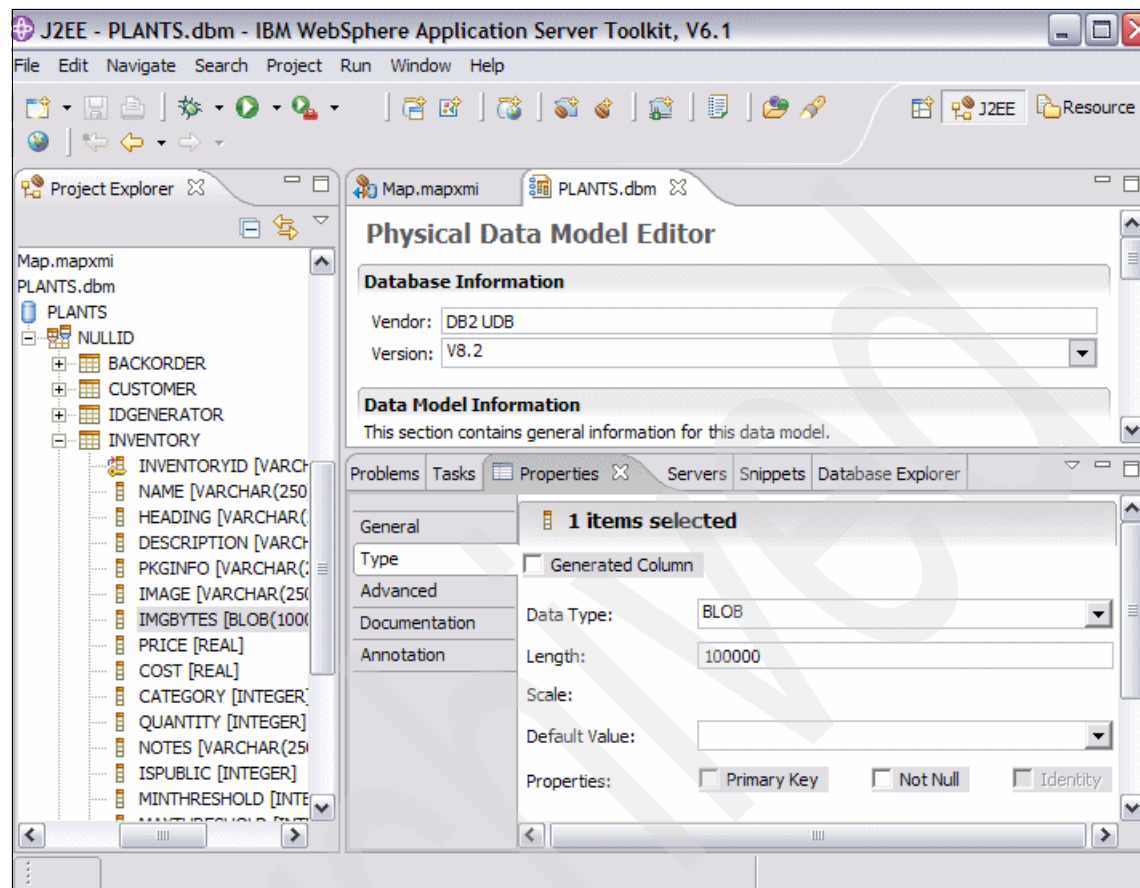


Figure 13-13 Table Editor

11. For data type, select **BLOB**. Then enter 100000 in the length field. Press Ctrl-S to save the change to the PLANTS.dbm file.
12. Because the mapping now has changed, we need to update the Table.ddl file. Right-click the **Deployment Descriptor (2.1 PlantsByWebSphere EJB Module)** project and select **EJB to RDB Mapping → Generate Schema DDL**.

13. The Table.ddl file in the PlantsByWebSphereEJB\ejbModule\META-INF\backends\DB2UDBNT_V82_1 folder is now updated. Example 13-1 shows part of the file.

Example 13-1 Plants by WebSphere Table.ddl

```
CREATE TABLE CUSTOMER (  
    CUSTOMERID VARCHAR(250) NOT NULL,  
    PASSWORD1 VARCHAR(250),  
    FIRSTNAME VARCHAR(250),  
    LASTNAME VARCHAR(250),  
    ADDR1 VARCHAR(250),  
    ADDR2 VARCHAR(250),  
    ADDR1CITY VARCHAR(250),  
    ADDR1STATE VARCHAR(250),  
    ADDR1ZIP VARCHAR(250),  
    PHONE VARCHAR(250)  
);  
  
CREATE TABLE INVENTORY (  
    INVENTORYID VARCHAR(250) NOT NULL,  
    NAME VARCHAR(250),  
    HEADING VARCHAR(250),  
    DESCRIPTION VARCHAR(250),  
    PKGINFO VARCHAR(250),  
    IMAGE VARCHAR(250),  
    IMGBYTES BLOB(100000),  
    PRICE REAL NOT NULL,  
    COST REAL NOT NULL,  
    CATEGORY INTEGER NOT NULL,  
    QUANTITY INTEGER NOT NULL,  
    NOTES VARCHAR(250),  
    ISPUBLIC INTEGER NOT NULL,  
    MINTHRESHOLD INTEGER NOT NULL,  
    MAXTHRESHOLD INTEGER NOT NULL  
);  
  
...  
...  
ALTER TABLE CUSTOMER ADD CONSTRAINT PK_CUSTOMER PRIMARY KEY  
(CUSTOMERID);  
ALTER TABLE INVENTORY ADD CONSTRAINT PK_INVENTORY PRIMARY KEY  
(INVENTORYID);
```

14. We can now generate the EJB deployment code necessary to support our DB2 database. Right-click the deployment descriptor for the PlantsByWebSphere EJB module and select **Deploy** to generate the EJB deployment code.

If you get any errors in the Problems view, you can try a Clean operation, as this sometimes eliminates them.

Change the back-end ID

Because we have created a new database back-end map, we can set the default back-end map for the EJB to the newly created DB2 map. To map to the new DB2 map, do the following:

1. Open the deployment descriptor for the EJB module, scroll down to the bottom of the Overview tab and select **DB2UDBNT_V82_1** as the Current Backend ID, as shown in Figure 13-14.

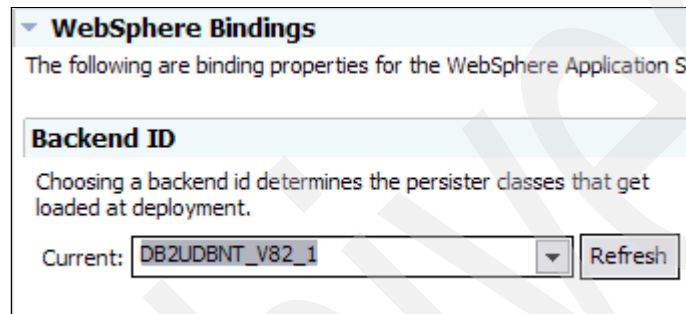


Figure 13-14 Setting default backend id for EAR file

2. Press Ctrl-S to save the deployment descriptor.

13.3.4 Setting the context root for Web modules

The context root is the part of the URL that comes after the protocol and address but before the path to servlets, JSPs, and so on. In a URL such as <http://www.plantsbywebsphere.com/PlantsByWebSphere/servlet/ShoppingServlet>, the context root is **PlantsByWebSphere**. It is used to separate Web modules from each other so that the Web container can dispatch Web requests to the right Web module.

When we created the **PlantsByWebSphereWeb** module, the Application Server Toolkit assigned it a context root with the same path, **PlantsByWebSphereWeb**. Because the sample application shipped and installed with WebSphere (if you install the samples) uses a context root of **PlantsByWebSphere**, we will change the context root also for our module to match.

1. Expand the Dynamic Web Projects section. Right-click the **PlantsByWebSphereWeb** module and select **Properties**.
2. Select the J2EE tab.

3. Change the context root to PlantsByWebSphere, as shown in Figure 13-15.

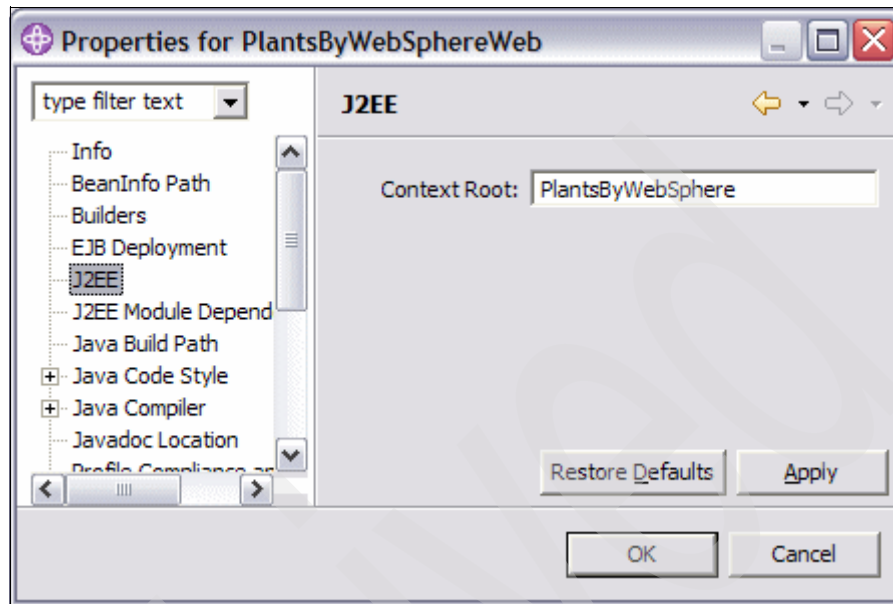


Figure 13-15 Setting context root for Web module

Click **OK**.

13.4 IBM EJB extensions: EJB caching options

This section discusses the caching options for entity and stateful session beans.

13.4.1 EJB container caching option for entity beans

The Enterprise JavaBeans specification defines three EJB caching options: options A, B, or C. Those options define how the EJB container handles entity bean instances between transactions. EJB caching options are set at the bean level, and are part of the IBM extensions deployment descriptor.

Caching option A

With caching option A, you assume that the entity bean has *exclusive* access to the underlying persistent store. In other words, between transactions, no one will modify the data. This includes a batch program updating the data, a Java application updating the data, or even the same entity bean running in a different container. This implies option A cannot be used in a clustered environment

(WLM). Note that it is your responsibility to ensure no other application will modify the data, as the EJB container has no way to control write access to the underlying database from other servers.

When caching option A is used, the entity bean instance is kept in a memory cache across transactions. At transaction commit, the entity bean attributes are synchronized with the underlying persistent store, and the bean instance remains cached in memory.

If you were tracing the calls made by the container, you would see something similar to Example 13-2. The first time the entity bean is used, its run time context is set (step 1), a bean is taken from the entity beans instance pool (step 2), the bean instance attributes are synchronized with the underlying data store (step 3), the method `setBalance` is invoked on the bean (step 4), and, finally, the bean attributes are saved back to the database (step 5). The bean is not returned to the pool. On subsequent calls, the `setBalance` method is invoked directly on the cached bean instance, and the bean attributes are synchronized with the underlying persistent data store.

Example 13-2 Entity beans call trace with option A caching

Transaction 1 (Begin)

Step 1: 1c9585f1 BranchAccount E called `setEntityContext()` method

Step 2: 1c9585f1 BranchAccount E called `ejbActivate()` method

Step 3: 1c9585f1 BranchAccount E called `ejbLoad()` method

Step 4: 1c9585f1 BranchAccount E called `setBalance()` method

Step 5: 1c9585f1 BranchAccount E called `ejbStore()` method

Transaction 1 (Commit)**Transaction 2 (Begin)**

Step 1: 284485f1 BranchAccount E called `setBalance()` method

Step 2: 284485f1 BranchAccount E called `ejbStore()` method

Transaction 2 (Commit)

Using caching option A can provide some performance enhancements at the expense of higher memory usage. You should only use it if you do not intend to use WebSphere clustering capabilities and you mostly access data in read mode.

Caching option B

With caching option B, you assume that you have *shared* access to the underlying database. This means the data could be changed by another application between transactions. When option B is used, the bean instance attributes are always synchronized with the underlying back-end data store at the beginning of every transaction. Similar to Option A, the bean is kept in the cache

between transactions. Therefore, if you were tracing the different calls made in Option B, you would obtain the trace shown in Example 13-3.

Example 13-3 Entity beans call trace with option B caching

Transaction 1 (Begin)

Step 1: 1c9585f1 BranchAccount E called setEntityContext() method

Step 2: 1c9585f1 BranchAccount E called ejbActivate() method

Step 3: 1c9585f1 BranchAccount E called ejbLoad() method

Step 4: 1c9585f1 BranchAccount E called setBalance() method

Step 5: 1c9585f1 BranchAccount E called ejbStore() method

Transaction 1 (Commit)

Transaction 2 (Begin)

Step 1: 284485f1 BranchAccount E called ejbLoad() method

Step 2: 284485f1 BranchAccount E called setBalance() method

Step 3: 284485f1 BranchAccount E called ejbStore() method

Transaction 2(Commit)

Caching option B can be safely used in a clustered environment, or when you are not sure if you have exclusive access to data. You are assured that you always work with the last committed data. Option B memory usage is the same as for option A. The performance of both options can slightly differ depending on the nature of your application.

Caching option C

Similar to option B, caching option C assumes *shared* access to the database. Unlike option B or A, the bean instance is returned to the entity beans pool at the end of the transaction. A new bean instance is used at the beginning of every transaction. Each transaction results in the sequence of calls shown in Example 13-4.

Example 13-4 Entity beans call trace with option C caching

Transaction (Begin)

Step 1: 1c9585f1 BranchAccount E called setEntityContext() method

Step 2: 1c9585f1 BranchAccount E called ejbActivate() method

Step 3: 1c9585f1 BranchAccount E called ejbLoad() method

Step 4: 1c9585f1 BranchAccount E called setBalance() method

Step 5: 1c9585f1 BranchAccount E called ejbStore() method

Step 6: 1c9585f1 BranchAccount E called ejbPassivate() method

Step 7: 1c9585f1 BranchAccount E called unsetEntityContext() method

Transaction (Commit)

Caching option C has the best memory usage at the expense of a larger number of methods calls. This is the default behavior.

How to set the EJB caching option

You must combine the Activate at and Load at options to set the EJB caching option to A, B, or C. Use Table 13-3 to choose the right combination.

Table 13-3 Setting entity EJB caching properties

Option	Activate at must be set to	Load at must be set to
Option A	Once	Activation
Option B	Once	Transaction
Option C (default)	Transaction	Transaction

To set the EJB caching option, do the following:

1. Open the EJB deployment descriptor.
2. Switch to the **Bean** tab.
3. Select the entity bean in the window to the left, then scroll down the options at right until you see the Bean Cache settings under the WebSphere extensions section, as in Figure 13-16.

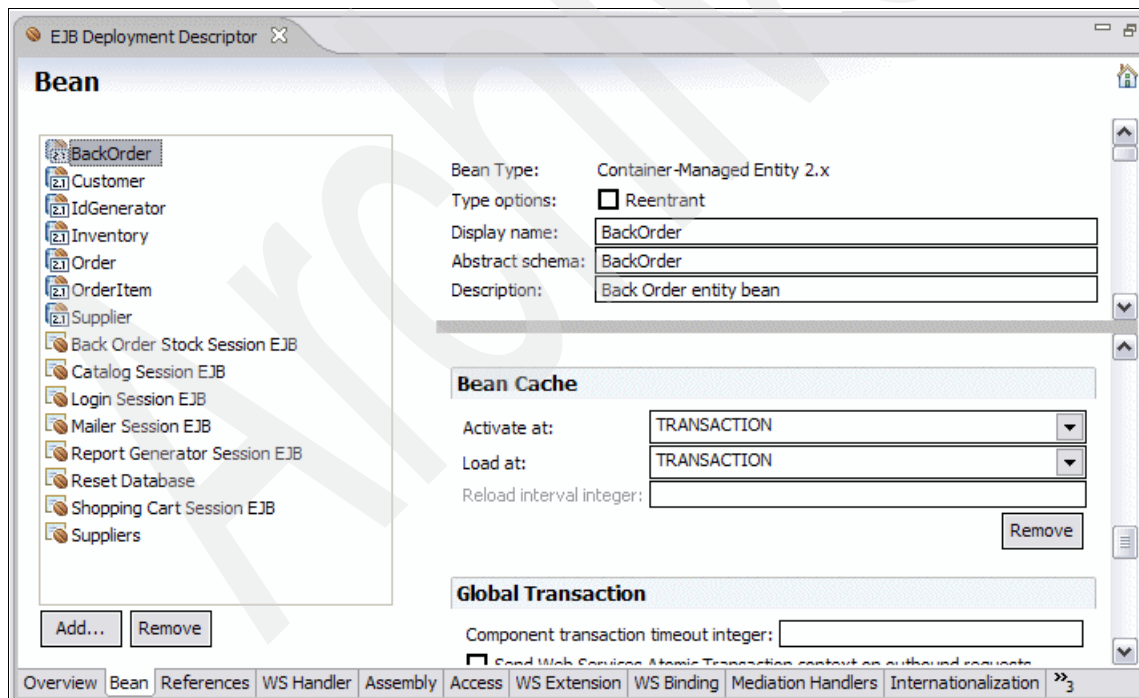


Figure 13-16 Setting the activate and load settings for entity beans

4. Select the **Activate at** and **Load at** options according to Table 13-3 on page 857.
5. Close and save the deployment descriptor.

The settings are saved in the `ibm-ejb-jar-ext.xml` file. They correspond to the following line:

```
<beanCache xmi:id="BeanCache_1" activateAt="ONCE" loadAt="TRANSACTION"/>
```

There is one line for each entity bean for which you have set this option.

In addition to the two standard options for EJB activation and data loading, WebSphere also supports some additional options, such as activate at activity session and to load data on a daily, weekly, or other interval basis. For more information about these WebSphere extensions, refer to the WebSphere IntoCenter (search for bean cache settings).

Note: Although the Application Server Toolkit and Rational Application Developer allow you to set the Activate at and Load at options also for stateless session EJBs, they have no effect.

13.4.2 EJB container caching option for stateful session beans

Similarly to entity beans, you can specify which caching strategy to use for stateful session beans. This caching option specifies the point at which an enterprise bean is activated and placed in the cache. Removal from the cache and passivation are also governed by this setting. Valid values are:

- ▶ Once (default)
Choosing Once indicates that the bean is activated when it is first accessed in the server process. It is passivated and removed from the cache at the discretion of the container, for example, when the cache becomes full.
- ▶ Transaction
Choosing Transaction indicates that the bean is activated at the start of a transaction. It is passivated and removed from the cache at the end of the transaction.

You can set this caching option by opening the EJB deployment descriptor for the EJB module. The Activate at setting is found on the Bean tab (Figure 13-17). Select the bean and scroll down to the Bean Cache category.

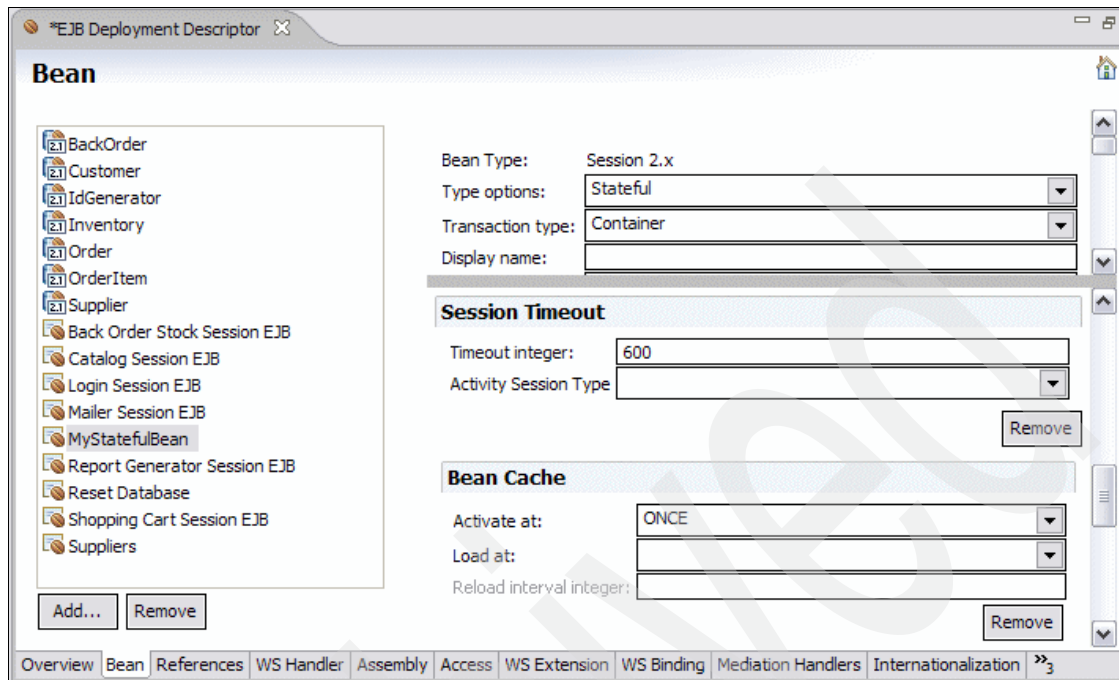


Figure 13-17 Activate settings for stateful session beans

Note: The Load at option does not have any effect on Stateful Session EJBs.

13.4.3 Stateful EJB timeout option

Additionally, you can specify a timeout value for stateful session beans. A bean can time out in the `METHOD_READY` or in the `PASSIVATED` state. If you try to access a bean that has timed out, you see an exception similar to that in Example 13-5.

Example 13-5 Stateful EJB timed out exception

```
com.ibm.ejs.container.SessionBeanTimeoutException: Stateful bean
StatefulBean0(BeanId(Webbank#webbankEJBs.jar#Transfer, ebf64d846a), state =
METHOD_READY) timed out.
```

Session beans that have timed out can be removed by the container, for example, if it needs to free memory. However, a well-written application should not rely on beans to time out to free memory. Instead, it is important that the

developer explicitly calls `remove()` on a bean when this stateful bean is not needed anymore.

The default timeout is 600 seconds. You can set the timeout integer value as a parameter of a stateful session bean by opening the EJB deployment descriptor and selecting the **Bean** tab, as in Figure 13-17 on page 859. By specifying a value of 0, you set the bean to never expire.

Setting this timeout inserts the following property in the `ejbExtensions` tag of the IBM bindings file:

```
<ejbExtensions xmi:type="ejbext:SessionExtension"
  xmi:id="Session_1_Ext" timeout="120">
```

Note: If a bean times out in the `METHOD_READY` state and is consequently removed by the container, the `ejbRemove()` method is called on the bean instance. If a bean times out in the passivated state, `ejbRemove()` is not called, according to the EJB specification.

13.5 IBM EJB extensions: EJB access intents

Access intents are used to optimize the access to relational data. Access intents are only applicable to EJB 2.x beans. For EJB 1.1 beans, you still use the old method of setting the transaction isolation level at the method level, as well as mark methods as read-only. In this section, we only cover access intents for EJB 2.x beans.

Access intent policies are specifically designed to supplant the use of isolation level and read-only, method-level modifiers found in the extended deployment descriptor for EJB Version 1.1 enterprise beans. You cannot specify isolation level and read-only modifiers for EJB Version 2.x enterprise beans. The WebSphere persistence manager uses access intent hints to make decisions about isolation level, cursor management, and more.

13.5.1 Transaction isolation levels overview

Transaction isolation levels provide a trade-off between accuracy of reads versus concurrent readers. The levels can best be described by the types of read anomalies they permit and forbid. Consider the read anomalies that can occur with two concurrent transactions, T1 and T2:

- Dirty read

T1 reads data that has been modified by T2, before T2 commits.

► Non-repeatable read

Non-repeatable read is caused by fine-grained locks.

- T1 reads a record and drops its lock.
- T2 updates.
- T1 re-reads different data.

► Phantom read

This is a non-repeatable read involving a range of data and inserts or deletes on the range.

- T1 reads a set of records that match some criterion.
- T2 inserts a record that matches the criterion.
- T1 continues processing the set, which now includes records that were not part of the original matching set.

There are four possible settings for the transaction isolation level:

► Repeatable read (TRANSACTION_REPEATABLE_READ)

This setting permits phantom reads and forbids both dirty and unrepeatable reads.

► Read committed (TRANSACTION_READ_COMMITTED)

This setting permits non-repeatable and phantom reads and forbids dirty reads.

► Read uncommitted (TRANSACTION_READ_UNCOMMITTED)

This setting permits all the read anomalies, including dirty reads, non-repeatable reads, and phantom reads.

► Serializable (TRANSACTION_SERIALIZABLE)

This setting forbids all the read anomalies.

The container applies the isolation level as follows:

- For entity beans with container managed persistence (CMP), the container generates code that ensures the desired level of isolation for each database access.
- For session beans and bean-managed persistence (BMP) entity beans, the container sets the isolation level at the start of each transaction, for each database connection.

The transaction isolation level is tied to a database connection. The connection uses the isolation level specified in the first bean that uses the connection. The

container throws an `IsolationLevelChangeException` whenever the connection is used by another bean method that has a different isolation level.

Not all databases support all JDBC isolation levels. Moreover, JDBC definitions for isolation levels might not match the database definition of isolation levels. As an example, DB2 definitions for isolation levels follow the naming conventions used in *Transaction Processing: Concepts and Techniques* by Gray. Table 13-4 shows a mapping between EJB and DB2 isolation levels.

Table 13-4 Mapping JDBC isolation levels to DB2 isolation levels

JDBC isolation level	DB2 isolation level
TRANSACTION_SERIALIZABLE	Repeatable Read
TRANSACTION_REPEATABLE_READ	Read Stability
TRANSACTION_READ_COMMITTED	Cursor Stability
TRANSACTION_READ_UNCOMMITTED	Uncommitted Read

To learn more, refer to the documentation provided by your database product.

13.5.2 Concurrency control

Concurrency control is the management of contention for data resources. A concurrency control scheme is considered *pessimistic* when it locks a given resource early in the data-access transaction and does not release it until the transaction is closed. A concurrency control scheme is considered *optimistic* when locks are acquired and released over a very short period of time at the end of a transaction.

The objective of optimistic concurrency is to minimize the time over which given resource is unavailable for use by other transactions. This is especially important with long-running transactions, which under a pessimistic scheme would lock up a resource for unacceptably long periods of time.

WebSphere uses *an overqualified update scheme* to test whether the underlying data source has been updated by another transaction since the beginning of the current transaction. With this scheme, the columns marked for update and their original values are added explicitly through a WHERE clause in the UPDATE statement so that the statement fails if the underlying column values have been changed. As a result, this scheme can provide column-level concurrency control; pessimistic schemes can control concurrency at the row level only.

Optimistic schemes typically perform this type of test only at the end of a transaction. If the underlying columns have not been updated since the

beginning of the transaction, pending updates to container-managed persistence fields are committed and the locks are released. If locks cannot be acquired or if some other transaction has updated the columns since the beginning of the current transaction, the transaction is rolled back and all work performed within the transaction is lost.

Pessimistic and optimistic concurrency schemes require different transaction isolation levels. Enterprise beans that participate in the same transaction and require different concurrency control schemes cannot operate on the same underlying data connection unless the connection is able to change its isolation level on an individual-query basis. Some, but not all, JDBC drivers can do this. For those JDBC drivers that cannot, mixing concurrency controls requires the use of multiple connections within a transaction.

Whether or not to use optimistic concurrency depends on the type of transaction. Transactions with a high penalty for failure might be better managed with a pessimistic scheme. For low-penalty transactions, it is often worth the risk of failure to gain efficiency through the use of an optimistic scheme.

13.5.3 Using EJB 2.x access intents

Access intents policies let you define, in a very flexible and powerful way, how relational data will be accessed if you use BMP or CMP entity beans.

Important: Although you can set access intents on a BMP, you are responsible for reading the access intent metadata from your code and applying the corresponding change to the isolation levels yourself by calling `connection.setTransactionLevel()`. The EJB container has no control over your persistence strategy, and therefore cannot perform the same tasks as it can for CMPs. Access intent data is valid in the WebSphere naming service for the time of the transaction. See the Information Center for more coding examples.

Access intents policies

Seven access intent policies are available. They cover a wide variety of ways to access data. They are summarized in Table 13-5 on page 864.

Table 13-5 Access intent policies

Access Intent Policy	Concurrency control	Used for update	Transaction isolation level	Notes
wsPessimisticRead	Pessimistic	No	For Oracle, read committed. Otherwise, repeatable read.	Read locks are held for the duration of the transaction. Updates are not permitted; the generated SELECT query does not include FOR UPDATE.
wsPessimisticUpdate	Pessimistic	Yes	For Oracle, read committed. Otherwise, repeatable read.	The generated SELECT FOR UPDATE query grabs locks at the beginning of the transaction.
wsPessimisticUpdate-Exclusive	Pessimistic	Yes	Serializable.	SELECT FOR UPDATE is generated; locks are held for the duration of the transaction.
wsPessimisticUpdate-NoCollision	Pessimistic	Yes	Read committed.	The generated SELECT query does not include FOR UPDATE. <i>No locks are held, but updates are permitted.</i>
wsPessimisticUpdate-WeakestLockAtLoad (DEFAULT VALUE)	Pessimistic	Yes	For Oracle, read committed, Otherwise, repeatable read.	For Oracle, this is the same as wsPessimisticUpdate. Otherwise, the generated SELECT query does not include FOR UPDATE; locks are escalated by the persistent store at storage time if updates were made.
wsOptimisticRead	Optimistic	No	Read committed.	

Access Intent Policy	Concurrency control	Used for update	Transaction isolation level	Notes
wsOptimisticUpdate	Optimistic	Yes.	Read committed.	Generated overqualified-update query forces failure if CMP column values have changed since the beginning of the transaction.

There are two critical questions to ask yourself when using access intents:

- ▶ At which point in my transaction do I access data? This is critical in selecting on which method you must set the access intent.
- ▶ How do I want to access data? This is critical to selecting the best access intent to apply on the method.

Choosing where to apply the access intent

This is critical because it is at this point that the WebSphere persistence manager decides which access intent to use. To illustrate this point, we will use the following example.

Assume a session EJB called Consultation obtains the CustomerAccount balance using `getBranchAccountBalance`, as in Example 13-6.

Example 13-6 Obtaining CustomerAccount balance using getBranchAccountBalance

```
int getCustomerAccountBalance () {
    ...
    custAcct = (CustomerAccountLocal) custAcctHome.findByPrimaryKey(custAcctKey);
    return custAcct.getBranchBalance();
}
```

Imagine you have applied `AccessWriteIntent1` on the `findByPrimaryKey()` method of the `CustomerAccount` bean, and `AccessReadIntent2` on the `getBranchBalance()` method. Because the first access to the database in the transaction started by the call to `getCustomerAccountBalance()` is done by the `findByPrimaryKey` method, then `AccessWriteIntent1` is used for all calls within the transaction. `AccessReadIntent2` will be ignored by the persistence manager and is therefore useless in this case.

This might be satisfactory or not, depending on what you want to achieve. The critical point here is that you can use the `findByPrimaryKey` method in read and update transactions. If you use it in an update transaction, you probably want to

execute it with, for example, a `PessimisticUpdate` intent. If you access data only for reading it, this would be more than you need.

There are two main solutions you can adopt for this problem. The simplest one is to have two or more versions of your finder methods, specialized by access intent. You could use the standard `findByPrimaryKey` in update scenarios and add another finder method, such as `findByPrimaryKeyForRead`, and use it in read-only scenarios. You would set the access intent of the `findByPrimaryKeyForRead` finder, say to `wsOptimisticRead`. The default access intent (`wsPessimisticUpdate-WeakestLockAtLoad`) is fine in most cases for the `findByPrimaryKey()` method, as well as for other finder or non-finder methods.

Important: This solution is also well adapted to BMPs. By having a different `findByPrimaryKey` method for read and write transactions, you can easily set a different isolation level in the code for each of them. You can also define a different `SELECT` query, one with a `FOR UPDATE` clause and one without, then call them from those different methods.

Another solution is to run `findByPrimaryKey`, or another finder, in its own transaction. Do this by applying a `RequiresNew` transaction flag on it. Take another look at the previous sample:

1. The `getCustomerAccountBalance` method starts a new transaction.
2. `findByPrimaryKey` is called. The current transaction is paused. The `findByPrimaryKey` method executes within its own transaction and, therefore, own access intent. The call to `findByPrimaryKey()` returns an *unhydrated* instance, which means it has not been activated nor loaded.
3. The transaction initiated by the session bean resumes.
4. `getBalance()` is called on the instance returned by `findByPrimaryKey`. The instance is hydrated and the access intent specified for this method is used. Any other method calls within the transaction will execute with the same access intent.

Note: WebSphere Application Server V6.0 and V6.1 also includes a feature (inherited from WebSphere Application Server V5 Enterprise Edition) that provides an extension to access intents called Application Profiles, which handles the problem mentioned above in a powerful way. Application profiles let you externally specify a set of tasks (that is, a flow of calls in your code), and specify which access intent should be used for a specific task. For information about Application Profiles, please refer to the WebSphere Information Center.

Choosing the right access intent

The main rule is: keep it simple.

Start with the default setting (wsPessimisticUpdate-WeakestLockAtLoad), and work from there. Specifying access intents on all your business methods could lead to a configuration, debugging, and maintenance nightmare. Specify access intents on a selected number of methods. Also, choose access intents wisely.

- ▶ Access intents can be applied to your business methods, to the `findByPrimaryKey()` method, as well as the create and remove method. As much as possible, avoid other methods.
- ▶ Make sure that no method is configured with more than one access intent policy. Applications that are misconfigured in this way will not be runnable until the configuration errors are fixed.
- ▶ For entity beans backed by tables with nullable columns, use optimistic policies with caution. Nullable columns are automatically excluded from overqualified updates at deployment time. This means that at commit time, those columns will not be used in the update statement to check whether the data has changed or not. Therefore, concurrent changes to a nullable field might result in lost updates. Using the Application Server Toolkit, you can set a property on each enterprise bean attribute called *OptimisticPredicate*, as shown in Figure 13-18 on page 868. You can change this property by editing the data mappings of your EJBs. When this property is set, the column, even if it is nullable, will be reflected in the overqualified update statement that is generated in the deployment code to support optimistic policies.

WebSphere Application Server supports an optimistic concurrency control scheme for EJB 2.x CMP entity beans that allows you to add a column for collision detection in your relational database table. This column is reserved to determine if a record has been updated. When using a collision detection column, the overqualified UPDATE statement only needs the collision detection column and the primary key. To manage the collision detection column, provide your own database trigger implementation. Using the collision detection column overcomes the nullable column limitation and the unsupported optimistic concurrency control data types, such as BLOBs and CLOBs.

Tip: If you want to check which SQL code is executed for an optimistic update, check the `storeUsingOCC` method in the `<bean>FunctionSet` generated class.

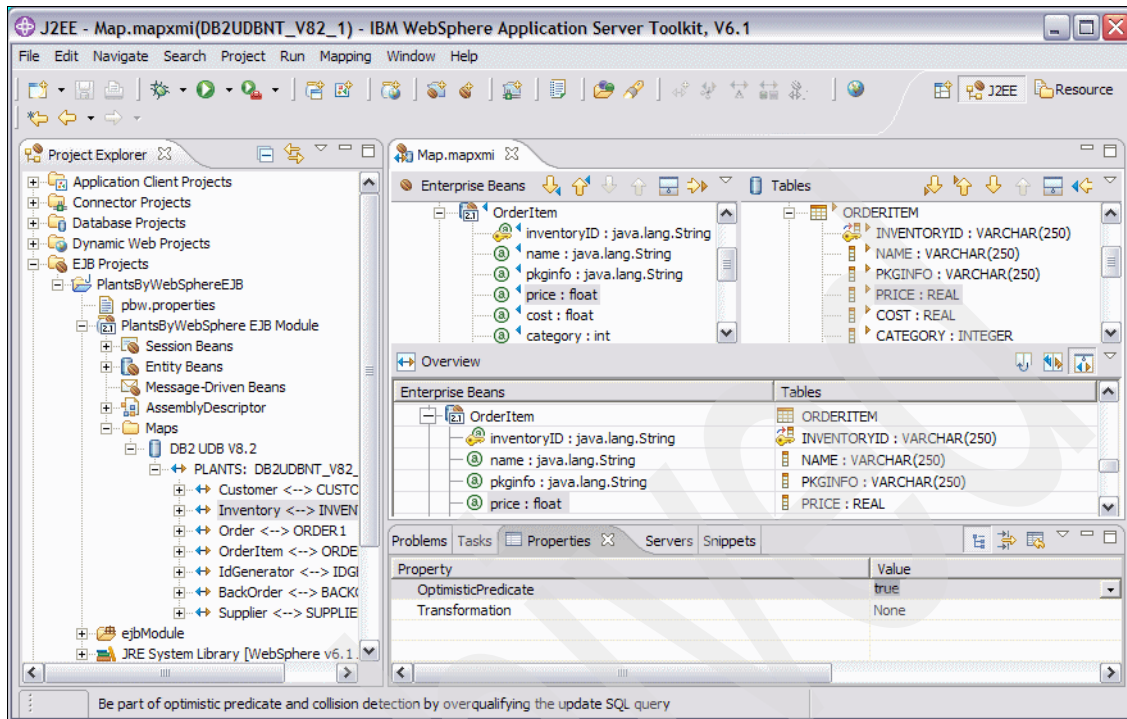


Figure 13-18 Optimistic predicate property

If a bean is loaded for read intent and an update is attempted during that transaction, then the persistence manager will raise an `UpdateCannotProceedWithIntegrity` exception. In other words, if you call `findByPrimaryKeyForRead()` and an update is attempted, it will fail.

Important: The behavior described above is true for all access intents except `wsPessimisticUpdate-NoCollision`. This access intent will not flag updates, even if no locks are held. Our recommendation is that you avoid this access intent in production.

13.5.4 Using read-ahead hints

Read-ahead schemes enable applications to minimize the number of database round trips by retrieving a working set of CMP beans for the transaction within one query. Read-ahead involves activating the requested CMP beans and caching the data for their related beans, which ensures that data is present for the beans that are most likely to be needed next by an application.

A read-ahead hint is a canonical representation of the related beans to be read. It is associated with a finder method for the requested bean type, which must be an EJB 2.x-compliant CMP entity bean. Currently, only `findByPrimaryKey` methods can have read-ahead hints. Only beans related to the requested beans by a container-managed relationship (CMR), either directly or indirectly through other beans, can be read ahead.

To set Read-ahead hints, do the following:

1. Open the EJB deployment descriptor editor.
2. Select the **Access** tab and scroll down to the WebSphere Extensions section.
3. Click the **Add** button to the right of the Access Intent for Entities 2.x (Method Level) field. See Figure 13-19.

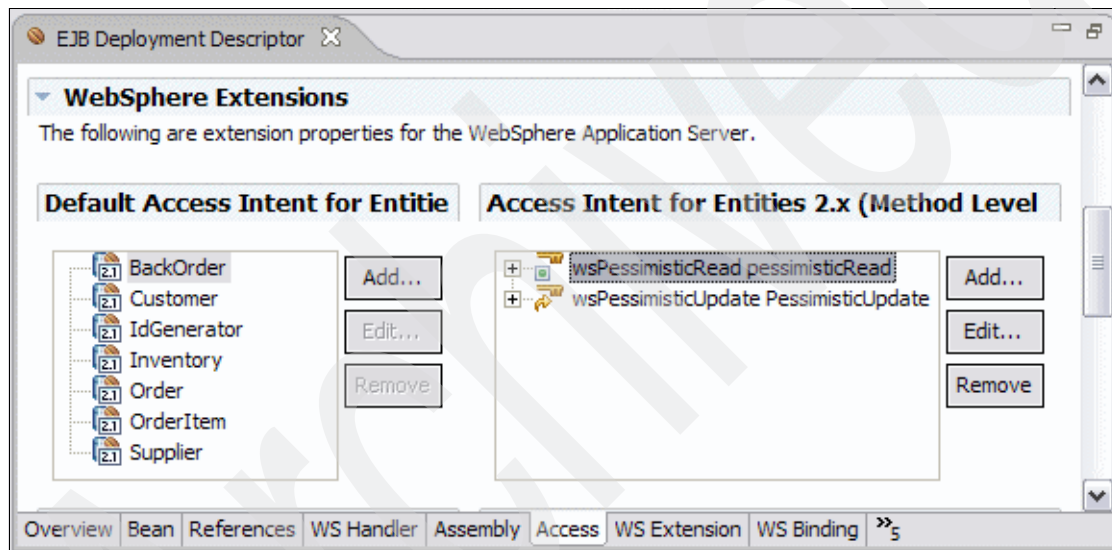


Figure 13-19 Adding a read ahead hint

See Figure 13-20 on page 870.

Edit 2.x Access Intent

Access Intent
Enter name and description.

Name:

Access intent name:

A value of read indicates the caller does not intend to drive an update method on the entity bean. This policy locks tables or rows being read in the SQL statement.

Description:

☒ Read ahead hint

Persistence option

☐ Verify read only data

☐ Deferred operation

☐ Batch

< Back **Next >** Finish Cancel

Figure 13-20 Specifying read-ahead hint

- Follow the windows in the wizard to select the beans and methods, then click **Finish**.

Note: While using read-ahead hints can improve performance by minimizing the number of database round trips required to retrieve the data for the related beans, overdoing it can have an adverse effect. For example, using read-ahead hints on long or complex paths can result in a query that is too complex to be useful.

13.5.5 Tracing access intents behavior

You can obtain a very detailed trace of the EJB persistence manager behavior by specifying the following trace specification for an application server:

```
com.ibm.ejs.container.*=all=enabled:com.ibm.ejs.persistence.*=all=enabled:  
com.ibm.ws.appprofile.*=all=enabled.
```

13.6 IBM EJB extensions: inheritance relationships

Support for entity beans inheritance, which is not part of the EJB 1.1 nor EJB 2.x specifications, is also available in the toolkit. Support for enterprise entity beans relationships for EJB 1.1, although not standard, is also available using this tool. Refer to the toolkit documentation for more details.

13.7 IBM Web module extensions

WebSphere Application Server also provides multiple extensions for Web modules. To work with these extensions, open the Web deployment descriptor by double-clicking the Web module in the J2EE Hierarchy view. To see the IBM Web module extensions, select the **Extensions** tab, as in Figure 13-21.

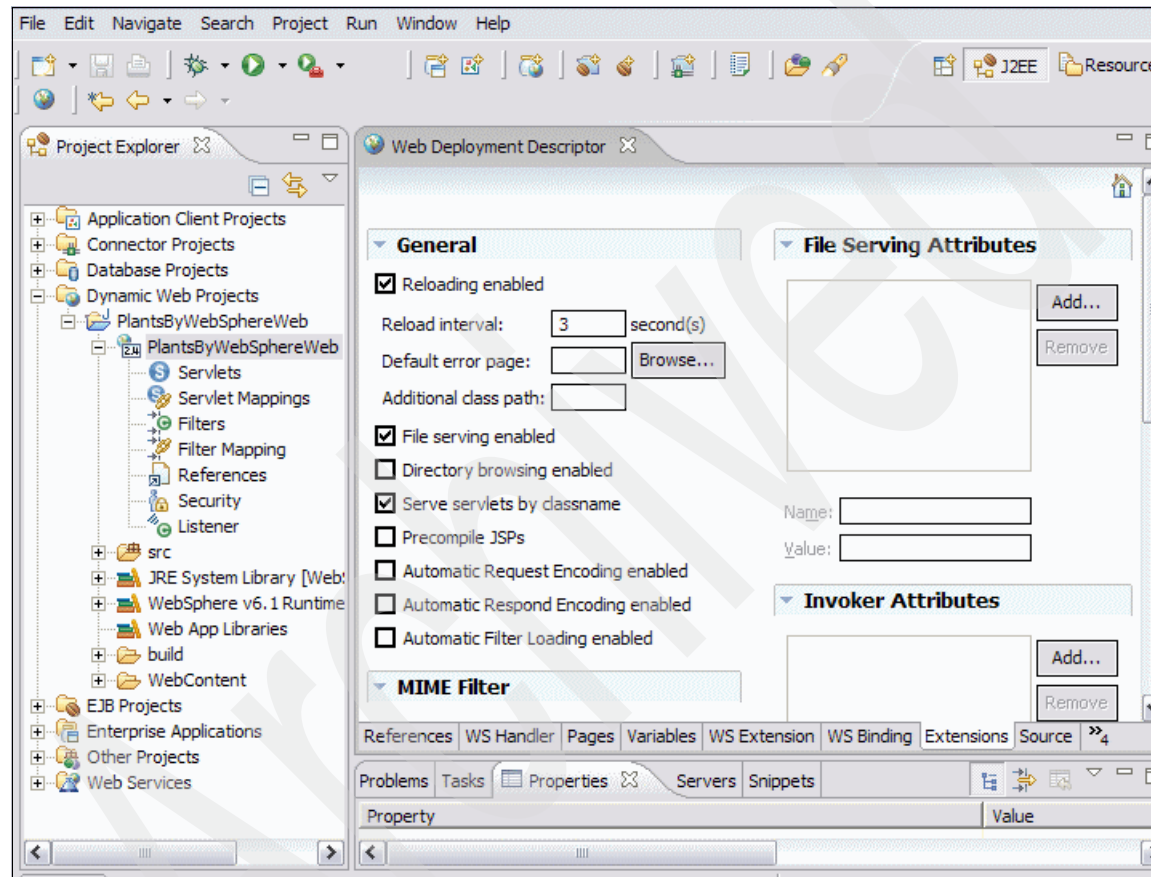


Figure 13-21 Web module extensions

13.7.1 File serving servlet

When dealing with static content (HTML pages, images, style sheets, and so on), you can choose to have these resources served by WebSphere, or have them served by the HTTP server itself.

If you want WebSphere to serve the static content of your application, you must enable file servlet, also known as the file serving servlet or file serving enabler.

This servlet serves up any resource file packaged in the WAR file. The File serving enabled attribute is set to true by default. By changing it to false, the Web server plug-in will not send requests for static content to WebSphere, but leave it up to the HTTP server to serve them.

If you want the Web server to serve static content, you can experience better performance than using WebSphere in this instance, because the Web server is serving the content directly. Moreover, a Web server has more customization options than the file servlet can offer.

However, using the WebSphere file serving servlet has the advantage of keeping the static content organized in a single, deployable unit with the rest of the application. Additionally, this allows you to protect the static pages using WebSphere security.

To enable this option, check the **File serving enabled** box. Enter attributes used by the file serving servlet in the File Serving Attributes section.

13.7.2 Web application auto reload

If you check the **Reloading enabled** option, the class path of the Web application is monitored and all components, JAR or class files, are reloaded whenever a component update is detected. The Web module's class loader is shut down and restarted. The reload interval is the interval between reloads of the Web application. It is set in seconds.

The auto reload feature plays a critical role in hot deployment and dynamic reload of your application.

Important: You must set the Reloading enabled option to true for JSP files to be reloaded when they are changed on the file system. Reloading a JSP does not trigger the reload of the Web module, because separate class loaders are used for servlets and JSP.

This option is set to true by default, with the reload interval set to three (3) seconds. In production mode, you might consider making the reload interval much higher.

13.7.3 Serve servlets by class name

The invoker servlet can be used to invoke servlets by class name. Note that there is a potential security risk with leaving this option set in production. It should be seen as more of a development-time feature, for quickly testing your servlets.

A better alternative than this option is to define servlet mappings in the Web deployment descriptor for the servlets that should be available.

This option is turned off by default.

13.7.4 Default error page

This page will be invoked to handle errors if no error page has been defined, or if none of the defined error pages matches the current error.

13.7.5 Directory browsing

This Boolean defines whether it is possible to browse the directory if no default page has been found.

This option is turned off by default.

13.7.6 JSP attributes

The JSP engines provides several options that can be set to customize its behavior:

- keepgenerated

If this Boolean is set to true, the source code of the servlet created by compilation of a JSP page is kept on the file system. Otherwise, it is deleted as soon as the servlet code has been compiled; only the .class file is available.

- scratchdir

This string represents the directory in which the generated class files will be generated. If this string is not set, code is created under:

`<profile_root>\temp`

- jdkSourceLevel

This setting specifies which JDK level JSPs will be generated and compiled for. Valid values are 13 (default), 14, and 15. To compile for Java 5.0, use 15.

There are several more options that can be set. Search for “JSP engine configuration parameters” in the WebSphere InfoCenter for the full list.

13.7.7 Automatic HTTP request and response encoding

The Web container no longer automatically sets request and response encodings and response content types. The programmer is expected to set these values using the methods available in the Servlet 2.4 API. If you want the application server to attempt to set these values automatically, check the **Automatic Request Encoding enabled** option in order to have the request encoding value set. Similarly, you can check the **Automatic Response Encoding enabled** option in order to have the response encoding and content type set.

The default value of the `autoRequestEncoding` and `autoResponseEncoding` extensions is false, which means that both the request and response character encoding is set to the Servlet 2.4 specification default of ISO-8859-1. Different character encodings are possible if the client defines character encoding in the request header, or if the code uses the `setCharacterEncoding(String encoding)` method.

If the `autoRequestEncoding` value is set to true, and the client did not specify character encoding in the request header, and the code does not include the `setCharacterEncoding(String encoding)` method, the Web container tries to determine the correct character encoding for the request parameters and data.

The Web container performs each step in the following list until a match is found:

- ▶ Looks at the character set (charset) in the Content-Type header.
- ▶ Attempts to map the server's locale to a character set using defined properties.
- ▶ Attempts to use the `DEFAULT_CLIENT_ENCODING` system property, if one is set.
- ▶ Uses the ISO-8859-1 character encoding as the default.

If you set the `autoResponseEncoding` value to true and the client:

- ▶ The client did not specify character encoding in the request header.
- ▶ The code does not include the `setCharacterEncoding(String encoding)` method,

The Web container does the following:

- ▶ Attempts to determine the response content type and character encoding from information in the request header.
- ▶ Uses the ISO-8859-1 character encoding as the default.

13.8 IBM EAR extensions: Sharing session context

In accordance with the servlet 2.4 API specification, the session manager supports session scoping by Web module only. Only servlets in the same Web module can access the data associated with a particular session. WebSphere Application Server provides an option that you can use to extend the scope of the session attributes to an enterprise application. Therefore, you can share session attributes across all the Web modules in an enterprise application.

This option can be set in the toolkit in the enterprise application deployment descriptor.

1. Open the deployment descriptor by double-clicking the enterprise application.
2. Check the **Shared session context** option in the WebSphere Extensions section. See Figure 13-22 on page 876.

The screenshot shows the 'WebSphere EAR deployment descriptor' configuration window. The 'WebSphere Extensions' section is expanded, showing the 'Shared session context' checkbox checked. Other sections include 'General Information' (Display name: PlantsByWebSphere, Description:), 'Security Roles' (Details...), 'Modules' (Web PlantsByWebSphereWeb.war, EJB PlantsByWebSphereEJB.jar), and 'Icons' (Small: Browse..., Large: Browse...). The 'Reload interval' is set to 0 seconds.

Figure 13-22 EAR deployment descriptor

Important: To use this option, you must install all the Web modules in the enterprise application in the same application server. You cannot use this option when one Web module is installed in one server and the second Web module is installed in a different server.

In such split installations, applications might share session attributes across Web modules using distributed sessions. However, session data integrity is compromised when concurrent access to a session is made in different Web modules.

Sharing HTTP session context also severely restricts the use of some session management features, like time-based writes. For enterprise applications on which this option is enabled, the session management configuration set at the Web module level is ignored. Instead, the session management configuration defined at the enterprise application level is used.

13.9 Exporting the PlantsByWebSphere EAR file

Once you have made all the changes to your application and are ready to deploy, you need to export the EAR file to a location where it can be picked up for deployment by the application server.

To export the Plants by WebSphere sample application, do the following:

1. Select **File** → **Export**.
2. Select **EAR file** as the export target and click **Next**.
3. Select to export the **PlantsByWebSphere** EAR project and enter a suitable destination for the EAR file, such as C:\PlantsByWebSphere.ear. Then click **Finish**.

13.10 WebSphere Enhanced EAR

The Enhanced EAR, introduced in WebSphere Application Server V6.0, is a normal J2EE EAR file, but with additional configuration information for resources required by J2EE applications. While adding this extra configuration information at packaging time is not mandatory, it can simplify deployment of J2EE applications to WebSphere if the environments where the application is to be deployed is similar.

Table 13-6 shows the resources supported by the Enhanced EAR and the scope in which they are created.

Table 13-6 Scope for resources in WebSphere Enhanced EAR file

Resource	Scope
JDBC providers	Application
Data sources	Application
Resource adapters	Application
JMS resources	Application
Substitution variables	Application
Class loader policies	Application
Shared libraries	Server
JAAS authentication aliases	Cell
Virtual hosts	Cell

New in V6.1: The Enhanced EAR has been improved with V6.1 to include support for J2C Resource Adapters (RAR files) and JMS resources.

J2C Resource Adapters can be configured either as embedded or external resources. An embedded RAR is packaged within an enterprise application (EAR), deployed as a part of J2EE application installation, and is removed when the application is uninstalled from the server. An external RAR is packaged as a stand-alone RAR file, is deployed explicitly on a WebSphere node, and is not managed as a J2EE application. If an adapter is used only by a single application, it should be configured as an embedded RAR. If it is to be shared between multiple applications, it should be an external RAR.

When an Enhanced EAR is deployed to WebSphere Application Server, WebSphere can automatically configure the resources specified in the Enhanced EAR. This reduces the number of configuration steps required to set up the WebSphere environment to host the application.

When an Enhanced EAR is uninstalled, the resources that are defined at the application level scope are removed as well. However, resources defined at a scope other than application level are not removed because they might be in use by other applications.

Resources created at Application level scope are limited in visibility to only that application.

New in V6.1: In WebSphere Application Server V6.0, the administrative console did not allow you to view or change Application-scope settings. In V6.1, you can now do this. However, you cannot create new resources at the application scope using the administrative console. To add new application-scoped resources you must use the Application Server Toolkit or Rational Application Developer.

To view the application scoped resources, select **Applications** → **Enterprise Applications** → **<application>**. Select **Application scoped resources** in the References section. If there are no application scoped resources, you will not see this option.

13.10.1 Configuring a WebSphere Enhanced EAR

The supplemental information in an Enhanced EAR is modified by using the WebSphere Enhanced EAR editor, the Deployment tab of the application deployment descriptor in the Application Server Toolkit.

Note: Before adding or removing J2EE modules using the Module page in the Application Deployment Descriptor editor, do the following:

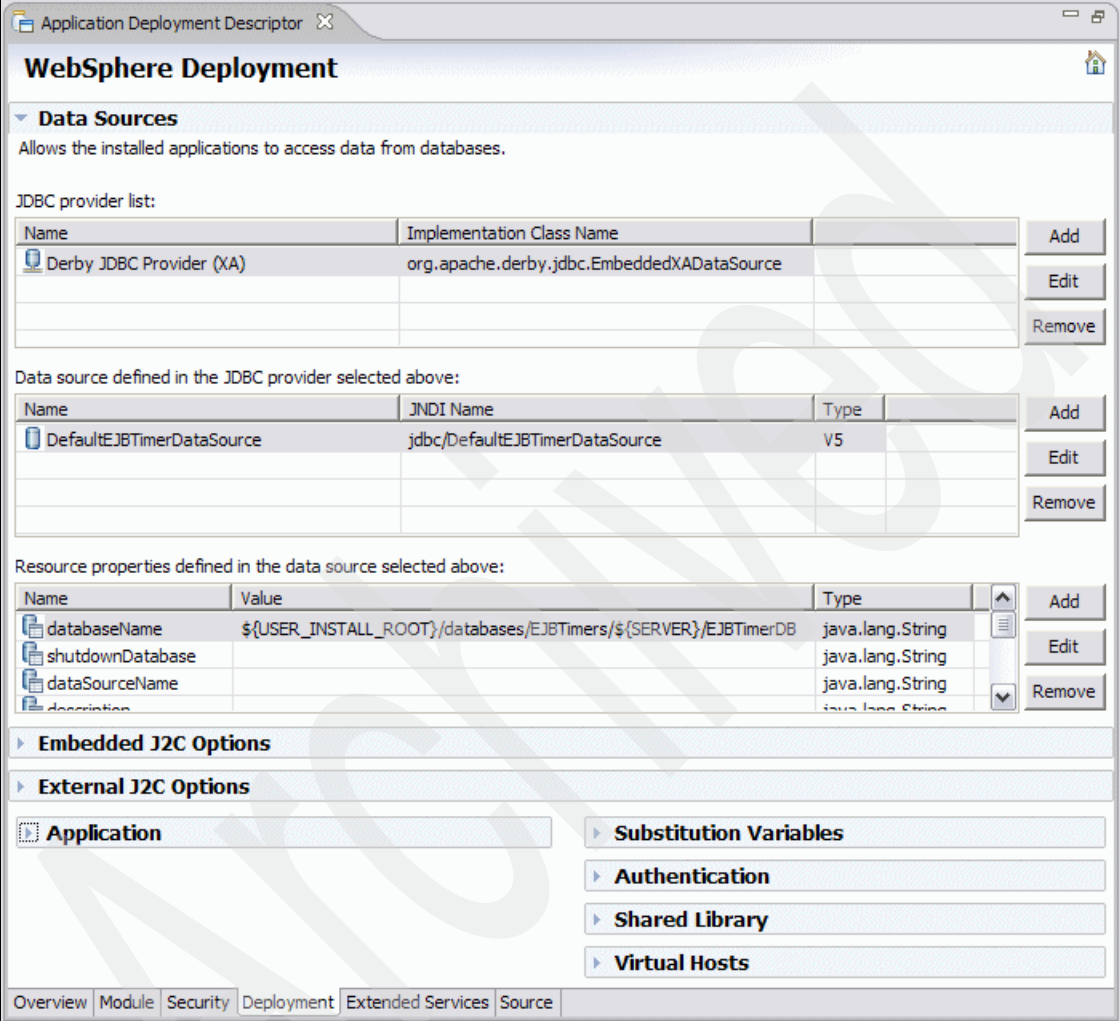
1. Click the **Deployment** tab to activate the functions in the deployment page.
2. Add your modules to the Module page.

Complete this task for each application deployment descriptor editor session that you want to add or remove modules from the Module page.

To access the Enhanced EAR deployment options, do the following:

1. In the J2EE Project Explorer view, expand **Enterprise Applications**, and then the application.

2. Double-click **Deployment Descriptor** and select the **Deployment** tab. This opens up the Enhanced EAR editor, as shown in Figure 13-23.



The screenshot shows the 'WebSphere Deployment' window with the 'Deployment' tab selected. It contains three main sections: 'Data Sources', 'Embedded J2C Options', and 'External J2C Options'. The 'Data Sources' section includes a 'JDBC provider list' table with one entry 'Derby JDBC Provider (XA)' and its implementation class name. Below it is a table for 'Data source defined in the JDBC provider selected above:' with one entry 'DefaultEJBTimerDataSource'. The 'Resource properties defined in the data source selected above:' section contains a table with properties like 'databaseName', 'shutdownDatabase', and 'dataSourceName'. The 'External J2C Options' section includes expandable sections for 'Application', 'Substitution Variables', 'Authentication', 'Shared Library', and 'Virtual Hosts'. At the bottom, there is a tabbed interface with 'Overview', 'Module', 'Security', 'Deployment', 'Extended Services', and 'Source' tabs.

Name	Implementation Class Name
Derby JDBC Provider (XA)	org.apache.derby.jdbc.EmbeddedXADataSource

Name	JNDI Name	Type
DefaultEJBTimerDataSource	jdbc/DefaultEJBTimerDataSource	V5

Name	Value	Type
databaseName	\${USER_INSTALL_ROOT}/databases/EJBTimers/\${SERVER}/EJBTimerDB	java.lang.String
shutdownDatabase		java.lang.String
dataSourceName		java.lang.String

Embedded J2C Options

External J2C Options

Application

Substitution Variables

Authentication

Shared Library

Virtual Hosts

Overview | Module | Security | **Deployment** | Extended Services | Source

Figure 13-23 WebSphere Enhanced EAR editor

In the Application section in Figure 13-24, you can see the class loader policies and class loader mode configured for each of the containing module. Plants by WebSphere runs fine with the default policies and modes, so they do not need to be changed.

▼ Application

Displays the applications to be deployed onto the servers.

Applications:

PlantsByWebSphere

PlantsByWebSphereWeb.war

PlantsByWebSphereEJB.jar

Classloader mode:

PARENT_FIRST

WAR classloader policy:

MODULE

Start weight:

10

Figure 13-24 Configuring class loader mode and class loader policies

In the Enhanced EAR editor, you also configure resources such as JDBC providers, datasources, JMS resources, class loader policies, JAAS authentication aliases, virtual hosts, and so forth.

To configure the Plants by WebSphere application, we need to add the following:

- ▶ JAAS authentication alias
- ▶ JDBC provider for DB2
- ▶ Data source for DB2 database

Just to show the editor, we will also configure a new virtual host for a domain called `www.plantsbywebsphere.com`.

Configuring a JAAS authentication alias

To configure the JAAS authentication alias, do the following:

1. In the Deployment tab, expand the **Authentication** section.
2. Click the **Add** button.
3. In the dialog box that displays, enter:
 - `plantsbywebsphere` as the alias
 - A user ID with access to the PLANTS database
 - The password for the user ID.
 - Plants by WebSphere as the description

4. Click **OK**. See Figure 13-25 on page 882.

Authentication
Allows to define a login configuration used by Java Authentication and Authorization Service (JAAS).

JAAS authentication list:

Alias	User ID	Description
plantsbywebsphere	db2admin	Plants by WebSphere

Add Edit Remove

Figure 13-25 Configuring JAAS authentication alias for Plants by WebSphere

Configuring a DB2 JDBC provider

To configure the DB2 JDBC provider, do the following:

1. Click the **Add** button next to the JDBC provider list in the **Data Sources** section.
2. In the dialog box:
 - Select **IBM DB2** as the Database type.
 - Select **DB2 Universal JDBC Driver Provider (XA)** as the JDBC provider type.

See Figure 13-26 on page 882.

Database type:

- User-defined
- IBM DB2
- Cloudscape
- Informix
- Sybase
- Oracle

JDBC provider type:

- DB2 Universal JDBC Driver Provider
- DB2 Universal JDBC Driver Provider (XA)
- DB2 Legacy CLI-based Type 2 JDBC Driver
- DB2 Legacy CLI-based Type 2 JDBC Driver (XA)
- DB2 UDB for iSeries (Native)
- DB2 UDB for iSeries (Native XA)

Description: XA DB2 Universal JDBC Driver-compliant Provider, Datasources created

Figure 13-26 Creating a DB2 JDBC Provider

Click **Next**.

3. In the next dialog box, enter a name for the JDBC provider (for administration purposes only) and leave the other properties as the default values. See Figure 13-27 on page 883.

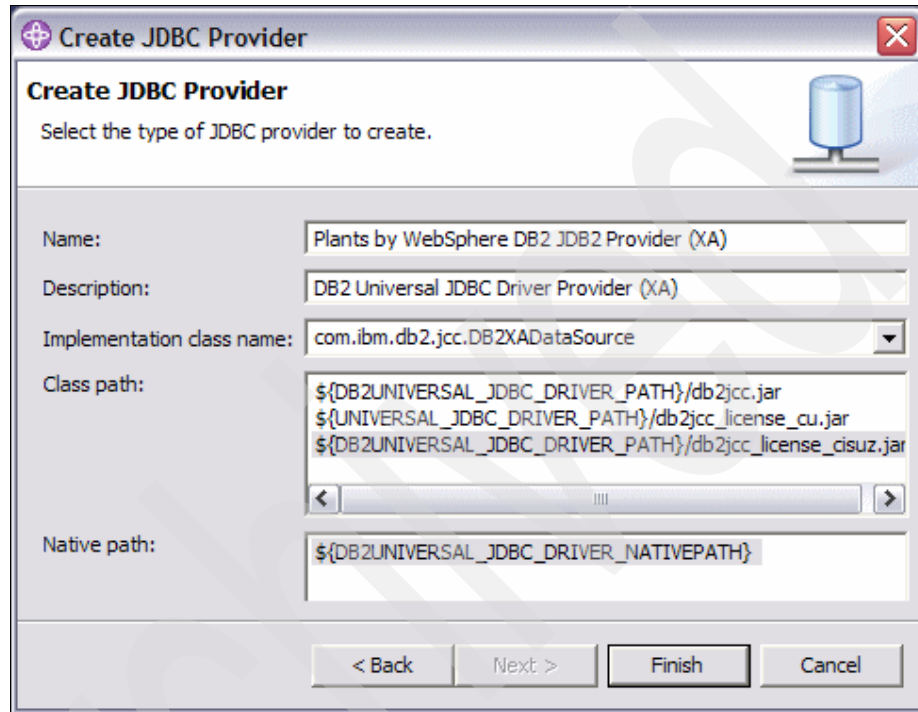


Figure 13-27 Creating a DB2 JDBC provider

Click **Finish**.

4. Select the **Plants by WebSphere DB2 JDBC Provider (XA)** you just created and click the **Add** button next to the Data source list, as in Figure 13-28.

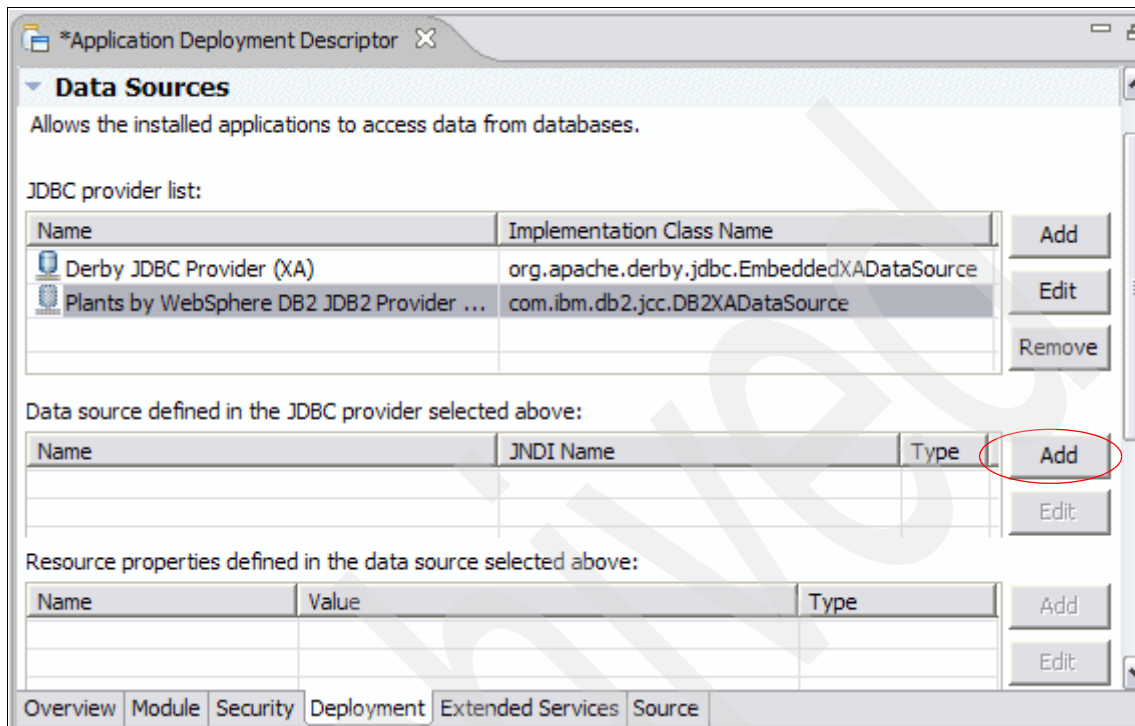


Figure 13-28 Add a data source

5. In the Create a Data Source dialog box, select **DB2 Universal JDBC Driver Provider (XA)** as the JDBC provider type and **Version 5.0 data source** as the data source type, as in Figure 13-29.

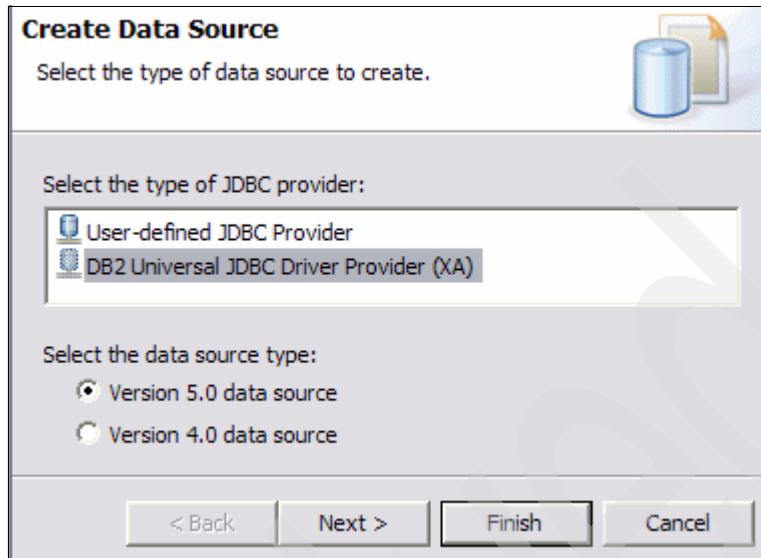
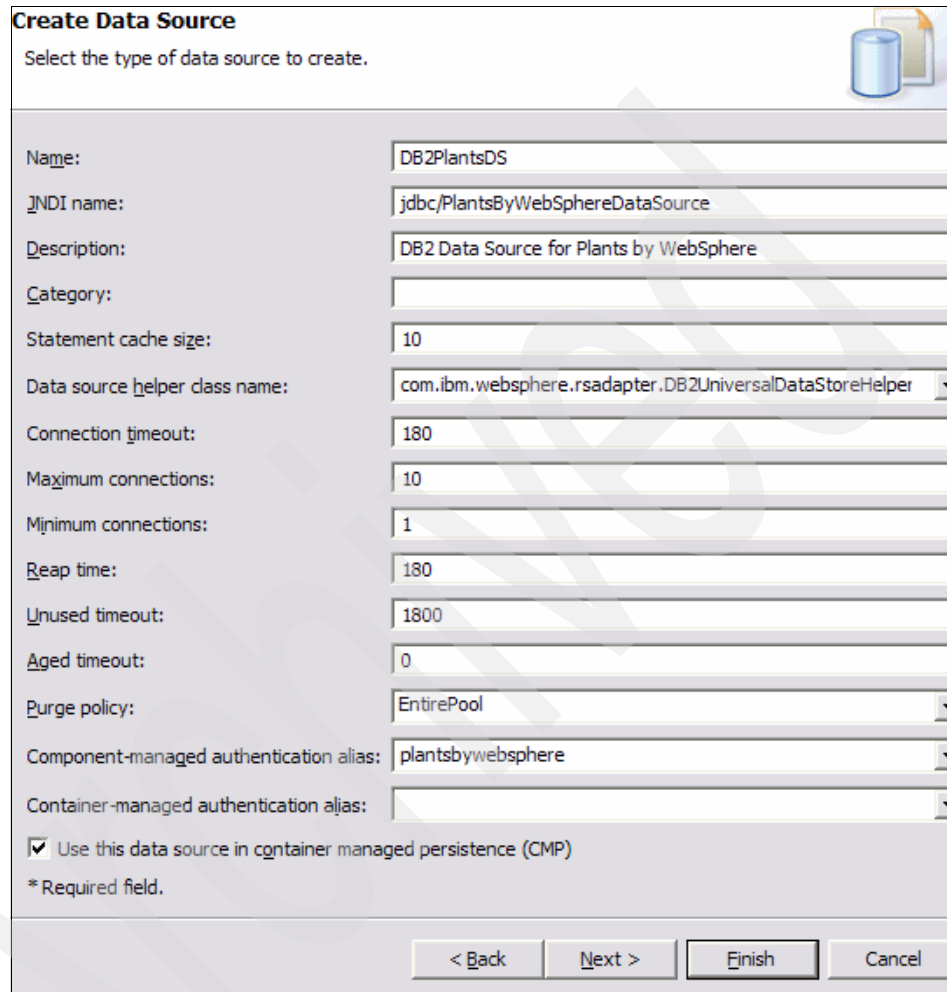


Figure 13-29 Creating a DB2 data source

Click **Next**.

6. In the dialog box displayed enter the appropriate values for the DB2 data source. See Figure 13-30 on page 886.



The image shows a 'Create Data Source' dialog box with a title bar and a close button. The main area contains several labeled text fields and dropdown menus. The fields are: Name (DB2PlantsDS), JNDI name (jdbc/PlantsByWebSphereDataSource), Description (DB2 Data Source for Plants by WebSphere), Category (empty), Statement cache size (10), Data source helper class name (com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper), Connection timeout (180), Maximum connections (10), Minimum connections (1), Reap time (180), Unused timeout (1800), Aged timeout (0), Purge policy (EntirePool), Component-managed authentication alias (plantsbywebsphere), and Container-managed authentication alias (empty). There is a checkbox for 'Use this data source in container managed persistence (CMP)' which is checked. Below the checkbox is a note '* Required field.' At the bottom right are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

Create Data Source

Select the type of data source to create.

Name: DB2PlantsDS

JNDI name: jdbc/PlantsByWebSphereDataSource

Description: DB2 Data Source for Plants by WebSphere

Category:

Statement cache size: 10

Data source helper class name: com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper

Connection timeout: 180

Maximum connections: 10

Minimum connections: 1

Reap time: 180

Unused timeout: 1800

Aged timeout: 0

Purge policy: EntirePool

Component-managed authentication alias: plantsbywebsphere

Container-managed authentication alias:

☒ Use this data source in container managed persistence (CMP)

* Required field.

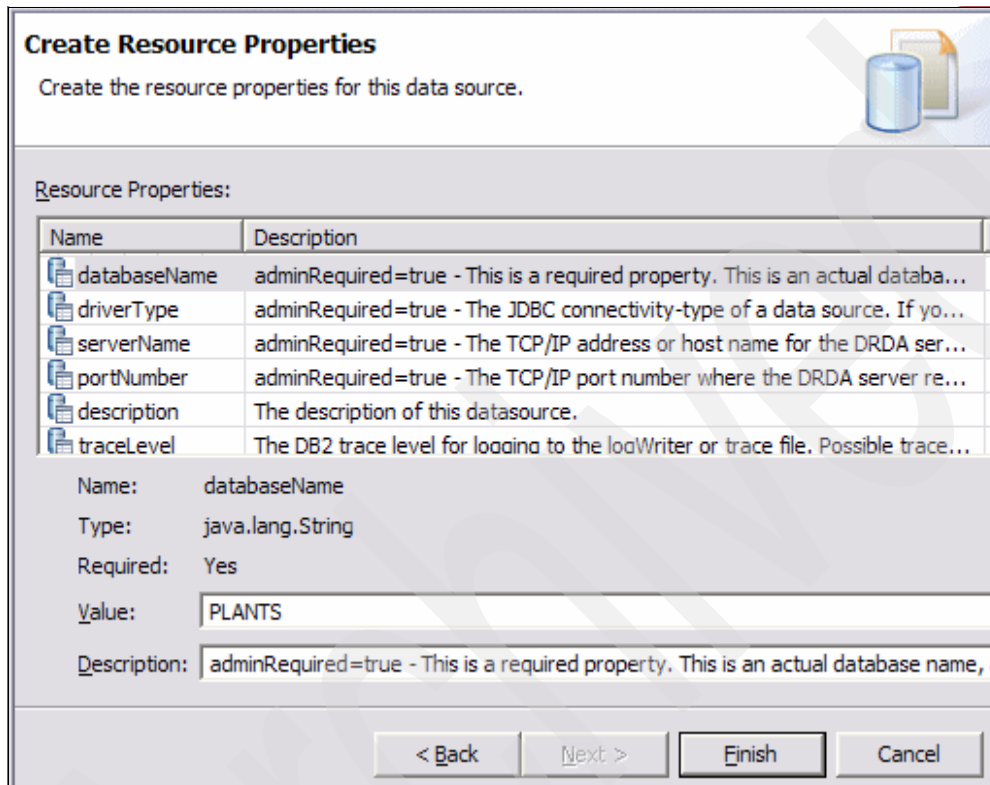
< Back Next > Finish Cancel

Figure 13-30 Creating a DB2 data source

- Enter DB2PlantsDS as the name.
- Enter jdbc/PlantsByWebSphereDataSource as the JNDI name.
- Enter DB2 Data Source for Plants by WebSphere as the description.
- Select **plantsbywebsphere** as the Component-managed authentication alias.
- Check **Use this data source in container manager persistence (CMP)**.

Click **Next**.

7. In the Create Resource Properties dialog box, select **databaseName** and enter PLANTS as the value. Then also select the **driverType** and enter 2 as the value. See Figure 13-31 on page 887.



The dialog box is titled "Create Resource Properties" and contains the instruction "Create the resource properties for this data source." It features a table of resource properties and a section for editing a specific property.

Name	Description
databaseName	adminRequired=true - This is a required property. This is an actual databa...
driverType	adminRequired=true - The JDBC connectivity-type of a data source. If yo...
serverName	adminRequired=true - The TCP/IP address or host name for the DRDA ser...
portNumber	adminRequired=true - The TCP/IP port number where the DRDA server re...
description	The description of this datasource.
traceLevel	The DB2 trace level for loading to the logWriter or trace file. Possible trace...

Below the table, the "Name" field is set to "databaseName", the "Type" is "java.lang.String", and "Required" is "Yes". The "Value" field contains "PLANTS". The "Description" field shows the same text as the table description for databaseName.

At the bottom, there are four buttons: "< Back", "Next >", "Finish", and "Cancel".

Figure 13-31 Setting database properties for DB2 data source

JDBC driver type 2 means that the database is local to the machine running the WebSphere application, or that it has a DB2 Connect™ client that can make the database look local.

Click **Finish**.

When you are finished, your data source configuration should look like Figure 13-32 on page 888.

The screenshot shows the 'WebSphere Deployment' console with the 'Data Sources' section expanded. It displays the 'JDBC provider list' and the 'Data source defined in the JDBC provider selected above'.

JDBC provider list:

Name	Implementation Class Name
Derby JDBC Provider (XA)	org.apache.derby.jdbc.EmbeddedXADataSource
Plants by WebSphere DB2 JDB2 Provider ...	com.ibm.db2.jcc.DB2XADataSource

Data source defined in the JDBC provider selected above:

Name	JNDI Name	Type
DB2PlantsDS	jdbc/PlantsByWebSphereDataSource	V5

Resource properties defined in the data source selected above:

Name	Value	Type
databaseName	PLANTS	java.lang.String
driverType	2	java.lang.Integer
serverName		java.lang.String
portNumber	50000	java.lang.Integer

The console has tabs for Overview, Module, Security, Deployment, Extended Services, and Source.

Figure 13-32 DB2 data source configured

Adding a virtual host

To configure a virtual host, do the following:

1. Expand the **Virtual Hosts** section of the Deployment tab and click the **Add** button next to the Virtual host name list.
2. In the Add Host Name Entry dialog box, enter `plantsbywebsphere_host` and click **OK**. Your new virtual host will appear in the Virtual Hosts list. See Figure 13-33.

Virtual Hosts
Enables a single host machine to resemble multiple host machines.

Virtual host name list:

Host Name	
plantsbywebsphere_host	

Add Edit Remove

Figure 13-33 Add a new virtual host

- Click the **Add** button next to the Host aliases list.
- In the Add Host Alias Entry dialog box, enter `www.plantsbywebsphere.com` for the host name and 80 for the port number. Click **OK**.

Repeat the procedure to add port number 9081 as well. We will use this port when we deploy the application later.

The host aliases will appear in the list (Figure 13-34).

Virtual Hosts
Enables a single host machine to resemble multiple host machines.

Virtual host name list:

Host Name	
plantsbywebsphere_host	

Add Edit Remove

Host aliases are defined by the virtual host name selected above:

Host Name	Port	
www.plantsbywebsphere.com	80	
www.plantsbywebsphere.com	9081	

Add Edit Remove

Figure 13-34 Configuring the virtual host for Plants by WebSphere

- When you are finished, press Ctrl-S to save the deployment descriptor editor.

Setting default virtual host for Web modules

Just because we have configured a new virtual host, `plantsbywebsphere_host`, in the Enhanced EAR file does not mean that all our Web modules automatically use it.

The default virtual host for a Web module created in the Application Server Toolkit or Rational Application Developer is `default_host`, which is also the case for the Web module of the Plants by WebSphere application. This setting can be found in the `ibm-web-bdn.xmi` file in the `/WEB-INF` directory of each Web module.

To configure the Web modules to default to the `plantsbywebsphere_host` instead, do the following:

1. Expand **Dynamic Web Project** in the Project Explorer view.
2. Expand the **PlantsByWebSphereWeb** project and double-click **Deployment Descriptor**.
3. Scroll to the bottom of the Overview page and replace `default_host` with `plantsbywebsphere_host`, as shown in Figure 13-35.

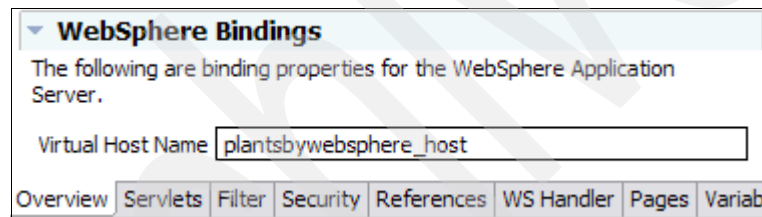


Figure 13-35 Setting default virtual host for a Web module

4. Save the deployment descriptor by pressing Ctrl-S and then close it.

Examining the WebSphere Enhanced EAR file

The information about the resources configured is stored in the `ibmconfig` subdirectory of the EAR file's `META-INF` directory. Expanding this directory reveals the well-known directory structure for a cell configuration, as seen in Figure 13-36 on page 891. You can also see the scope level where each resource is configured.

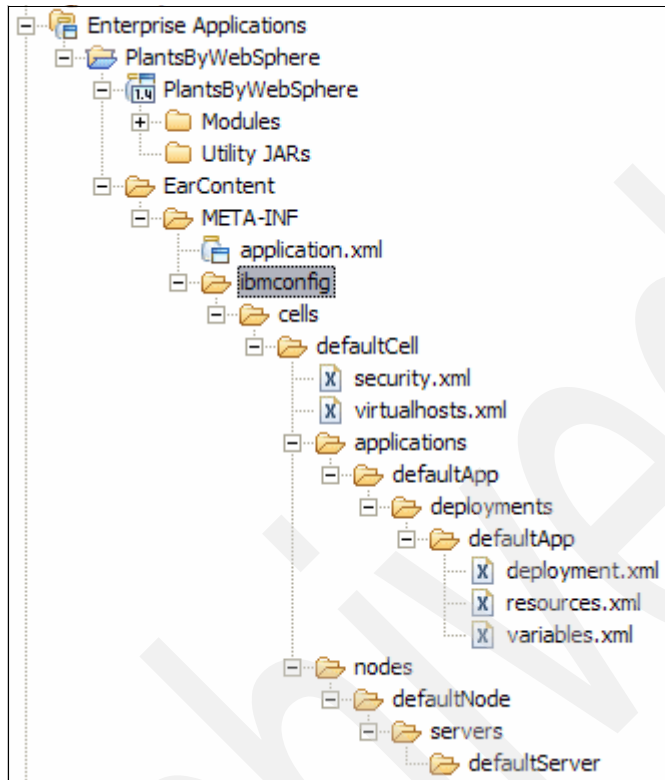


Figure 13-36 Enhanced EAR file contents

After you have re-packaged the application into an Enhanced EAR, export it as explained in “Exporting the PlantsByWebSphere EAR file” on page 877.

At deployment time, WebSphere Application Server uses this information to automatically create the resources.

13.11 Packaging recommendations

Here are some basic rules to consider when packaging an enterprise application:

- ▶ The EJB JAR modules and Web WAR modules comprising an application should be packaged together in the same EAR module.
- ▶ When a Web module accesses an EJB module, you should not package the EJB interfaces and stubs in the WAR modules. Thanks to the class loading architecture, EJB stubs and interfaces are visible by default to WAR modules.

- ▶ Utility classes used by a single Web module should be placed within its WEB-INF/lib folder.
- ▶ Utility classes used by multiple modules within an application should be placed at the root of the EAR file as Utility Projects, so they are accessible both by servlets and EJBs.
- ▶ Utility classes used by multiple applications can be placed on a directory referenced through a shared library definition.

See 12.5, “Learning class loaders by example” on page 811 for more details on how WebSphere finds and loads classes.

Deploying applications

In Chapter 13, “Packaging applications” on page 829, we discuss how to use the Application Server Toolkit to perform common tasks for packaging an application. In this chapter, we show you how to deploy the application. We take you through setting up the environment for the application, and then deploying the application itself. Next, we explain how to deploy the client part of the application. The deployment tasks in this chapter can also be automated using command-line tools, as explained in Chapter 5, “Administration with scripting” on page 249.

WebSphere Application Server V6 supports the J2EE Deployment API Specification (JSR-88), which defines standard APIs to enable deployment of J2EE applications and stand-alone modules to J2EE application servers. For more information about how to use this API, see the WebSphere Information Center by searching for JSR-88 and browse to the section discussing *Installing J2EE modules with JSR-88*.

The topics in this chapter include:

- ▶ Preparing the environment
- ▶ Generating deployment code
- ▶ Deploying the application
- ▶ Deploying application clients
- ▶ Updating applications

14.1 Preparing the environment

In this chapter, we show you how to set up a fairly complete environment for the Plants by WebSphere application and deploy the EAR file. You will not always need or want to customize the environment as extensively as we do in this chapter. Some steps are optional. If all you want to do is deploy your application quickly, using the WebSphere defaults for directory names, log files, and so forth, skip to 14.3, “Deploying the application” on page 913.

The steps in this section are performed typically by the application deployer. To deploy the Plants by WebSphere application, do the following:

1. Create the DB2 database for Plants by WebSphere. This step is required.
2. Create an environment variable for Plants by WebSphere server. This step is optional.
3. Create an application server to host the application. This step is optional.
4. Customize the IBM HTTP Server configuration. This step is optional.
5. Define a JDBC provider, data source, and authentication alias. This step is required if you are not using an Enhanced EAR.
6. Define virtual hosts. This step is optional and not required if you are using an Enhanced EAR.

If the application to be deployed is a WebSphere Enhanced EAR file, the resources configured in the Enhanced EAR file are created automatically when the application is deployed.

14.1.1 Creating the Plants by WebSphere DB2 database

The WebSphere samples by default use Cloudscape as the database. However, in this chapter, we will configure Plants by WebSphere to use a DB2 UDB 8.2 database instead.

To set up the DB2 database, make sure you have DB2 installed and running. Then run the following commands:

1. Select **Start** → **Programs** → **IBM DB2** → **Command Line Tools** → **Command Window**.
2. Create the database using the commands in Example 14-1 on page 895.

Example 14-1 Creating the DB2 database

```
DB2 CREATE DATABASE PLANTS PAGESIZE 16 K
D2B CONNECT TO PLANTS USER <user_id> USING <password>
DB2 -tvf Table.ddl
db2 connect reset
```

The Table.ddl file is located in the PlantsByWebSphereEJB\ejbModule\META-INF\backends\DB2UDBNT_V82_1 directory of your Application Server Toolkit workspace. See “Creating a new database mapping and schema” on page 848.

Note: Because the Plants by WebSphere application stores images in the database using a BLOB, we needed a tablespace with a larger pagesize than the default of 4 KB. To specify this as the default when the database was created, we used the PAGESIZE keyword. This is supported on DB2 UDB 8.2.2, also called 8.1 Fix Pack 9, and later. We used DB2 Express 8.2.5

14.1.2 Creating an environment variable

We recommend that you use WebSphere environment variables, rather than hard-coded paths when deploying an application. In the following steps, we assume you have declared a PLANTSBYWEBSphere_ROOT variable. You will use it when specifying, for example, the JVM log's location.

Be certain you declare this variable at the right scope. For example, if you define this variable at the application server scope, it will only be known at that level. As long as you work with the WebSphere Application Server Base or Express editions, this is fine. But if you later decide to use the Network Deployment edition and you create a cluster of application servers, the PLANTSBYWEBSphere_ROOT variable will need to be defined at the cluster or cell level.

Use the steps in 4.1.10, “Using variables” on page 156 to create a PLANTSBYWEBSphere_ROOT variable with a value of C:\apps\PlantsByWebSphere.

There are several ways to organize WebSphere applications. Some companies prefer to create a directory for each application, as we do in our example, such as C:\apps\<application_name>, and keep all resources and directories required by the application in subdirectories under this directory. This strategy works well when deploying only one application per application server, again as we do in our example, because the application server's log files could then all be changed to point to c:\apps\<application_name>\logs.

Other companies prefer to organize resources by resource type, and so create directories such as `c:\apps\logs\<application_name.log>`, `c:\apps\properties\<application_name.properties>`, and so on.

And some companies prefer to stick with the vendor defaults as far as possible. For WebSphere, that means that the applications are installed in the `<profile_home>/installedApps` directory and the logs files are written to the `<profile_home>/logs/<server_name>` directory.

Which option you choose is a matter of personal preferences and corporate guidelines.

Note: Make sure you create the target directory you specify for the `PLANTSBYWEBSPPHERE_ROOT` variable before proceeding. If the directory is not created, the application server will not start.

14.1.3 Creating the Plants by WebSphere application server

In a distributed server environment, you have the option of using a single application server, or creating multiple application servers or clusters.

The advantages of deploying multiple applications to a single application server is that it consumes less resources. There is no overhead for any extra application server processes. Another benefit is that applications can make in-process calls to each other. For example, servlets in one EAR file could access Local interfaces of EJBs in another EAR file.

One alternative to using a single application server is to deploy each application to its own server. The advantages of deploying only one application on an application server is that it gives you greater control over the environment. The JVM heap sizes and environment variables are set at application server level, so all applications running in an application server share the JVM memory given to the application server and they would all see the same environment variables. Running each application in its own application server could also make it easier to perform problem determination. For example, if an application runs amok and consumes a lot of CPU, you could see which application it is by looking at the process ID of the application server.

In our example, we create a unique application server on which to run the Plants by WebSphere sample application.

Note: For a full discussion of application server properties, see 4.4, “Working with application servers” on page 170.

To create an application server, do the following:

1. Select **Servers** → **Application Servers**.
2. Click the **New** button and provide the information shown in Figure 14-1 on page 897.

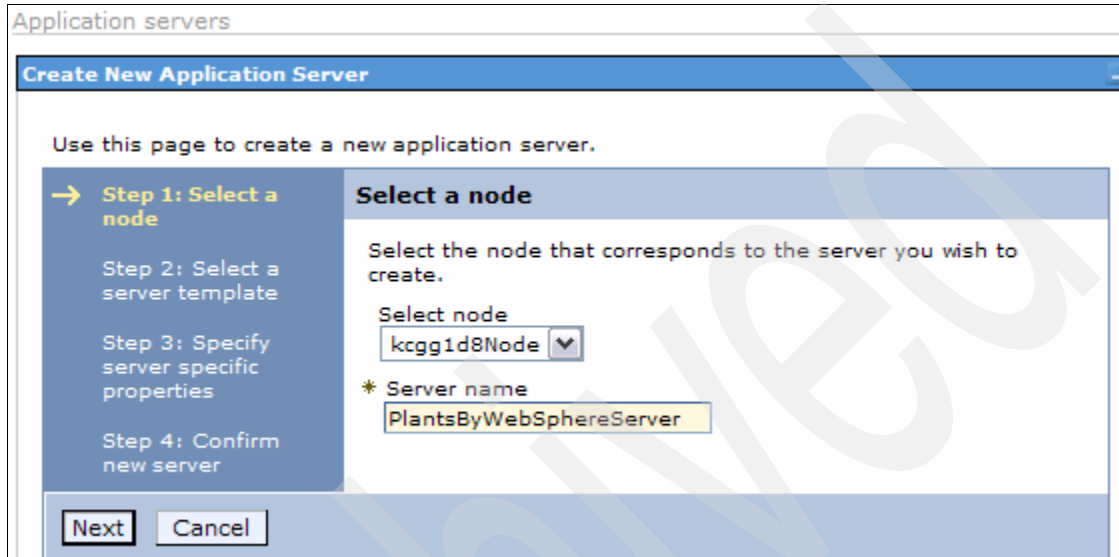


Figure 14-1 Creating the WebSphere Bank application server

- Node
Select the node on which the application server will be created.
 - Server name
Enter the application server name, such as PlantsByWebSphereServer.
- Click **Next**.
3. In Step 2, select which server template to use as the base for this new application server. The DeveloperServer template is used when setting up a server for development use and will cause the JVM to prioritize quick startup (by disabling bytecode verification, and performing JIT compilations with a lower optimization level). This option should not be used on a production server, where long run throughput is more important than early server startup. If you have not created any templates on your own, then select the WebSphere **default**. Otherwise, select the server template you want to use and click **Next**.
 4. In step 3, you can select, if you want, WebSphere to generate a unique set of port numbers for this application server. This ensures the ports defined for

this server does not conflict with another server currently configured on this node. Check the **Generate Unique Http Ports** box and click **Next**.

5. On the Summary page, click **Finish**.

Changing the working directory

The next thing we want to do is to change the working directory for the application server process. This directory is the relative root for searching files. For example, if you do a `File.open("foo.gif")`, `foo.gif` must be present in the working directory. This directory will be created by WebSphere if it does not exist. We recommend that you create a specific working directory for each application server.

1. Select the server, **PlantsByWebSphereServer**, you just created.
2. Expand the **Java and Process Management** in the Server Infrastructure section and select **Process Definition**.
3. Scroll down the page and change the working directory from `${USER_INSTALL_ROOT}` to `${PLANTSBYWEBSPPHERE_ROOT}/workingDir`.
4. Click **OK**.

Note: The working directory will not be created if you use a composed path, such as `C:/apps/PlantsByWebSphere/workingDir`. If you want to use such a path, create it before starting the application server, or the startup sequence fails.

Changing the logging and tracing options

Next, we want to customize the logging and tracing properties for the new application server. There are several ways to access the logging and tracing properties for an application server:

- ▶ Select **Troubleshooting** → **Logs and Trace** in the navigation bar, then select a server.
- ▶ Select **Servers** → **Application Servers**, select a server, and then select **Logging and Tracing** from the Troubleshooting section.
- ▶ Select **Servers** → **Application Servers**, select a server, select **Process definition** from the Java and Process Management section. Select **Logging and Tracing** from the Additional Properties section.

Because we have just finished updating the application server process definition, we will take the third navigation path to customize the location of the JVM logs, the diagnostic trace logs, and the process logs.

1. Select **Logging and Tracing**.

2. Select **JVM Logs**.

This allows you to change the JVM standard output and error file properties. Both are rotating files. You can choose to save the current file and create a new one, either when it reaches a certain size, or at a specific moment during the day. You can also choose to disable the output of calls to `System.out.print()` or `System.err.print()`.

We recommend that you specify a new file name, using an environment variable to specify it, such as:

```
${PLANTSBYWEBSPIHERE_ROOT}/logs/SystemOut.log
```

```
${PLANTSBYWEBSPIHERE_ROOT}/logs/SystemErr.log
```

Click **OK**.

3. Select **Diagnostic Trace**.

Each component of the WebSphere Application Server is enabled for tracing with the JRes interface. This trace can be changed dynamically while the process is running using the Runtime tab, or added to the application server definition from the Configuration tab. As shown in Figure 14-2, the trace output can be either directed to memory or to a rotating trace file.

Change the trace output file name so the trace is stored in a specific location for the server using the `PLANTSBYWEBSPIHERE_ROOT` variable and select the **Log Analyzer** format.

Configuration

General Properties

☒ Enable Log

Trace Output

☐ Memory Buffer

* Maximum Buffer Size
8 thousand entries

☒ File

* Maximum File Size
20 MB

* Maximum Number of Historical Files
1

* File Name
\${PLANTSBYWEBSphere_ROOT}/trace.log

Trace Output Format
Log Analyzer

Additional Properties

[Change Log Detail Levels](#)

Buttons: Apply, OK, Reset, Cancel

Figure 14-2 Specifying diagnostic trace service options

Click **OK**.

4. Select **Process Logs**.

Messages written by native code (JNI) to standard out and standard error streams are redirected by WebSphere to process logs, usually called native_stdout.log and native_stderr.log. Change the native process logs to:

`${PLANTSBYWEBSphere_ROOT}/logs/native_stdout.log`

`${PLANTSBYWEBSphere_ROOT}/logs/native_stderr.log`

Click **OK**.

5. All log files produced by the application server are now redirected to the `${PLANTSBYWEBSphere_ROOT}/logs` directory. Save the configuration.

Note: The rest of this example assumes a default HTTP port of 9081 for the Web container. Before proceeding, check the application server you created to determine the port you should use:

1. Select **Servers** → **Application Servers**.
2. Select the **PlantsByWebSphereServer**.
3. Select **Ports** in the Communications section.
4. Scroll down the page and note the port listed for WC_defaulthost.

14.1.4 Defining the Plants by WebSphere virtual host

Enhanced EAR file users: If you are using an Enhanced EAR file, the virtual host can be defined at packaging time. See “Adding a virtual host” on page 888.

Web modules need to be bound to a specific virtual host. For our sample, we chose to bind the PlantsByWebSphereWeb module to a specific virtual host called plantsbywebsphere_host. This virtual host has the following host aliases:

- ▶ www.plantsbywebsphere.com:80
- ▶ www.plantsbywebsphere.com:9081

Any request starting with <plantsbywebsphere_host_alias>/PlantsByWebSphere, such as `http://www.plantsbywebsphere.com:9081/PlantsByWebSphere`, is served by the Plants by WebSphere application.

Tip: You can restrict the list of hosts used to access the Web application by removing hosts from the virtual host definition.

Imagine you want to prevent users from directly accessing the Plants by WebSphere application from the WebSphere internal HTTP server when they invoke `http://www.plantsbywebsphere.com:9081/PlantsByWebSphere`. In other words, you want to force all requests to go through the Web server plug-in. You can achieve this by removing `www.plantsbywebsphere.com:9081` from the virtual host aliases list.

To create the plantsbywebsphere_host virtual host, do the following:

1. Select the **Environment** → **Virtual Hosts** entry in the navigation pane.
2. Click **New**.
3. Enter the virtual host name, plantsbywebsphere_host.
4. Click **Apply**.
5. Select **Host Aliases** in the Additional Properties section.

6. Add the two aliases shown in Figure 14-3 by clicking **New**, entering the values, and clicking **OK**.

New Delete		
		
Select	Host Name ↕	Port ↕
☐	www.plantsbywebsphere.com	80
☐	www.plantsbywebsphere.com	9081
Total 2		

Figure 14-3 WebSphere Bank virtual host aliases

7. Click **OK**.
8. Save the configuration.

14.1.5 Creating the virtual host for IBM HTTP Server and Apache

Now that we have defined a `plantsbywebsphere_host` virtual host, we need to configure the Web server to serve the host aliases in the virtual host. The steps below are valid for both the IBM HTTP Server V6 and Apache 2.0.

Configuring virtual hosting

Note: It is not necessary to create a virtual host in `httpd.conf`. It is required only if you want to customize the configuration, for example, by separating the logs for each virtual host. This is not normally done.

Creating virtual hosts is done using the `VirtualHost` directive, as in Example 14-2.

Example 14-2 Using `VirtualHost`

```
<VirtualHost www.plantsbywebsphere.com:80>
  ServerAdmin webmaster@plantsbywebsphere.com
  ServerName www.plantsbywebsphere.com
  DocumentRoot "C:\IBM\HTTPServer\htdocs\plantsbywebsphere"
  ErrorLog logs/plantsbywebsphere_error.log
  TransferLog logs/plantsbywebsphere_access.log
</VirtualHost>
```

If you want to have multiple virtual hosts for the same IP address, you must use the `NameVirtualHost` directive. See Example 14-3.

Example 14-3 Using the `NameVirtualHost` and `VirtualHost` directives

NameVirtualHost 9.23.456.789:80

```
<VirtualHost itso_server:80>
  ServerAdmin webmaster@itso_server.com
  ServerName itso_server
  DocumentRoot "C:\IBM\HTTPServer\htdocs\itso_server"
  ErrorLog logs/itso_server_error.log
  TransferLog logs/itso_server_access.log
</VirtualHost>

<VirtualHost www.plantsbywebsphere.com:80>
  ServerAdmin webmaster@plantsbywebsphere.com
  ServerName www.plantsbywebsphere.com
  DocumentRoot "C:\IBM\HTTPServer\htdocs\plantsbywebsphere"
  ErrorLog logs/plantsbywebsphere_error.log
  TransferLog logs/plantsbywebsphere_access.log
</VirtualHost>
```

The `www.plantsbywebsphere.com` and the `itso_server` hosts have the same IP address, 9.23.456.789. We have set this by inserting the following line in the machine hosts file, located in `%windir%\system32\drivers\etc` or in `/etc` on UNIX systems):

```
9.23.456.789 www.plantsbywebsphere.com itso_server
```

In a real-life environment, this would probably be achieved by creating aliases at the DNS level. In any event, you must be able to ping the host you have defined, using commands such as **ping `www.plantsbywebsphere.com`**.

As you can see in Example 14-3, each virtual host has a different document root. Make sure that the directory you specify exists before you start the HTTP server. While testing the setup, you can place an `index.html` file at the document root stating which virtual host is being called. This lets you easily see which virtual host is being used.

You must restart the IBM HTTP Server to apply these changes. If you are running a Windows system, we recommend that you try to start the server by running **apache.exe** from the command line rather than from the Services window. This allows you to spot error messages thrown at server startup.

If your virtual hosts are correctly configured, invoking `http://www.plantsbywebsphere.com` or `http://itso_server` returns different HTML pages.

14.1.6 Creating a DB2 JDBC provider and data source

Enhanced EAR file users: If you are using an Enhanced EAR file, the JDBC provider, data source, and J2C authentication entry can be defined at packaging time. See “Configuring a DB2 JDBC provider” on page 882.

The Plants by WebSphere sample application uses a relational database, via entity beans, to store information. To access this database, you need to define a data source and then associate it with the entity beans. The Plants by WebSphere sample application is configured for Cloudscape by default. In Chapter 13, “Packaging applications” on page 829, however, we modified the Plants by WebSphere application to run against a DB2 database instead. We will now create the DB2 JDBC provider, data source, and JAAS authentication alias required to run against DB2.

For detailed information about JDBC providers and data sources, refer to 6.2, “JDBC resources” on page 305.

Configuring environment variables for DB2 JDBC driver

For the DB2 Universal JDBC Provider to find its classes, the DB2UNIVERSAL_JDBC_DRIVER_PATH and DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH environment variables must be set up. To set up these variables, do the following:

1. Select **Environment** → **WebSphere Variables**.
2. Locate and click the **DB2UNIVERSAL_JDBC_DRIVER_PATH** entry.
3. In the value field, enter the path to where the DB2 JDBC driver is located. For example, for DB2, the location is likely to be:

C:\Program Files\IBM\SQLLIB\java

See Figure 14-4.

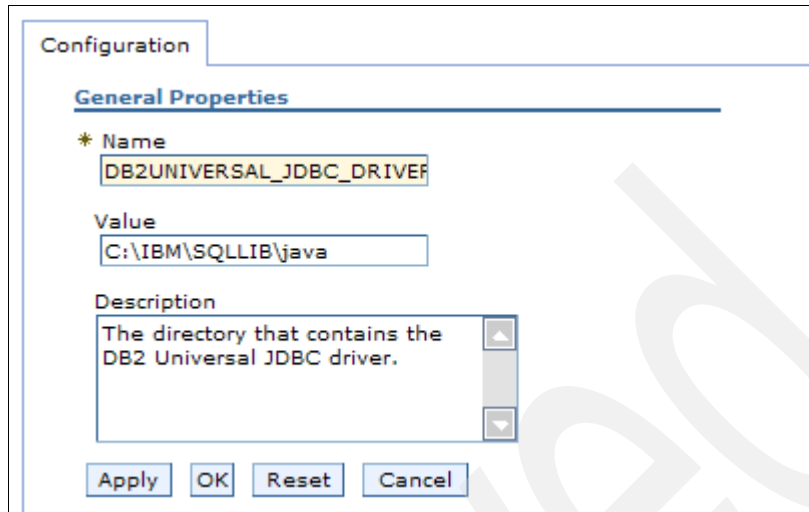


Figure 14-4 Configuring DB2 Driver Path

Click **OK**.

4. Repeat the process for the DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH variable. For DB2, it should use the same path, C:\Program Files\IBM\SQLLIB\java.

Configuring J2C authentication data

The user ID and password required to access the database are specified in a J2C authentication data entry.

1. Select **Security** → **Secure administration, applications, and infrastructure**. Expand the Java Authentication and Authorization Service section and select **J2C authentication data**.

2. Click **New**, and specify the following information to create the authentication data. Once completed, the authentication information should be similar to Figure 14-5.
 - Alias
Enter the name of the security information alias, such as webspherebank.
 - User ID
Enter a user ID with the proper authority to access the database.
 - Password
Enter the password for the user ID.

Secure administration, applications, and infrastructure

Secure administration, applications, and infrastructure > JAAS - J2C authentication data > New

Specifies a list of user identities and passwords for Java(TM) 2 connector security to use.

Configuration

General Properties

* Alias
plantsbywebsphere

* User ID
db2admin

* Password

Description
Plants by WebSphere authen

Apply OK Reset Cancel

Figure 14-5 Creating WebSphere Bank JAAS authentication alias

3. Click **OK**.

Creating the Plants by WebSphere JDBC provider

The following steps take you through the creation of a JDBC provider targeting a DB2 database. To create a JDBC provider from the administrative console, do the following:

1. Expand the **Resources** entry and select the **JDBC Providers** entry.

2. Select the scope of this resource. In a stand-alone server environment, it is sufficient to create the data source at the server level. Otherwise, define it at the cluster or cell level. A rationale for this is to be able to share the definition across multiple servers in a cluster. To change this, select the server you are deploying to in the scopes list and click **Apply**.
3. Click the **New** button.
4. In the Configuration dialog box, select the general properties for the JDBC provider, as shown in Figure 14-6.

→ Step 1: Create new JDBC provider

Step 2: Enter database class path information

Step 3: Summary

Create new JDBC provider

Set the basic configuration values of a JDBC provider, which encapsulates the specific vendor JDBC driver implementation classes that are required to access the database. The wizard fills in the name and the description fields, but you can type different values.

Scope
cells:kcg1d8Cell:nodes:kcg1d8Node

* Database type
DB2

* Provider type
DB2 Universal JDBC Driver Provider

* Implementation type
XA data source

* Name
DB2 Universal JDBC Driver Provider (XA)

Description
XA DB2 Universal JDBC Driver-compliant Provider. Datasources created under this provider support the use of XA to perform 2-phase commit processing. Use of driver type 2 on WAS z/OS is not supported for datasources created under this provider.

Next Cancel

Figure 14-6 Creating a DB2 JDBC provider

- Database type: DB2
- Provider type: DB2 Universal JDBC Driver Provider
- Implementation type: XA data source

- Name: DB2 Universal JDBC Driver Provider (XA)

Note: We used the DB2 XA-capable JDBC Driver for the Plants by WebSphere sample. If your application does not require two-phase commit capabilities, use the regular driver. If using an XA-capable driver, it is a best practice to indicate that it is an XA-capable driver by including XA in its name, such as MyJDBCdriverXA.

5. The next window allows you to change the location for the JDBC driver files, but because we configured the paths earlier, we do not need to do it again. Click **Next**.
6. On the Summary page, click **Finish**.

Creating the Plants by WebSphere data source

The next step is to create the data source for the Plants by WebSphere DB2 database. To create a data source, do the following:

1. Select **Resources** → **JDBC Providers**.
2. Select the **DB2 Universal JDBC Driver Provider (XA)** and select **Data Sources** under Additional Properties.
3. Click **New** to add the new data source. See Figure 14-7.

Create a data source

→ Step 1: Enter basic data source information

Step 2: Enter database specific properties for the data source

Step 3: Summary

Enter basic data source information

Set the basic configuration values of a data source for association with a JDBC provider. A data source supplies the physical connections between an application server and the database.

Requirement: Use the Data sources (WebSphere(R) Application Server) console pages if your applications are based on the Enterprise JavaBeans (TM) (EJB) 1.0 specification or the Java(TM) Servlet 2.2 specification.

Scope

cells:kcgg1d8Cell:nodes:kcgg1d8Node

JDBC provider name

DB2 Universal JDBC Driver Provider (XA)

* Data source name

PlantsByWebSphereDS

* JNDI name

jdbc/PlantsByWebSphereDataSource

Component-managed authentication alias and XA recovery authentication alias

Select a component-managed authentication alias. The selected authentication alias will also be set as the XA recovery authentication alias if your JDBC Provider supports XA. If you choose to [create a new J2C authentication alias](#), the wizard will be canceled.

kcgg1d8CellManager/plantsbywebsphere ▼

Next

Cancel

Figure 14-7 Plants by WebSphere basic data source properties

- Data source name
Enter the data source name, which must be unique in the administrative domain or cell. We recommend that you use a value indicating the name of the database this data source is targeting, such as “PlantsByWebSphereDS”.
- JNDI name
Enter the name by which applications access this data source. If not specified, the JNDI name defaults to the data source name prefixed with jdbc/. For the Plants by WebSphere, set this field to jdbc/PlantsByWebSphereDataSource. This value can be changed at any time after the data source has been created.

- Component-managed authentication alias

Enter the J2C alias used for connecting to the data source by selecting the authentication alias created previously, <cell name>/plantsbywebsphere.

Note: The component-managed authentication alias is used when the res-auth tag in the deployment descriptor is set to Application and the application itself does not specify a user ID when obtaining a connection (that is, when `datasource.getConnection()` is called). The container-managed application alias (used when res-auth tag is set to Container) has been deprecated since WebSphere Application Server V6.0.

4. Click **Next**. On the second page, enter the information shown in Figure 14-8.

Step 1: Enter basic data source information

→ Step 2: Enter database specific properties for the data source

Step 3: Summary

Enter database specific properties for the data source

Set these database-specific properties, which are required by the vendor JDBC driver to support the connections that are made to this data source.

- * Database name:
- * Driver type:
- Server name:
- Port number:
- ☒ Use this data source in container managed persistence

Previous Next Cancel

Figure 14-8 Plants by WebSphere database specific properties

- Database name

Enter the name of the database, PLANTS in our example.

- Driver type

Select the driver type to use. In our environment, the database is on the same machine as our WebSphere installation, so we can use a type 2 driver and do not need to enter the server name. If the database is on a remote server from the WebSphere machine, you would either use a type

- 4 driver (and supply the name of the database server) or you would use a type 2 driver and catalog the database on the WebSphere machine.
- Use this Data Source in container-managed persistence (CMP)
 - Check this option to indicate that the Plants by WebSphere data source is used for persisting the application entity beans.
- 5. Click **Next**, and then on the summary page, click **OK**.
- 6. Save the configuration.
- 7. Test the connection by selecting the data source and clicking the **Test Connection** button.

14.2 Generating deployment code

At some point, you will need to generate the deployment code for the EJBs. You can do this in Rational Application Developer, in the Application Server Toolkit, from the command line, or at deployment time using the install windows in the WebSphere administrative console. The deployment code should match the version of the run time target.

For information about how to generate the deployment code using the Application Server Toolkit, see “Creating a new database mapping and schema” on page 848.

14.2.1 Using EJBDeploy command-line tool

You can generate the EJB deployment code using the EJBDeploy command-line tool. The syntax of the EJBDeploy command is shown in Example 14-4.

Example 14-4 EJBDeploy syntax

EJBDeploy (v6.1, o0619.34)

Syntax: EJBDeploy inputEar workingDirectory outputEar [options]

Options:

-cp "jar1;jar2"	List of jar filenames required on classpath
-codegen	Only generate the deployment code, do not run RMIC or Javac
-bindear:options	Bind references within the EAR
-dbschema schema	The name of the schema to create
-dbvendor DBTYPE	Set the database vendor type, to one of:
DB2UDB_V81 DB2UDB_V82	
DB2UDBOS390_V7 DB2UDBOS390_V8 DB2UDBOS390_NEWFN_V8	
DB2UDBISERIES_V53 DB2UDBISERIES_V54	
DERBY_V10	
INFORMIX_V93 INFORMIX_V94 INFORMIX_V100	

MSSQLSERVER_2000	MSSQLSERVER_2005
ORACLE_V9I	ORACLE_V10G
SYBASE_V1250	SYBASE_V15
SQL92 (*)	SQL99 (*) *Deprecated

-debug	Compile the code with java debug information
-keep	Do not delete the contents of the working directory
-ignoreErrors	Do not halt for compilation or validation errors
-quiet	Only display errors, suppress informational messages
-nowarn	Disable warning and informational messages
-noinform	Disable informational messages
-rmic "options"	Set additional options to use for RMIC
-trace	Trace progress of the deploy tool
-sqlj	Use SQLJ instead of JDBC
-OCCColumn	Add a column for collision detection for WebSphere 6.0 or later release
-outer	Use OUTER semantics for path expressions in EQL queries for J2EE 1.3 applications
-complianceLevel	JDK level for compiler compliance

For a complete description of the `EJBDeploy` command and its parameters, see the Information Center. Search for *ejbdeploy*.

Example 14-5 shows a sample `EJBDeploy` run using the Plants by WebSphere EAR file.

Example 14-5 EJBDeploy sample run

```
C:\WebSphere\AppServer\bin>ejbdeploy c:\PlantsByWebSphereEnhanced.ear c:\temp
c:\PlantsByWebSphereEnhanced_Deployed.ear -dbvendor DB2UDB_V82
Starting workbench.
framework search path: c:\WebSphere\AppServer\deploytool\itp\plugins
Creating the project.
Deploying jar PlantsByWebSphereEJB
Validating
Generating deployment code
Generating DDL
Generating DDL
...
...
Invoking RMIC.
Writing output file
Shutting down workbench.
EJBDeploy complete.
0 Errors, 62 Warnings, 0 Informational Messages
```

Not shown in the listing above are the warnings that the validators spot. These warnings are things like if a local variable has been declared but is not used or if a Serializable class does not specify a static final long serialVersionUID. These warnings are not critical.

Tip: WebSphere Application Server also provides a set of Ant tasks that you can use to automate the packaging and deployment of your applications. One of those tasks allows you to call EJBDeploy. Search for *Ant tasks* in the Information Center for more details.

14.3 Deploying the application

In this section, we show the steps required to deploy the application to WebSphere Application Server. We show you how to deploy a regular EAR file as well as an Enhanced EAR file, and then also how to not honor the configuration information packaged into the Enhanced EAR file.

Follow these steps to deploy the application:

1. Select **Applications** → **Install New Application** from the administrative console navigation bar.
2. Check the **Local file system** box and click the **Browse** button to locate the `PlantsbyWebSphere.ear` file.

From the install windows, you can install files that are located either on the same machine as the browser you are using to access the WebSphere administrative console, the local file system option, or on the WebSphere Application Server itself, the remote file system option. If you select the Local file system option, the administrative console automatically uploads the file you select to the application server, or to the deployment manager if this is a distributed server environment. If you select the Remote file system check box, you can browse all the nodes in the cell to find the file. The file is then, if necessary, uploaded to the application server or deployment manager.

New in V6.1: WebSphere Application Server V6.1 allows you to take a shortcut when installing an application. If you select the **Prompt me only when additional information is required** option, only the windows where you actually need to fill out some information during installation are shown.

For this example, however, we will explain the options, so select **Show me all installation options and parameters**. Then click **Next**.

3. In the next window, specify default bindings for the application you are deploying. Unless you check the **Override** option, bindings already specified

in the EAR are not altered. The various bindings you can specify in this page are documented in Table 14-1 on page 914. If you do choose to override bindings, select **Generate Default Bindings** at the top of this window to apply changes to the application you are deploying.

In this example, the bindings were set in the application EAR file using the Application Server Toolkit and there is no need to override them. The defaults are also correct.

Table 14-1 Application default bindings

Binding name	Detailed information
EJB prefix	You can generate default EJB JNDI names using a common prefix. EJBs for which you did not specify a JNDI name will get a default name, built by concatenating the prefix and the EJB name. If you specify a prefix of myApp/ejb, then JNDI names default to myApp/ejb/EJBName, such as myApp/ejb/Account.
Override	Enter whether you want to override the current bindings. By default, existing bindings are not altered.
EJB 1.1 CMP bindings	You can bind all EJB 1.1 CMP entity beans to a specific data source, including user ID and password.
Connection Factory bindings	You can bind all EJB modules to a specific data source. You will have to go to the next window to override this setting at the EJB level.
Virtual host bindings	You can bind all Web modules to a specific virtual host, such as plantsbywebsphere_host.
Specify bindings file	You can also create a specific bindings file using your favorite editor and load it during application installation by clicking Browse next to the specific bindings file. For information about using a bindings file, see 14.3.1, "Using a bindings file" on page 919.

4. Click **Next**.

The rest of the wizard is divided into steps. The number of steps depends on your application, for example, if it contains EJB modules or Web modules, you will see windows prompting for the information necessary to deploy them.

5. Step 1: Select installation options.

Step 1 gives you a chance to review the installation options. You can specify various deployment options, such as JSP precompiling, and whether you want to generate EJB deployment code.

If you are deploying an Enhanced EAR file, this is where you make the decision whether to use the resource configuration information packaged in

the Enhanced EAR file or not. If the EAR file you are installing is an Enhanced EAR, the install window preselects the **Process embedded configuration** check box. If you do not want to use the resource configuration information packaged in the Enhanced EAR file, you must deselect this check box.

Selecting the Pre-compile JSP option makes WebSphere compile all JSPs in the EAR file during install time. This causes the time-consuming task of JSP compilation to be performed during install time instead of during run time, preventing the first user that accesses the application to pay that penalty.

A second alternative to pre-compile JSPs is to use the JspBatchCompiler script found in the bin directory of the profile you are using to compile the JSPs after the application has been installed.

This page also allows you to specify file permissions for files in your application. To use one of the predefined file permissions, select it, and then click **Set file permissions**. You can also specify your own file permissions using regular expressions.

New in V6.1: The administrative console displays the Application Build ID of the application being installed. This string is specified in the MANIFEST.MF file in the EAR file's META-INF folder and can be set using the Application Server Toolkit.

The following is an example of a version number:

Implementation-Version: Version 1.2.3

New in V6.1: The Remote Request Dispatcher is an extension to the Web container that allows frameworks, servlets, and JSPs to include content from outside of the current executing resource's JVM as part of the response sent to the client.

To enable this feature, select the corresponding check boxes to allow dispatching or servicing includes to/from remote resources. For more information about this feature, search the InfoCenter for Remote request dispatcher.

Click **Next**.

6. Step 2: Map modules to servers.

Select the server on which you want each module deployed. For better performance, we recommend that you deploy all modules from one application in a single server. Especially, do not separate the EJB clients, usually servlets in Web modules, from the EJBs themselves.

Click the  icon to select all modules in the Plants by WebSphere EAR file. In the Clusters and Servers box, select **PlantsByWebSphereServer**. Then click **Apply**. This assigns all modules to the PlantsByWebSphereServer application server. If you deploy to a cluster, select the cluster instead of the single application server.

See Figure 14-9 on page 916.

Web servers: If you have a Web server defined, select both the Web server and WebSphereBankServer in the server list. Press and hold the CTRL key to select multiple servers. Mapping Web modules to Web servers ensures the Web server plug-in will be generated properly.

Map modules to servers

Specify targets such as application servers or clusters of application servers where you want to install the modules that are contained in your application. Modules can be installed on the same application server or dispersed among several application servers. Also, specify the Web servers as targets that serve as routers for requests to this application. The plug-in configuration file (plugin-cfg.xml) for each Web server is generated, based on the applications that are routed through.

Clusters and Servers:

WebSphere:cell=kcgg1d8Cell,node=kcgg1d8Node,server=server1
WebSphere:cell=kcgg1d8Cell,node=kcgg1d8Node,server=PlantsByWebSphereServer
WebSphere:cell=kcgg1d8Cell,node=kcgg1d8Node,server=webserver1

Apply

Select	Module	URI	Server
☐	PlantsByWebSphere EJB Module	PlantsByWebSphereEJB.jar,META-INF/ejb-jar.xml	WebSphere:cell=kcgg1d8Cell,node=kcgg1d8Node,server=PlantsByWebSphereServer
☐	PlantsByWebSphereWeb	PlantsByWebSphereWeb.war,WEB-INF/web.xml	WebSphere:cell=kcgg1d8Cell,node=kcgg1d8Node,server=PlantsByWebSphereServer

Figure 14-9 Mapping modules to application servers

Note: Steps 3 - 11 allow you to define bindings. We have already taken care of this using the Application Server Toolkit when we packaged the EAR file. You can skip directly to Step 11 if you like. See 15 on page 918.

7. Step 3: Select current back-end ID.

A single EAR file can contain multiple database mappings. At deployment time, you can choose which one you want to use. In this case, set it to DB2UDBNT_V82_1, because this is the version we are using.
Click **Next**.

8. Step 4: Provide JSP reloading options for Web modules.

Allows you to configure if and how often WebSphere should check for updates to JSP files, and if they should be reloaded or not. In a production environment, you may want to disable this to improve performance.
Click **Next**.

916 WebSphere Application Server V6.1: System Management and Configuration

9. Step 5: Map shared libraries.

If your application depends on shared libraries, you can specify them here. For more information about using shared libraries, see “Shared libraries” on page 809.

Click **Next**.

10. Step 6: Provide JNDI names for beans.

Use this window to bind the enterprise beans in your application or module to a JNDI name. In 13.3.1, “Defining EJB JNDI names” on page 842, we defined these values in Table 13-1, so the defaults shown should be correct.

Click **Next**.

11. Step 7: Map EJB references to beans.

Each EJB reference defined in your application must be mapped to an enterprise bean. We used the Application Server Toolkit to do this in 13.3.2, “Binding EJB and resource references” on page 844.

Click **Next**.

12. Step 8: Map default data source mapping for modules containing 2.x entity beans.

Specify the default data source for the EJB 2.x module containing 2.x CMP beans. In 13.3.3, “Defining data sources for entity beans” on page 846, we defined the JNDI name for the EJBs in the PlantsByWebSphere EJB module as eis/jdbc/PlantsByWebSphereDataSource_CMP. You see this in the window.

Click **Next**.

13. Step 9: Map data sources for all 2.x CMP beans.

Specify an optional data source for each 2.x CMP bean. Mapping a specific data source to a CMP bean overrides the default data source for the module containing the enterprise bean (defined in step 8, (12 on page 917). We do not need to do anything here.

Click **Next**.

14. Step 10: Map resource references to resources.

Each resource reference defined in the application must be mapped to the corresponding resource. The PlantsByWebSphere EJB module has several resource references for data sources, which are shown in this window.

Click **Next**.

At this step, we now get an Application Resource Warning. What this tells us is that a resource referenced by the application, mail/PlantsByWebSphere, is not defined for the scope to which we are installing our application. In fact, it is

not defined at all in our environment because we will not send any e-mails from the Plants by WebSphere. If you would like to do that, you should define a Mail session with the properties for your mail server and assign it a JNDI name of mail/PlantsByWebSphere.

Click **Continue**.

15. Step 11: Map virtual hosts for Web modules.

For each Web module, select the virtual host we created for the application (plantsbywebsphere_host).

Click **Next**.

16. Step 12: Map context roots for Web modules.

For each Web module, select the context root to bind the module against.

Click **Next**.

17. Step 13: Map security roles to users and groups.

Because the Plants by WebSphere EAR file contains security roles, we need to map them to users and groups in our target environment. However, because we have not enabled application security, WebSphere will not authenticate users trying to access the application. As a result, we do not need to map the roles to users and groups.

Click **Next**.

18. Step 14: Ensure all unprotected 2.x methods have the correct level of protection.

By default, EJB methods are unprotected. On this window, you can elect to refuse all calls to unprotected methods, or specify which methods you want to exclude.

Again, because we have not enabled J2EE security, WebSphere will not authenticate users trying to access the EJBs.

Click **Next**.

19. Step 15: Summary.

The Summary window gives an overview of application deployment settings. If those settings are fine, click **Finish** to deploy the application.

20. Save the configuration.

If you are working in a distributed server environment, make sure you synchronize the changes with the nodes so the application is propagated to the target application server (s).

21. If you mapped the Web modules to a Web server, make sure the Web server plug-in is regenerated and propagated to the Web server. For a quick refresh, restart the Web server.

Deployment is now complete. You can now launch the Plants by WebSphere application by pointing your browser to:

<http://www.plantsbywebsphere.com:9081/PlantsByWebSphere>

Make sure that the host name is one of the names you added to the `plantsbywebsphere_host` definition. See 14.1.5, “Creating the virtual host for IBM HTTP Server and Apache” on page 902.

Because the DB2 PLANTS database is now empty, it must be populated with data before the application will work. An administrative servlet is supplied for that purpose.

1. Click the **HELP** link in the upper right corner of the page, or go directly to <http://www.plantsbywebsphere.com:9081/PlantsByWebSphere/help.jsp>.
2. Select the **Logging** option and click **Save Setting**. This enables log messages in the application.
3. Click the **(Re)-populate database** link. This will load images and product descriptions from the EAR file and store them in the database.

After the database has been populated, you can test the application.

14.3.1 Using a bindings file

If generating default bindings during deployment, default names suitable for most applications are used. However, these defaults do not work if:

- ▶ You want to explicitly control the global JNDI names of one or more EJB files.
- ▶ You need tighter control of data source bindings for container-managed persistence (CMP) beans. That is, you have multiple data sources and need more than one global data source.
- ▶ You must map resource references to global resource JNDI names that are different from the `java:comp/env` name.

In such cases, you can use a specific bindings file to customize the bindings created.

To use a bindings file when installing an application, load it by clicking **Browse** next to the Specific bindings file option. When using this file, only specify bindings that differ from the defaults, not the full bindings.

Example 14-6 is an example showing a bindings file used to change the JNDI name of an EJB.

Example 14-6 Using a bindings file to change the JNDI of an EJB

```
<?xml version="1.0"?>
<!DOCTYPE dfldbndngs SYSTEM "dfldbndngs.dtd">
<dfldbndngs>
  <module-bindings>
    <ejb-jar-binding>
      <jar-name>helloEjb.jar</jar-name>
      <!-- this name must match the module name in the .ear file -->
      <ejb-bindings>
        <ejb-binding>
          <ejb-name>HelloEjb</ejb-name>
          <!-- this must match the <ejb-name> entry in the EJB jar DD -->
          <jndi-name>com/acme/ejb/HelloHome</jndi-name>
        </ejb-binding>
      </ejb-bindings>
    </ejb-jar-binding>
  </module-bindings>
</dfldbndngs>
```

14.4 Deploying application clients

To run a Java-based client/server application, the client application executes in a client container of some kind. You might, for example, use a graphical Swing application that calls EJBs on an application server. WebSphere Application Server V6 supports the following five types of application client environments:

- ▶ J2EE application client

This client uses services provided by the J2EE client container.

This client is a Java application program that accesses EJBs, JDBC databases, and JMS queues. The J2EE application client program runs on client machines. This program allows the same Java programming model as other Java programs. However, the J2EE application client depends on the application client run time to configure its execution environment, and it uses the JNDI name space to access resources, the same as you would in a normal server application (like a servlet).

The J2EE application client brings the J2EE programming model to the client, and provides:

- XML deployment descriptors

- J2EE naming (java:comp/env), including EJB references and resource references

The J2EE application client is launched using the launchClient script, which sets up the environment with the necessary classpaths, and so on, for you.

- ▶ Thin application client

This client does not use services provided by the J2EE client container.

This client provides a lightweight Java client programming model and is best suited for use in situations where a Java client application exists, but the application must be enhanced to make use of EJBs. It can also be used where the client application requires a thinner, more lightweight environment than the one offered by the J2EE application client. The Thin application client includes the IBM JDK. When launching the Thin application client, you must set up the correct classpaths yourself and make sure that the required libraries for your application and the WebSphere libraries are included.

- ▶ Pluggable application client

This client does not use services provided by the J2EE Client Container.

This client is similar to the Thin application client, but does not include a JVM. The user is required to provide a JVM. It can also use the Sun JDK instead of the IBM JDK.

- ▶ Applet application client

In the Applet client model, a Java applet embedded in an HTML document executes in a Web browser. With this type of client, the user accesses an enterprise bean in the application server through the Java applet in the HTML document.

- ▶ ActiveX® to EJB Bridge application client

The ActiveX application client allows ActiveX programs to access enterprise beans through a set of ActiveX automation objects. The ActiveX application client uses the Java Native Interface (JNI) architecture to programmatically access the Java virtual machine (JVM) API. Therefore, the JVM code exists in the same process space as the ActiveX application (Visual Basic®, VBScript, or Active Server Pages files) and remains attached to the process until that process terminates.

The capabilities of the different application clients are shown in Table 14-2.

Table 14-2 Application client features comparison

Available functions	J2EE client	Thin client	Pluggable client	Applet client	ActiveX client
Provides all the benefits of a J2EE platform.	Yes	No	No	No	Yes
Portable across all J2EE platforms.	Yes	No	No	No	No
Provides the necessary run time support for communication between a client and a server.	Yes	Yes	Yes	Yes	Yes
Supports the use of nicknames in the deployment descriptor files.	Yes	No	No	No	Yes
Supports use of the RMI-IIOP protocol.	Yes	Yes	Yes	Yes	Yes
Browser-based application.	No	No	No	Yes	No
Enables development of client applications that can access enterprise bean references and CORBA object references.	Yes	Yes	Yes	Yes	Yes
Enables the initialization of the client application run time environment.	Yes	No	No	No	Yes
Supports security authentication to enterprise beans.	Yes	Yes	Yes	Limited	Yes
Supports security authentication to local resources.	Yes	No	No	No	Yes
Requires distribution of application to client machines.	Yes	Yes	Yes	No	Yes

Available functions	J2EE client	Thin client	Pluggable client	Applet client	ActiveX client
Enables access to enterprise beans and other Java classes through Visual Basic, VBScript, and Active Server Pages (ASP) code.	No	No	No	No	Yes
Provides a lightweight client suitable for download.	No	Yes	Yes	Yes	No
Enables access JNDI APIs for enterprise bean resolution.	Yes	Yes	Yes	Yes	Yes
Runs on client machines that use the Sun Java Runtime Environment.	No	No	Yes	No	No
Supports CORBA services (using CORBA services can render the application client code nonportable).	Yes	No	No	No	No

Install the application client environments from the WebSphere installation windows by selecting the **Launch the installation wizard for WebSphere Application Clients** option. The installation package contains the following installable components:

- ▶ IBM Java Runtime Environment (JRE™), or an optional full Software Development Kit
- ▶ WebSphere Application Server run time for J2EE application client applications, or Thin application client applications
- ▶ An ActiveX to EJB Bridge run time for ActiveX to EJB Bridge application client applications (only for Windows)
- ▶ IBM plug-in for Java platforms for Applet client applications (Windows only)

Note: The J2EE client is automatically installed as part of a full WebSphere install. In other words, if you will run the client application on a machine that already has WebSphere installed, you do not need to install the WebSphere J2EE client on top.

14.4.1 Defining application client bindings

The Plants by WebSphere sample application does not provide any client application. For the purpose of discussing the client container, we will therefore use the WebSphere Bank sample application that was shipped with WebSphere Application Server V6.0 but has been removed in V6.1.

WebSphere Bank provides four client applications: the GetAccounts, FindAccounts, TransferWS and TransferJMS clients. The various client applications demonstrate the capabilities of the WebSphere Bank sample application.

For an application client to be able to access resources, such as EJBs provided by a J2EE server application, the proper bindings must be set up. You need to specify the complete naming structure to reach the server where the EJBs are deployed. For example, the machine where we deployed the application for testing has the Network Deployment version installed. The WebSphere Bank application is running in the WebSphereBank application server on node ITSONode1.

Therefore, the ejb/Bank/Customer EJB reference can be bound to:

cell/nodes/ITSONode1/servers/WebSphereBankServer/ejb/Bank/Customer

If you have created a cluster of application servers, use:

cell/clusters/<clusterName>/ejb/Bank/Customer

You will also need to change the provider URL, according to the target server. If you are running in a single server environment you can simply use:

ejb/Bank/Customer

When you have configured the proper bindings for the resources, you must export the EAR file and copy it to the client machine. Although you do not need the complete contents of the EAR file to run the application client, for example, the Web modules, it is better to keep a single EAR file. This is mainly for maintenance purposes.

14.4.2 Launching the J2EE client

A J2EE client application needs a container to run in. In this example, we will use the J2EE application client container. This container can be started using the `launchClient` program in the `<<was_home>>/bin` directory. The `launchClient` program has the following syntax:

```
Usage: launchClient [-profileName pName | -JVMOptions options | -help | -?]  
<userapp> [-CC<name>=<value>] [app args]
```

The elements of syntax are:

-profileName	This option defines the profile of the Application Server process in a multi-profile installation. The <code>-profileName</code> option is not required for running in a single profile environment or in an Application Clients installation. The default is <code>default_profile</code> .
-JVMOptions	This is a valid Java standard or nonstandard option string. Insert quotation marks around the option string.
-help, -?	Print the usage information.
<userapp.ear>	Type the path/name of the <code>.ear</code> file containing the client application.

The `-CC` properties are for use by the Application Client run time. There are numerous parameters available and because of this we only describe the more commonly used ones. For full explanation of all parameters, execute `launchClient -help`.

-CCverbose	Use this option with <code><true false></code> to display additional informational messages. The default is <code>false</code> .
-CCclasspath	This property is a classpath value. When an application is launched, the system classpath is not used. If you need to access classes that are not in the EAR file or part of the resource classpaths, specify the appropriate classpath here. Multiple paths can be concatenated.
-CCjar	This is the name of the client JAR file within the EAR file that contains the application you want to launch. This argument is only necessary when you have multiple client JAR files in the EAR file.
-CCBootstrapHost	This option is the name of the host server you want to connect to initially. The format is <code>your.server.ofchoice.com</code> .
-CCBootstrapPort	This option is the server port number. If not specified, the WebSphere default value (2809) is used.

- CCproviderURL** This option provides bootstrap server information that the initial context factory can use to obtain an initial context. WebSphere Application Server initial context factory can use either a CORBA object URL or an IIOP URL. CORBA object URLs are more flexible than IIOP URLs and are the recommended URL format to use. This value can contain more than one bootstrap server address. This feature can be used when attempting to obtain an initial context from a server cluster. In the URL, you can specify bootstrap server addresses for all servers in the cluster. The operation will succeed if at least one of the servers is running, eliminating a single point of failure. The address list does not process in a particular order. For naming operations, this value overrides the -CCBootstrapHost and -CCBootstrapPort parameters. An example of a CORBA object URL specifying multiple systems is:
-CCproviderURL=corbaloc:iiop:myserver.mycompany.com:9810,:mybackupserver.mycompany.com:2809
- CCtrace** Use this option with <true|false> to have WebSphere write debug trace information to a file. The value true is equivalent to a trace string value of com.*=all=enabled. Instead of the value true you can specify a trace string, for example, -CCtrace=com.ibm.ws.client.*=all=enabled. Multiple trace strings can be specified by separating them with a colon (:). You might need this information when reporting a problem to IBM Service. The default is false.
- CCtracefile** This option is the name of the file to which to write trace information. The default is to output to the console.
- CCpropfile** This option is the name of a properties file containing launchClient properties. In the file, specify the properties without the -CC prefix. For example: verbose=true.

The app args are for use by the client application and are ignored by WebSphere.

To start the WebSphere Bank GetAccounts client using the launchClient command, execute the command shown in Figure 14-7.

Example 14-7 Launching WebSphere Bank application client

```
C:\WebSphere\AppServer\profiles\AppSrv01\bin>launchClient.bat  
c:\WebSphereBankClient.ear -CCBootstrapPort=2809 -CCjar=GetAccounts.jar 100
```

```
IBM WebSphere Application Server, Release 6.1  
J2EE Application Client Tool  
Copyright IBM Corp., 1997-2006  
WSCL0012I: Processing command line arguments.  
WSCL0013I: Initializing the J2EE Application Client Environment.  
WSCL0035I: Initialization of the J2EE Application Client Environment has  
completed.  
WSCL0014I: Invoking the Application Client class  
com.ibm.websphere.samples.bank.client.GetAccounts  
Get the account numbers owned by a certain customer  
  
Getting the customer home...  
  
Done....  
  
Finding the customer....  
Done....  
Account Number: 101  
All done!  
  
Finding the customer from invoke ....  
Account Number: 101  
All done!  
  
Get accounts owned by customer from invoke...  
Account Number: 101  
Account balance: 200.0  
Get number of customer accounts from invoke...  
Account :1  
All done!
```

Because the WebSphereBank EAR file contains multiple client applications (JAR files), the -CCjar option must be used to specify which client application to launch.

The WebSphere Bank GetAccounts client application is a text-mode application that simply displays the accounts for a certain customer number 100 in our example.

14.5 Updating applications

WebSphere Application Server has features that allow applications to be updated and restarted at a fine-grained level. It is possible to update only parts of an application or module and only the necessary parts are restarted. You can:

- ▶ Replace an entire application (.ear file).
- ▶ Replace, add, or remove a single module (.war, EJB .jar, or connector .rar file).
- ▶ Replace, add, or remove a single file.
- ▶ Replace, add and remove multiple files by uploading a compressed file describing the actions to take.

If the application is running while being updated, WebSphere Application Server automatically stops the application, or only its affected components, updates the application, and restarts the application or components.

When updating an application, only the portion of the application code that changed needs to be presented to the system. The application management logic calculates the minimum actions that the system needs to execute in order to update the application. Under certain circumstances, the update can occur without stopping any portion of the running application.

WebSphere Application Server also has support for managing applications in a cluster for continuous availability. The action, Rollout Update, sequentially updates an application installed on multiple cluster members across a cluster. After you update an application's files or configuration, use the Rollout Update option to install the application's updated files or configuration on all cluster members of a cluster on which the application is installed.

Rollout Update does the following for each cluster member in sequence:

1. Saves the updated application configuration.
2. Stops all cluster members on a given node.
3. Updates the application on the node by synchronizing the configuration.
4. Restarts the stopped cluster members on that node.

This action updates an application on multiple cluster members while providing continuous availability of the application.

14.5.1 Replacing an entire application EAR file

To replace a full EAR with a newer version, do the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to update and click the **Update** button.

2. On the Preparing for the application installation window, select the **Replace the entire application** option.
3. Select either the **Local file system** or **Remote file system** option. Click the **Browse** button to select the updated EAR file. Click **Next**.
4. Proceed through the remaining windows and make any changes necessary. For information about the windows, see “Deploying the application” on page 913. On the Summary window, click **Finish**.
5. When the application has been updated in the Master repository, select the **Save** link.
6. If you are working in a distributed server environment, make sure that you also synchronize the changes with the nodes.
7. If the application update changes the set of URLs handled by the application (servlet mappings added, removed, or modified), make sure the Web server plug-in is regenerated and propagated to the Web server.

Note: It may take a few seconds for the WebSphere run time to pick up the changes and restart the application as necessary. If your changes do not seem to have effect, wait and try again. You can also look at the SystemOut.log file for the application server to see when it has restarted the application.

14.5.2 Replacing or adding an application module

To replace only a module, such as an EJB or Web module of an application, do the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to update and click the **Update** button.
2. On the Preparing for the application installation window, select the **Replace or add a single module** option.
3. In the Specify the path beginning with the installed application archive file... field, enter the relative path to the module to replace. For example, if you were to replace the HelloWorld module, enter HelloWorld. If you enter a path or file that does not exist in the EAR file, it will be added.
4. Select either the **Local file system** or **Remote file system** option and click the **Browse** button to select the updated module.
5. For Web modules, also enter the context root (for example, HelloWorld) in the Context root field.
6. Click **Next**.

7. Proceed through the remaining windows and make any necessary changes. For information about the windows, see “Deploying the application” on page 913. On the Summary window, click **Finish**.

Note: If you are adding a module, make sure to select the correct target server for the module in the Map modules to servers step.

8. When the application has been updated in the Master repository, select the **Save** link.
9. If you are working in a distributed server environment, make sure that you also synchronize the changes with the nodes.
10. If the application update changes the set of URLs handled by the application (servlet mappings added, removed, or modified), make sure the Web server plug-in is regenerated and propagated to the Web server.

Note: Modules can also be managed using the Manage Modules page. Select **Applications** → **Enterprise Applications** and click the link for the application. Then click the **Manage Modules** link in the Modules section. Select the module to modify and then click the **Remove**, **Update**, or **Remove File** buttons.

14.5.3 Replacing or adding single files in an application or module

To replace a single file, such as a GIF image or a properties file in an application or module, do the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to update and click the **Update** button.
2. On the Preparing for the application installation window, select the **Replace or add a single file** option.
3. In the **Relative path to file** field, enter the relative path to the file to replace in the EAR file. For example, if you were to replace the logo.gif in the images directory of the HelloWorld.war Web module, you would enter HelloWorld.war/images/logo.gif. If you enter a path or file that does not exist in the EAR file, it will be added.
4. Select either the **Local file system** or **Remote file system** option and click the **Browse** button to locate the updated file. Click **Next**.
5. On the Updating Application window, click **OK**.
6. When the application has been updated in the Master repository, select the **Save** link.

7. If you are working in a distributed server environment, make sure that you also synchronize the changes with the nodes.

14.5.4 Removing application content

Files can also easily be removed either from an EAR file or from a module in an EAR file.

Removing files from an EAR file

To remove a file from an EAR file, do the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to remove the file from and click the **Remove File** button.
2. In the Remove file dialog box, select the file to be removed and click **OK**.
3. Save the configuration.

Removing files from a module

To remove a file from a module, do the following:

1. Select **Applications** → **Enterprise Applications** and click the link for the application to which the module belongs.
2. Click the **Manage Modules** link under the Modules section.
3. Select the module to remove the file from and click the **Remove File** button.
4. In the Remove a file from a module dialog, select the file to be removed and click **OK**.
5. Save the configuration.

14.5.5 Performing multiple updates to an application or module

Multiple updates to an application and its modules can be packaged in a compressed file, .zip, or .gzip format, and uploaded to WebSphere Application Server. The uploaded file is analyzed and the necessary actions to update the application are taken.

Depending on the contents of the compressed file, this method to update an application can replace files in, add new files to, and delete files from the installed application all in one single administrative action. Each entry in the compressed file is treated as a single file, and the path of the file from the root of the compressed file is treated as the relative path of the file in the installed application.

- To replace a file, a file in the compressed file must have the same relative path as the file to be updated in the installed application.

- ▶ To add a new file to the installed application, a file in the compressed file must have a different relative path than the files in the installed application.
- ▶ To remove a file from the installed application, specify metadata in the compressed file using a file named META-INF/ibm-partialapp-delete.props at any archive scope. The ibm-partialapp-delete.props file must be an ASCII file that lists files to be deleted in that archive with one entry for each line. The entry can contain a string pattern, such as a regular expression that identifies multiple files. The file paths for the files to be deleted must be relative to the archive path that has the META-INF/ibm-partialapp-delete.props file.
- ▶ To delete a file from the EAR file (not a module), include a META-INF/ibm-partialapp-delete.props file in the root of the compressed file. In the .props file, list the files to be deleted. File paths are relative to the root of the EAR file.

For example, to delete a file named docs/readme.txt from the root of the HelloApp.ear file, include the line docs/readme.txt in the META-INF/ibm-partialapp-delete.props file in the compressed file.

- ▶ To delete a file from a module in the EAR, include a module_uri/META-INF/ibm-partialapp-delete.props file in the compressed file. The module_uri part is the name of the module, such as HelloWeb.war.

For example, to delete images/logo.gif from the HelloWeb.war module, include the line images/logo.gif in the HelloWeb.war/META-INF/ibm-partialapp-delete.props file in the compressed file.

- ▶ Multiple files can be deleted by specifying each file on its own line in the metadata .props file.

Regular expressions can also be used to target multiple files. For example, to delete all JavaServer Pages (.jsp files) from the HelloWeb.war file, include the line `.*jsp` in the HelloWeb.war/META-INF/ibm-partialapp-delete.props file. The line uses a regular expression, `.*jsp`, to identify all .jsp files in the HelloWeb.war module.

As an example, assume we have prepared the compressed HelloApp_update.zip file shown in Figure 14-10.

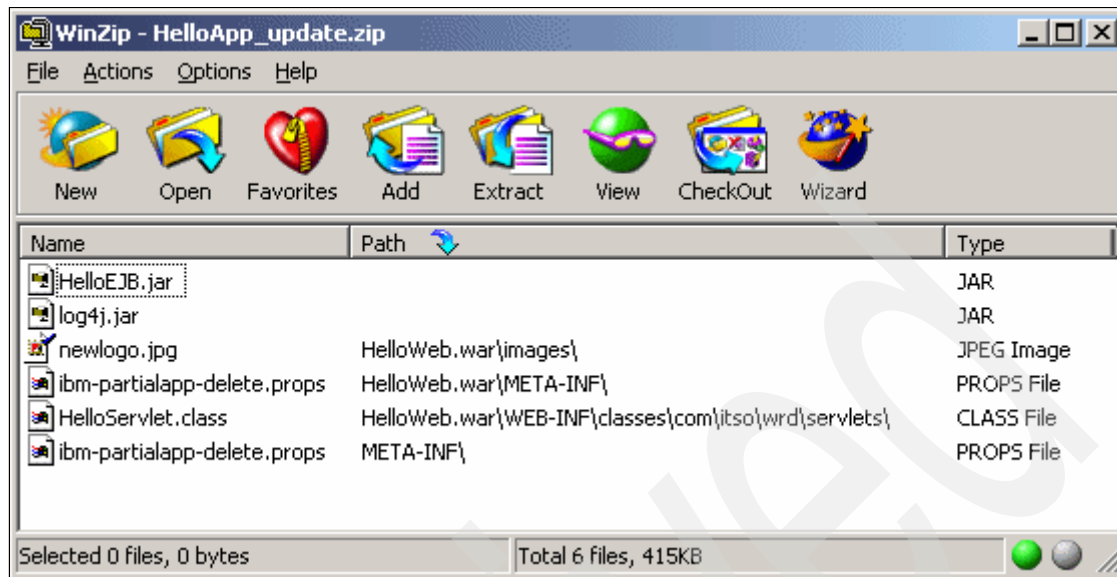


Figure 14-10 HelloApp_update.zip compressed file

The META-INF/ibm-partialapp-delete.props file contains the following line:

docs/readme.txt

The HelloWeb.war/META-INF/ibm-partialapp-delete.props contains the following lines:

images/logo.gif

When performing the partial application update using the compressed file, WebSphere does the following:

- ▶ Adds the log4j.jar file to the root of the EAR.
- ▶ Updates the entire HelloEJB.jar module.
- ▶ Deletes the docs/readme.txt file (if it exists) from the EAR file, but not from any modules.
- ▶ Adds the images/newlogo.jpg file to the HelloWeb.war module.
- ▶ Updates the HelloServlet.class file in the WEB-INF/classes/com/itso/wrd/servlets directory of the HelloWeb.war module.
- ▶ Deletes the images/logo.gif file from the HelloWeb.war module.

To perform the actions specified in the HelloWorld_updated.zip file, do the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to update and click the **Update** button.
2. On the Preparing for the application installation window, select the **Partial Application** option.
3. Select either the **Local file system** or **Remote file system** option and click the **Browse** button to select the compressed ZIP file with the modifications you have created. Click **Next**.
4. On the Updating Application window, click **OK**.
5. When the application has been updated in the Master repository, select the **Save To Master Configuration** link.
6. If in a distributed server environment, make sure the **Synchronize changes with Nodes** option is selected so that the application is distributed to all nodes. Click the **Save** button. The application is distributed to the nodes, updated, and restarted as necessary.
7. If the application update changes the set of URLs handled by the application (servlet mappings added, removed or modified), make sure the Web server plug-in is regenerated and propagated to the Web server.

14.5.6 Rolling out application updates to a cluster

The new Rollout Update feature allows you to easily roll out a new version of an application, or part of an application using the techniques described previously, to a cluster. The Rollout Update feature takes care of stopping the cluster members, distributing the new application, synchronizing the configuration, and restarting the cluster members. The operation is done sequentially over all cluster members in order to keep the application continuously available.

When stopping and starting the cluster members, the Rollout Update feature works on node level, so all cluster members on a node are stopped, updated, and then restarted, before the process continues to the next node.

Because the Web server plug-in module is not able to detect that an individual application on an application server is unavailable, the Rollout Update feature always restarts the whole application server hosting the application. Because of this, if HTTP session data is critical to your application, it should either be persisted to database or replicated to other cluster members using the memory-to-memory replication feature.

The order in which the nodes are processed and the cluster members are restarted is the order in which they are read from the cell configuration repository.

There is no way to tell the Rollout Update feature to process the nodes and cluster members in any particular order.

Assume we have an environment with two nodes, ITSONode1 and ITSONode2, and a cluster called HelloCluster, which has one cluster member on each node (HelloServer1 on ITSONode1 and HelloServer2 on ITSONode2). Assume we have an application called HelloApp deployed and running on the cluster. To update this application using the Rollout Update feature we would do the following:

1. Select **Applications** → **Enterprise Applications**. Select the application to update and click the **Update** button.
2. On the Preparing for the application installation window, select the appropriate action depending on the type of update. In this example, we will update the entire application EAR to a new version, so we select the **Replace the entire application** option.
3. Select either the **Local file system** or **Remote file system** option and click the **Browse** button to select the updated EAR file. Click **Next**.
4. Proceed through the remaining windows and make any changes necessary. For information about the windows, see “Deploying the application” on page 913. On the Summary window, click **Finish**.

5. When the application has been updated in the master repository, the status window shown in Figure 14-11 on page 936 is displayed.

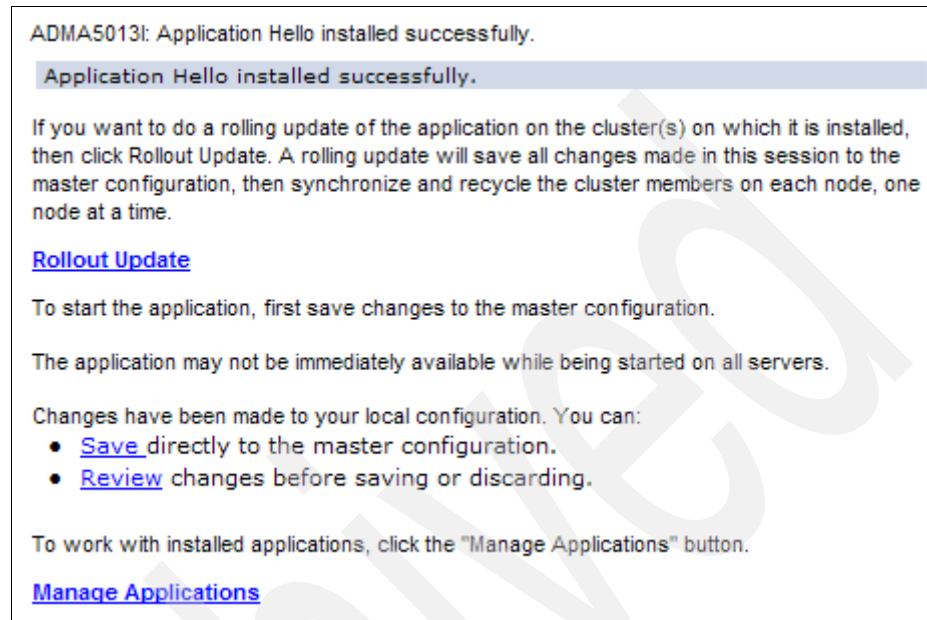


Figure 14-11 Preparing for application rollout

You then have two options to start the rollout action:

- Click the **Rollout Update** link.
- Click the **Manage Applications** link and on the Enterprise Applications window, select the application and click the **Rollout Update** button.

Note: Do not click the Save to Master Configuration link or otherwise save the configuration yourself. The Rollout Update will do that for you. If you save the configuration yourself, the rollout update action will be canceled and it will be handled as a normal application update.

During the rollout, the window in Figure 14-12 on page 937 is displayed in the status window.

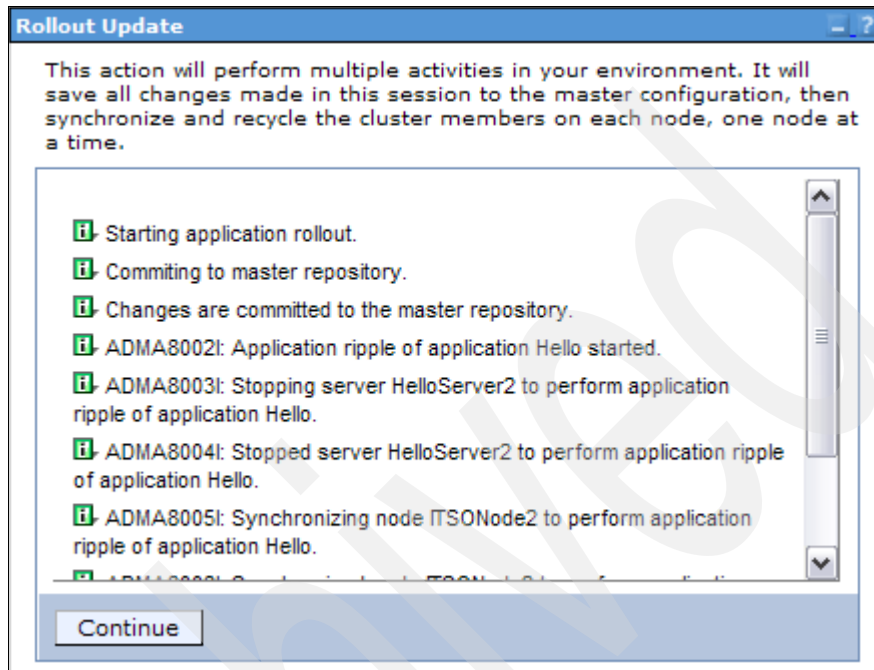


Figure 14-12 Rolling out an application

For each node, the cluster members are stopped, the application is distributed, and they are restarted. When the rollout has completed (the last message says “The application rollout succeeded”, click **Continue**.

6. If the application update changes the set of URLs handled by the application (servlet mappings added, removed, or modified), make sure the Web server plug-in is regenerated and propagated to the Web server.

Note: The automatic file synchronization of the node agent is temporarily disabled during the rollout process and then re-enabled afterwards, if it was previously enabled. The Rollout Update feature works regardless of the automatic file synchronization setting. However, in production systems, the automatic synchronization is often disabled anyway to give the administrator greater control over exactly when changes made to the cell configuration are distributed to the nodes.

Although the Rollout Update feature makes it very easy to roll out an application to a cluster while keeping the application continuously available, make sure that your application can handle the roll out.

For example, assume you have version 1.0 of an application running in a cluster consisting of two application servers, server1 and server2, and that HTTP session data is persisted to a database. When you roll out version 2.0 of the application and server1 is stopped, the Web server plug-in redirects the users on server1 to server2. Then, when server1 is started again, bringing up version 2.0 of the application, the plug-in will start distributing requests to server1 again. Now, if the application update incurred a change in the interface of any class stored in the HTTP session, when server1 tries to get these session objects from the database, it might run into a deserialization or class cast exception, preventing the application from working properly.

Another situation to consider is when the database structure changes between application versions, as when tables or column names change name or content. In that case, the whole application might need to be stopped and the database migrated before the new version can be deployed. The Rollout Update feature would not be suitable in that kind of scenario.

So it is very important to understand the changes made to your application before rolling it out.

14.5.7 Hot deployment and dynamic reloading

Hot deployment and dynamic reloading characterize how application updates are handled when updates to the applications are made by directly manipulating the files on the server. In either case, updates do not require a server restart, though they might require an application restart:

- Hot deployment of new components

Hot deployment of new components is the process of adding new components, such as WAR files, EJB JAR files, EJBs, servlets, and JSP files to a running application server without having to stop and then restart the application server.

However, in most cases, such changes require the application itself to be restarted, so that the application server run time reloads the application and its changes.

► Dynamic reloading of existing components

Dynamic reloading of existing components is the ability to change an existing component without the need to restart the application server for the change to take effect. Dynamic reloading can involve changes to the:

- Implementation of an application component, such as changing the implementation of a servlet
- Settings of the application, such as changing the deployment descriptor for a Web module

To edit the files manually, locate the binaries in use by the server. See “Repository files used for application execution” on page 44. Although the application files can be manually edited on one or more of the nodes, these changes will be overwritten the next time the node synchronizes its configuration with the deployment manager. Therefore, we recommend that manual editing of an application’s files should only be performed in the master repository, located on the deployment manager machine.

Note: Unless you are familiar with updating applications by directly manipulating the server files, it might be better to use the administrative console Update wizard.

There are three settings that affect dynamic reload:

► Reload classes when application files are updated

In order for application files to be reloaded automatically after an update, Reload classes when application files are updated must be enabled and the Polling interval for updated files must be greater than 0.

Select **Applications** → **Enterprise Applications**, and click the link for the application. In the Detail properties section, click the **Class loading and update detection** link.

► Application Server class loader policy

The application server’s class loader policy should be set to Multiple. If it is set to Single, the application server will need to be restarted after an application update.

Select **Servers** → **Application Servers**, and click the **Server** link. The setting is found in the General Properties section.

► JSP Reload options for Web modules

A Web container reloads a Web module only when this setting is enabled.

Select **Applications** → **Enterprise Applications**, and click the link for the application. In the Web Module Properties section, click the **JSP reload**

options for web modules link, and then select the JSP enable class reloading option and enter a polling interval.

For more information about using hot deployment and dynamic reload, see the *Updating applications* and *Hot deployment and dynamic reloading* topics in the Information Center.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this IBM Redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 944. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *IBM WebSphere Application Server V5.1 System Management and Configuration*, SG24-6195
- ▶ *Planning and Designing for WebSphere Application Server V6.1*, SG24-7305
- ▶ *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257
- ▶ *WebSphere Application Server Network Deployment V6: High Availability Solutions*, SG24-6688
- ▶ *WebSphere Application Server V6: High Availability Solutions*, REDP-3971
- ▶ *WebSphere Application Server V6 Migration Guide*, SG24-6369
- ▶ *WebSphere Application Server V6 Problem Determination for Distributed Platforms*, SG24-6798
- ▶ *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392
- ▶ *WebSphere Application Server V6 System Management & Configuration Handbook*, SG24-6451
- ▶ *WebSphere Application Server V6.1 Security Handbook*, SG24-6316
- ▶ *WebSphere Business Integration Server Foundation V5.1 Handbook*, SG24-6318
- ▶ *WebSphere MQ in a z/OS Parallel Sysplex Environment*, SG24-6864

Other publications

These publications are also relevant as further information sources:

- ▶ Giotta, et al, *Professional JMS*, Wrox Press Inc., 2001, ISBN 1861004931
- ▶ Gray, et al, *Transaction Processing: Concepts and Techniques*, Elsevier Science & Technology Books, 1992, ISBN 1558601902
- ▶ Marinescu, et al, *EJB Design Patterns*, Wiley, John & Sons, Incorporated, 2002, ISBN 0471208310
- ▶ Monson-Haefel, et al, *Enterprise JavaBeans, Fourth Edition*, O'Reilly Media, Incorporated, 2004, ISBN 059600530X
- ▶ Monson-Haefel, et al, *Java Message Service*, O'Reilly Media, Incorporated, 2000, ISBN 0596000685
- ▶ Yusuf, *Enterprise Messaging Using JMS and WebSphere*, Pearson Education, 2004, ISBN 0131468634

Online resources

These Web sites are also relevant as further information sources:

- ▶ Enterprise JavaBeans Technology
<http://java.sun.com/products/ejb/>
- ▶ IBM alphaWorks emerging technologies
<http://www.alphaworks.ibm.com>
- ▶ IBM developerWorks
<http://www.ibm.com/developerworks/>
- ▶ IBM HTTP Server documentation library
<http://www.ibm.com/software/webservers/httpservers/library/>
- ▶ IBM WebSphere Developer Technical Journal
<http://www-106.ibm.com/developerworks/websphere/techjournal/>
- ▶ J2EE Connector Architecture
<http://java.sun.com/j2ee/connector/>
- ▶ *Java 2 Platform Enterprise Edition, V1.4 API Specification*, found at:
<http://java.sun.com/j2ee/1.4/docs/api/index.html>
- ▶ *Java 2 Platform Enterprise Edition Specification, V1.4*, found at:
http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf

- ▶ Java Community Process home
<http://www.jcp.org/en/jsr/all>
- ▶ JavaMail API Specification
<http://java.sun.com/products/javamail/reference/api/index.html>
- ▶ Java Message Service (JMS)
<http://java.sun.com/products/jms>
- ▶ JDBC Technology
<http://java.sun.com/products/jdbc/index.html>
- ▶ *MBeanInspector for WebSphere Application Server*, found at:
<http://www.alphaworks.ibm.com/tech/mbeaninspector>
- ▶ *Persistent Client State HTTP Cookies*, found at:
http://home.netscape.com/newsref/std/cookie_spec.html
- ▶ *Sample Scripts for WebSphere Application Server Versions 5 and 6*, found at:
<http://www-106.ibm.com/developerworks/websphere/library/samples/SampleScripts.html>
- ▶ *Service Data Objects*, found at:
<ftp://www6.software.ibm.com/software/developer/library/j-commonj-sdowmt/Commonj-SDO-Specification-v1.0.doc>
- ▶ Tcl Developer Xchange
<http://www.tcl.tk/>
- ▶ WebSphere Application Server home page
<http://www.ibm.com/software/webservers/appserv/was/>
- ▶ WebSphere Application Server Information Center
<http://www.ibm.com/software/webservers/appserv/infocenter.html>
- ▶ WebSphere Application Server support (Fix Packs, fixes, and hints and tips)
<http://www.ibm.com/software/webservers/appserv/support.html>
- ▶ WebSphere Application Server system requirements
<http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html>
- ▶ *WebSphere MQ Using Java*, found at:
<http://www-306.ibm.com/software/integration/mqfamily/library/manuals/manuals/crosslatest.html>

- ▶ *WebSphere z/OS V6 -- WSC Sample ND Configuration*, found at:
<http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100653>
- ▶ *WebSphere for z/OS V6.1 - New Things Encountered During Configuration*, found at:
<http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100781>
- ▶ Worldwide WebSphere User Group
<http://www.websphere.org>
- ▶ XDoclet Attribute Oriented Programming
<http://xdoclet.sourceforge.net/xdoclet/index.html>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Symbols

`${DRIVER_PATH}` 317
`$AdminApp` 258–259
 edit 293
 editInteractive 293
 install 291
 installInteractive 291
 options 291
 uninstall 292
`$AdminConfig` 257, 259, 265–266, 269
 create 288, 294
 getid 288
 modify 288
 parent 267
 remove 291
 save 288, 292
`$AdminControl` 257, 260, 279
 queryNames 260–261
 refreshRepositoryEpoch 36
 stopServer 280
`$AdminTask` 258–259, 265, 273
 createCluster 276
 createSIBus 275
 exportServer 247
 exportWasprofile 248
`$Help` 258

A

access intent 860, 863, 865, 867
 application profile 866
 application profiles 197
 tracing 871
access intent policies 860, 864
Activate at option 857
activation.jar 342
ActivationSpec JavaBean 426, 428, 430, 443, 446
Active Server Pages 921
ActiveX to EJB Bridge application client 921, 923
activity session service 198
addNode 86, 201, 204–206, 208, 210
adjunct process 113
admin_host 227
AdminApp 283–284

AdminConfig 259, 290, 296
AdminControl 259, 262, 284
administration services 164, 199
administrative console
 deploying during profile creation 59, 68, 112
 interrupted session 140
 logging in 140
 preferences 146
 scope 148
 securing 159
 session timeout 142
 starting 138–139
administrative console port 63, 71
administrative console secure port 63, 71
administrative security 59, 62, 68, 70, 75, 78, 140, 159–160
 per resource instance 161
administrator role 161
AdminSecurityManager role 161
AdminServer 130
AdminService 27–28
AdminTask 259, 290, 296, 299
AffinityCookie 386
Aged Timeout 324–325
alias destination 549, 588, 640
ALL_AUTHENTICATED 790
Ant tasks 913
Apache 902
apachectl 377
applet application client 921
application
 deploying 913
 editing with wsadmin 293
 exporting 233
 finding the URL 238
 installation 229, 231, 291
 listing 283
 multiple updates 931
 preventing from starting 234
 preventing startup 293
 removing files 931
 single file update 930
 single module update 930
 starting 179, 234

- starting order 234
- starting with wsadmin 284
- stopping 234
- stopping with wsadmin 284
- uninstalling 233
- uninstalling with wsadmin 292
- updating 928
- viewing 235
- viewing EJB modules 236
- Application Build ID 915
- Application class loader 799–800, 807
- application class loader 796, 798–800, 802–804, 807–809, 814, 816, 820
- application classloader policy 190
- application client 9, 920
 - bindings 924
 - deployment 920
 - launching 925
- application data repository 39
- application module
 - updating 929
- application profiling service 197
- Application Resource Warning 917
- application scoped resources 149
- application security 159
- application server
 - creating 171, 289, 896–897
 - customizing 188
 - logs and trace 898
 - modifying with wsadmin 294
 - removing 290
 - restarting 183
 - run time attributes 185
 - starting 178, 281
 - stopping 181, 183, 282
- Application Server Facilities 421
- application server name 108
- application server profile 48–51, 55–56, 67, 91, 94, 97, 125–126, 139, 170–171, 178, 181, 243, 246, 254, 371
- application server template 224
- Application Server Toolkit 7, 9, 95, 97–98, 832, 838, 840, 846, 890, 895, 911, 914, 916
- application.xml 43, 839
- application-client.xml 839
- applications
 - deployment
 - dynamic reload 873
 - hot deployment 873

- managing 230
- ARM 134
- asynchronous beans 305
- asynchronous messaging 401–402
- attributes 263, 266, 269
- authentication
 - component managed 361
 - component-managed 319, 339
 - container 362
 - container-managed 319
 - resource 361
- authentication alias 629, 655, 894
- auto reload 873
- automatic file synchronization 937
- Automatic Request Encoding enabled option 875
- Automatic Response Encoding enabled option 875
- automatic synchronization 33
- autoRequestEncoding 875
- autoResponseEncoding 875
- averaging period 190

B

- back-end ID 848, 853, 916
- backupConfig 243–244
- BBOCCINS 168
- bean managed activity session 738
- bean managed transaction 738–739
- bean managed transactions 442
- Bean Scripting Framework (BSF) 250
- binding 842, 847
 - application client bindings 924
 - compound name 746
 - configured 742
 - corbaname 746
 - CorbaObjectNameSpaceBinding 762
 - data sources 846
 - EJB JNDI names 842
 - EJB references 844
 - EjbNameSpaceBinding 762
 - IndirectLookupNameSpaceBinding 762
 - name 743
 - overriding defaults 913
 - simple name 745
 - StringNameSpaceBinding 762
- bindings connection 491–492
- bindings file 919
- BLOB 850, 867, 895
- BMP 861, 863

- boot class path 163
- bootstrap 757, 763–764, 766–767, 773–774, 776, 778, 782, 796
- bootstrap class loader 796, 799
- bootstrap endpoint 655
- bootstrap server 521, 523, 534–535, 566, 582
- BOOTSTRAP_ADDRESS 203
- BootstrapBasicMessaging 566
- bootstrapnoderooot 768
- bootstrapped client 534
- BootstrapSecureMessaging 566
- bootstrapserverroot 768
- BootstrapTunneledMessaging 566
- BootstrapTunneledSecureMessaging 566
- both mode 699–700
- Built-in Mail Provider 342, 344
- bus
 - including in federation 203
- bus connector role 609, 620
- bus member 541, 547, 550, 600, 602, 604, 610
 - adding to the service integration bus 631
- bus security 619

C

- cache 197, 694, 712, 728–729, 733, 855, 858
 - EJB 855, 857–858
- cache ID 689–690
- cache size 197
- cached connection handles 469
- cached handles 328, 340
- cacheGroups 269
- caching option 858
- caching option A 854
- caching option B 855
- caching option C 856
- Caching Proxy 6, 8
- caching strategy 858
- CCI 330
- cci.jar 332
- cell 31, 41, 50, 138
- cell name 59, 78, 94, 107
- cell persistent root 742, 748, 751, 753, 756, 763, 766–768, 772–773, 780–781
 - definition of 748
- cell profile 96–97
- cell root context 770
- cellroot 768
- central administration 50

- central management 48
- certificate 116
- certificate authority 115
- channel 194
- character encoding 875
- CICS 332
- class loader 198–199, 796, 800, 802–803, 807, 811, 818
 - Java 2 class loaders 796
 - policies 804
 - WebSphere class loaders 800
- class loader delegation 807
- class loader order 807
- class loader policies 880
- class loader policy 804, 806
- class loader tree 797
- Class Loader Viewer 810–811, 816
- class loader viewer service 200
- class loading 163
- class loading mode 190
- class loading policy 804
- class path 163
- classloader policy 190
- ClassNotFoundException 795, 802
- classpath 839
 - JDBC provider 315
 - protocol provider 346
 - resource adapter 337
 - URL provider 352
- clean 838, 853
- cleanup interval 197
- client connection 491–493
- client mode 699
- client/server replication 699
- client-server topology 702, 707
- CLOB 867
- clone separator 396
- Cloudscape 12, 306, 831, 848, 894
- cluster 144, 171, 222, 228, 231, 544, 594, 596, 599–600, 607–608, 610, 729, 854, 856, 895, 916, 924, 928, 934, 937–938
 - add a server using wsadmin 298–299
 - adding messaging engines 642
 - create with wsadmin 296
 - creating 222
 - managing with wsadmin 285
 - message-driven beans 607
 - messaging engine 596, 602
 - messaging engines 529

- restarting servers 183
- starting 285
- starting and stopping 226
- stopping 285
- viewing topology 226
- cluster short name 173
- cluster.xml 41
- CMP 861, 863, 868–869, 917
- CMP 2.0 enterprise bean 333
- cmpConnectionFactory 333
- collision detection 867
- com.ibm.scripting.host 253
- com.ibm.websphere.rsadapter 309
- com.ibm.ws.rsadapter.cci 309
- com.ibm.ws.rsadapter.jdbc 309
- com.ibm.ws.rsadapter.spi 309
- com.ibm.ws.scripting.connectionType 253–254
- com.ibm.ws.scripting.defaultLang 253
- com.ibm.ws.scripting.traceFile 253
- com.ibm.ws.scripting.traceString 253
- com.sun.jndi.LdapCtxFactory 745
- command assistance 147, 300
- Common Client Interface (CCI) 330
- Common Object Request Broker Architecture (CORBA) 744
- communication channel 31
- communication settings 199
- completeObject 287
- completeObjectName 262, 278, 281
- component managed authentication 361
- component-managed authentication 319, 339
- component-managed authentication alias 496, 517, 631, 910
- compound class loader 814
- concurrency control 862, 864
- concurrent message consumers 421
- configuration ID 267
- configuration reload 615, 618
- configurator role 161
- connected mode 254
- connection factory 332–333, 408, 515, 523, 526, 597
 - bindings 914
 - CMP 320
 - data source 310
 - J2C 337–338
 - JCA 331
 - resource adapter 333
 - WebSphere MQ 492
- connection handles 329
- connection management 423
- connection management contract 330
- Connection object 307, 411
- connection pool 732–733
- connection pooling 305, 308, 322, 330
- connection proximity 528, 530
- Connection Timeout 323–324
- ConnectionFactory object 411
- ConnectionWaitTimeoutException 323–324
- connectors
 - JMX 21
- console
 - See administrative console
- console page
 - preferences 150
- container authorization type 848
- container-managed activity session 738
- container-managed authentication 319, 362
- container-managed authentication alias 319
- container-managed persistence 320
 - See also CMP
- container-managed persistence (CMP) 830, 863
- container-managed relationships (CMR) 830, 869
- container-managed transaction 440, 738–739
- container-manager persistence (CMP) 847
- Content-Type header 875
- context 743, 760
- context root 233, 238–240, 853, 918, 929
- control region 112–113
- controller process 112–113
- controller region 168
- cookies 672, 676, 678–679, 681, 688
- CORBA 763, 784, 788
 - naming service groups 792
 - naming service users 791
 - URL 763–764
- corbaloc 742, 749, 752–753, 756–758, 764–765, 767, 769, 774, 776–777, 779–780, 784
- corbaname 742, 745–746, 757–759, 761–762, 765, 773–774
- core group 41, 164, 173, 200, 204, 207
- core group policy 597, 644
- CosNaming 743, 759, 761, 763, 766, 769, 784
 - CORBA 769
 - CORBA interface 744
 - distributed 759
 - INS 757
 - JNDI plug-in 766

- security 789
- CosNaming service 757
- CosNamingCreate 790
- CosNamingDelete 790
- CosNamingRead 790
- CosNamingWrite 790
- createApplicationServer 289
- createClusterMember 298
- createSubcontext 790
- createUsingTemplate 299
- CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESSES 203
- CTRACE parmlib member 110
- cursor management 860
- custom profile 49, 51, 56, 78–79, 84, 86, 88, 126, 201
- custom registry 6
- custom services settings 164
- custom user registry 117

D

- daemon 53, 114, 168, 170
- data centric 400
- data replication service 698–699, 703
- data source 306–308, 311, 326, 629, 831, 847, 881, 884, 888, 894, 904, 908–909, 917, 919
 - binding using the AAT 846
 - creating 311, 317
 - mapping to CMP beans 917
 - version 4 310
 - version 5 308
- data source classes 307
- data store 548, 568, 572, 627–628
- DataAccessFunctionSet 309
- database mapping 849, 916
- database mapping editor 850
- database persistence 694, 716
- database reauthentication 328
- Database servers 12
- DataDirect Technologies JDBC Drivers for WebSphere Application Server 9
- datagram 404
- DataSource object 306–307
- DataStoreHelper 309
- DB2 9, 306, 894
- DB2UNIVERSAL_JDBC_DRIVER_PATH 904
- DDL 231
- debugging service 200

- default application 112
- default bindings 914, 919
- default context 760
- default data source mapping 917
- default error page 874
- default node group 67
- default profile 51, 53, 58, 61, 68–69, 80, 370, 389
- default template 89
- default virtual host 191
- default_host 227, 241–242
- DefaultCoreGroup 164, 204
- DefaultNodeGroup 219–220
- defaultroot 768
- defaults 268
- delegation 807, 818
- deleteClusterMember 299
- delivery mode 552
- deployer role 161
- deployment code 911
- deployment descriptor 31, 42–43, 332, 839–841, 854, 860
 - application 31
 - EJB module
 - IBM extensions 871
 - viewing 237
- deployment manager 9, 17–18, 67, 71, 84
 - starting 128, 139, 166–167, 279
 - starting on z/OS 168
 - stopping 166, 168–169, 279
 - stopping on z/OS 170
- deployment manager name 107
- deployment manager node 67
- deployment manager profile 49–50, 56, 60, 94, 97, 125–126, 162, 164, 166–167, 171
- deployment.xml 43
- Derby 306, 831, 848
- DES 705
- destination 401–402, 409, 549
 - configuration 518
- destroySubcontext 790
- DeveloperServer template 89, 897
- development mode 189
- Diagnostic Provider framework 166
- Diagnostic Provider service 186
- digest 32, 34, 36
- directory browsing 874
- Directory servers 12
- dirty read 860
- discovery address 29

- distributed 162
- distributed discovery 28
- distributed process discovery 28
- distributed server environment 16–17, 26, 41, 50, 55, 162, 369, 918, 929–931, 934
- distributed transaction 421
- Distribution and Consistency Services (DCS) 200
- dmgr 138
 - server 751
 - starting 139
- dmgr_profile_home 39
- DNS domain 681
- dumpNameSpace 199, 782
- durable subscription 407, 430, 464, 468, 485
- dynamic cache 198, 703
- dynamic cache service 195
- dynamic caching 137
- dynamic reload 873, 938–939
- DynamicCache object type 268–269

E

- EAR file
 - changing manually 35
- Edge Components 10
- editInteractive 293
- EIS 330–332, 334
- eis/datasourcename_CMP 320
- EJB
 - binding 745
- EJB caching 854–856
- EJB caching options 854
- EJB client 771
- EJB container 196, 855
- EJB Deploy Tool 911
- EJB deployment code 848, 852, 911
- EJB home 742–743, 761, 769, 772–774
- EJB module
 - deployment descriptor
 - IBM extensions 871
 - viewing 236
- EJB prefix 914
- EJB reference 842, 844–845, 917
- EJB references
 - mapping to beans 917
- EJB timer service 197
- ejbCreate 437
- EJBDeploy 911–913
- ejbExtensions 860

- EJBHome 716
- ejb-jar.xml 839–840
- EJBLocalObject 740
- EjbNameSpaceBinding 772
- EJBObject 716, 740
- ejbRemove 436, 439
- ejbRemove() 860
- embedded configuration 915
- embedded JMS Server 591
- end of servlet service 709–710, 721–722
- endpoint listener 200
- Enhanced EAR 877–881, 890, 894, 901, 913, 915
- Enterprise Application project 832
- enterprise applications
 - extensions sharing session context 876
 - See also application
- enterprise beans
 - isolation level attributes 862
- Enterprise Edition management API 19
- Enterprise Information Systems (EIS) 329
- Enterprise JavaBeans
 - using EJBDeploy 911
- Enterprise JavaBeans (EJB) 7
- EntirePool 325
- entity beans 842, 846, 854, 856, 858, 861, 863, 867, 869, 871, 904, 914
 - caching options 854
- environment entry 163
- environment variables 895
 - See also variables
- epoch 32, 34, 37
- EventProvider 25
- EVERYONE 790–791
- exception destination 579–580
- Extensions class loader 796, 800
- extensions class loader 796, 799–801, 803, 817
- ExtJarClassLoader 826

F

- factoryClassname 359, 361
- fail back 598–599, 644, 646, 650
- FailingConnectionOnly 325
- failover 50, 137
 - hot 698
 - messaging engine 545
- federate 49–51, 56, 79, 82, 86, 201, 370
 - application server 91, 97
 - node 86

- federated name space 742–743, 746, 748–750, 752–753, 756, 760, 770, 782
- federated repository 6
- file permission mode mask 163
- file permissions 915
- File Serving Enabled 386
- File serving enabled option 873
- file serving enabler 872
- file serving servlet 813, 872
- file servlet 872
- file store 548, 568, 624, 626
 - backup and restore 572
 - deleting 571
- file synchronization service 32–33, 213
- file-based registry 6
- file-based repository 162
- file-based user registry 116–117
- filter 150
- findByPrimaryKey 865, 867, 869
- finder method 866
- fire and forget 404
- First Steps 59, 66–67, 74–75, 77, 84, 182
- foreign bus 555–556, 558–561, 581, 588, 590, 615, 650, 653, 655–656
 - service integration bus 651
 - WebSphere MQ 584
- foreign bus link 650
- foreign cell binding 770, 788
- foreign destination 550, 666
- frequency settings 709
- full resynchronize 37

G

- garbage collection 163, 729
- GenPluginCfg 389–390
- getAttribute 263
- getConnection() 307, 310
- getid 267
- getLocalHost() 30
- guided activities 144

H

- Handle 716
- heap size 163, 198, 729, 896
- help 158
- help portlet 147
- heuristic transactions 188
- HFS 53, 108, 110

- high availability 137
 - messaging 601, 603, 607
 - messaging engine 594
- high-volume Web sites 732
- HLQ field 102
- HomeHandle 716
- host alias 227–228, 242, 889, 901
- hosts file 903
- hot deployment 873, 938
- hot failover 698
- htpasswd 379
- HTTP session 701
- HTTP session context 877
- HTTP transport 191
- HttpServletRequest
 - getHeader 689
 - getSession 679, 725
- HttpServletResponse
 - encodeRedirectURL 682
 - encodeURL 682
- HttpSession 686, 710–712, 715–716, 719–721, 723–725, 730, 739
 - getAttribute 721
 - getId 689
 - removeAttribute 719
 - setAttribute 719, 721

I

- IBM DB2 12
- IBM HTTP Server 6, 9, 380, 385, 894, 902
 - admin password 379
 - administration server port 379
 - NameVirtualHosts 903
 - remote administration 378
 - Startup errors 903
 - VirtualHosts 902
- ibm-application-bnd.xmi 43
- ibm-application-ext.xmi 43
- ibm-ejb-access-bean.xmi 840
- ibm-ejb-jar-bnd.xmi files 840
- ibm-ejb-jar-ext.xmi 840
- ibm-partialapp-delete.props 933
- IBMSession
 - isOverflow 686
 - sync 710–711
- IIOP 757, 763, 765
 - URL 763
- IMAP 342–343, 349

- IMGBYTES 850
- immediate stop 183
- IMS 12
- inactive pool cleanup interval 196
- inbound user ID 653, 657
- InboundBasicMessaging 563
- InboundBasicMQLink 564
- InboundSecureMessaging 564
- InboundSecureMQLink 564, 587
- index.html 903
- indings 916
- indirect foreign bus 591
- indirect routing 557
- IndirectLookupNameSpaceBinding 773
- InetAddress.getLocalHost() 30
- Informix 12, 306
- initial context 742–743, 763–764, 766–768, 774, 782–783
 - default 772–773
- initial context factory 763, 765, 769, 773–774, 782
- initial root context 768
- initial state 197
- InitialContext 763, 768
- install 291–292
- installed optional packages 810
- installedApps directory 44
- installInteractive 291
- interface centric 401
- internal server classes 189
- Interoperable Naming Service (INS) 742, 757
- invoke 264, 280
- invoker servlet 873
- iscadmins role 162
- ISO-8859-1 875
- isolation level 860–862, 864
- isolation level attributes 862
- IsolationLevelChangeException 862
- ISPF Customization Dialog 95–96, 122

J

- J2C 332, 337
- J2C authentication data 905
- J2C connection factory 337
- J2EE 1.4 19
- J2EE application client 920
- J2EE client 925
- J2EE Connector Architecture (JCA) 7
- J2EE security 918

- J2SE 1.4 19
- JAAS authentication alias 831, 881
- JACC authorization provider 6
- Jacl 250, 256, 282, 288, 292
- Jacl2Jython 251
- JAF 342
- Java 2 Platform, Enterprise Edition (J2EE) 4
- Java 2 security 43
- Java 2 Standard Edition (J2SE) 4
- Java 5 7, 832
- Java Activation Framework (JAF) 342
- Java and process management settings 163
- Java API for XML Messaging (JAXM) 434
- Java Authorization Contract for Containers (JACC) 6
- java comp name 786
- java comp/env 355
- Java DataBase Connectivity 305
 - See also* Resource providers JDBC
- Java Management Extensions (JMX) 19
- Java Native Interface (JNI) 307
- java.net.URLConnection 351
- java.net.URLStreamHandler 351, 353
- JavaBeans Activation Framework (JAF) 342–343
- JavaMail 342–344, 347–348
- JavaServer Pages 874
- javax.ejb.EJBHome 715
- javax.ejb.EJBObject 715
- javax.ejb.MessageDrivenBean 436
- javax.ejb.MessageDrivenContext 439
- javax.jms.JMSEException 528
- javax.jms.MessageListener 436
- javax.mail.Session 831
- javax.management.ObjectName 19
- javax.naming.Context 715
- javax.naming.directory 745
- javax.naming ldap 745
- javax.naming.ObjectFactory 359
- javax.servlet.http.HttpSessionActivationListener 724
- javax.servlet.http.HttpSessionAttributeListener 724
- javax.servlet.http.HttpSessionListener 724
- javax.transaction.UserTransaction 715
- javax.xml.messaging.ReqRespListener 437
- JAXM 437
- JCA 329, 331, 337
- JCA CCI 308, 310
- JCA connection manager 308, 330
- JCA resource adapter 329

- JDBC driver 307, 904
- JDBC driver type 2 887
- JDBC isolation level 862
- JDBC provider 306, 311, 317, 629, 831, 881–882, 894, 904, 906
 - configuring with wsadmin 299
 - creating 311
- JMS activation specification 445–446, 452, 455, 482, 484–485, 488, 491, 514
- JMS activation specification. See also Activation-Spec JavaBean
- JMS administered object 408, 427, 458, 461, 488
- JMS client 520–521, 524, 526, 534, 565
- JMS connection 411
- JMS connection factory 408, 451, 462, 469–470
- JMS destination 409, 428
 - generic JMS provider configuration 518
- JMS domains 407
- JMS exception 419
- JMS message 413–414
- JMS message selector 414
- JMS provider 407, 451
 - default messaging 451, 455, 464, 476, 479, 484, 524
 - generic 457, 459, 515
 - WebSphere MQ 455, 491
- JMS server 757
- JMS session 412–413
- JMX 16, 20, 254, 259
 - agent 21
 - architecture 19
 - connectors 21
 - enabled management application 21
 - ObjectName 26
- JMX connector type 164, 202, 206
- JNDI 743, 761, 769, 772–773, 775–777, 779, 781, 783–786, 790
 - APIs 748
 - caching 769
 - client 763
 - Context.list() 783
 - EJB Home 761
 - initial context 783
 - initial context factory 773–774
 - javax.naming.package 745
 - javax.naming.provider.url 763
 - JMS 409–410
 - objects registered by dmgr server 751
 - over CosNaming 743

- provider URL 742
- service provider 773
- using to federate name space 769
- JNDI binding 745
- JNDI name 842, 917
- JNI 163, 803, 809, 921
- JRas 899
- JSESSIONID 679, 681, 689
- JSP 194
 - finding the URL 238
- JSP precompile 914
- JSP reload 233
- JSP reloading options 916
- JspBatchCompiler 915
- JSR 116 7, 190
- JSR 116 specification 6
- JSR 168 7, 195
- JSR-003 19
- JSR-77 19
- JSR-88 893
- JTA XAResource API 421
- JVM log 899
- Jython 117, 142, 250
- Jython editor 301

L

- launchClient 925
- LDAP 117, 745, 769
- LDAP registry 6
- LDAP server 12
- leaf binding 743
- leaf node 797
- leaf object 743
- legacyRoot 752–753, 756, 772, 781
- listener port 445, 447, 493, 511, 513–515
- listTemplates 299
- Load at option 857
- Load Balancer 6, 8
- load balancing 395
- local home interface 844
- local mode 254
- location service daemon 134
- Location Service Daemon (LSD) 30
- log 165, 179, 183, 377
 - HTTP 369
 - startServer 167, 169, 180
 - stderr 163
 - stdout 163

- stopServer.log 184
- Web server 382
- Log Analyzer 899
- log and trace settings 165
- log4j 808
- logging 898
- logs 895
 - profile creation 65, 74, 83
 - startServer.log 66, 75
 - SystemOut.log 66, 75
- long name 107–108
- loopback address 30
- loose coupling 400
- Lotus Domino Enterprise Server 12

M

- mail 831
- mail from 349
- mail provider 343–344
- mail store 349
- mail transport 349
- mail.jar 342
- managed application server 170–171
- managed node 97, 368–369, 371
- managed objects 19, 25
- managed process 16, 744, 748–749, 754
- managed server 31
- ManagedConnectionFactory 310, 332
- manageprofiles 123, 126
- manageprofiles script 96
- map modules to servers 915
- master configuration 17, 141
- master repository 31
 - reset 35
 - save changes to 153, 157
- match criteria 598, 648
- Max Connections 323–324
- maximum failed deliveries 579
- maximum in-memory session count 673
- Mbean extensions 164
- MBean proxy 23
- MBean server 21, 23
- mbeanIdentifier 261
- MBeans 19–21, 25, 27–28, 262, 278
 - server 20
 - TraceService 286
- mbList 260
- mediation 555

- mediations authentication alias 609, 620
- memory-to-memory replication 223, 673, 698–699, 703–704
- message consumer 404, 412, 415, 418, 559, 596, 601
- message consumer pattern 405
- message endpoint 425, 430–432
- message endpoint proxy 432
- message listener 418, 420, 426, 441
- message listener service 200
- message order 450, 553
- message point 550, 553
- message producer 404, 412, 415, 552–553, 559, 596
- message producer pattern 404
- message selector 483
- message store 548, 568
- message-driven beans 200, 425, 430–431, 434–440, 442–449, 452, 482–486, 490, 511, 515, 552, 596, 605, 607
 - life cycle 437
- MessageEndpointFactory 425
- messaging client 590
- messaging engine 526, 541, 543–544, 567, 594, 600, 605
 - data store 548, 572–574, 576, 622, 624, 627
 - failover 545
 - name 547
 - policy type 597
 - preferred server 643
 - secure communications 568
- messaging provider 400, 402
- meta-data 269
- metadata 46
- METHOD_READY state 859–860
- Microsoft SQL Server 12
- MIME 229, 342
- Min Connections 324
- missing transaction context 328, 340
- monitor role 161
- monitoring and tuning 145
- monitoring policy 199
- mount point 53
- MQ client 593
- multi-broker domain 704
- multibroker.xml
 - 708
- multicast 29–30
- multicast address 29

- multi-row persistent session management 733
- multi-row schema 717–718
- multi-row session support 731
- multithreaded access detection 328
- multi-threaded garbage collection 729

N

- name bindings 743
- name server 742
- name space 742–743, 745
 - federation 760, 769
- name space bindings 149
 - configuring 785–786, 788
- name space root 768
- NameService 759, 765, 767–768
- NameServiceCellPersistentRoot 767
- NameServiceCellRoot 767
- NameServiceNodeRoot 767
- NameServiceServerRoot 753, 756, 765, 767
- namestore.xml 42
- naming clients 743
- nanny process 130
- native library path 316
- native path 337
- native_stderr.log 900
- native_stdout.log 900
- navigation tree 143
- nboundBasicMQLink 587
- nhanced 840
- NoClassDefFoundError 799
- node
 - adding 205
 - See also addNode
 - managing 201
 - removing 209
 - See also removeNode
 - rename 212
 - starting 215
 - stopping 129, 213, 215, 219
 - synchronization 213
 - See also syncNode
- node agent 17, 23, 26, 29–31, 33, 41, 67, 84, 88, 94, 129, 138, 179, 205, 260, 281, 742
 - restarting 219
 - starting 85, 128, 179, 215, 280
 - stopping 85, 217, 280
 - stopping on z/OS 219
- node agent name 108

- node group 41, 67, 207
- node group members 221
- node name 59, 78–79
- node persistent root 742, 748, 754, 756, 763
 - definition of 748
- node repository
 - reset 36
- non-durable subscription 408, 608
- nonpersistent message reliability 466
- nonpersistent MQ messages 663
- non-repeatable read 861
- non-serializable J2EE objects 715
- Novell eDirectory 13

O

- object pools 305
- onMessage 437
- operating systems 10
- operations 264
- operator role 161
- optimistic 862
- optimistic concurrency 863, 867
- OptimisticPredicate 867
- Oracle 12, 306, 864
- ORB service 164, 198
- ORB_LISTENER_ADDRESS 203
- OSGi 801
- outbound user ID 653, 657
- OutboundBasicMQLink 567
- OutboundSecureMQLink 567
- overwrite session management 673

P

- parallel start 189
- parent 267
- PARENT_FIRST 807, 812
- PARENT_LAST 807, 818
- PARMLIB 104
- partition ID 689
- partitioned queue 596
- partitioned queues 595
- passivated 859–860
- passivation 196, 858
- peer-to-peer mode 698
- peer-to-peer replication 699
- peer-to-peer topology 700–701, 707
- preferred servers only 648
- performance 137

- performance monitoring 137
- performance monitoring service 188, 199
- periodic synchronization 33
- persistence 846
- persistence manager 309, 860, 865, 868, 871
- Persistence Resource Adapter 309
- persistent message reliability 466
- persistent MQ messages 663
- persistent partition 742
- persistent session 685–686, 715
- persistent session database 731
- persistent store 684, 854
- pessimistic 862
- PessimisticUpdate 866
- phantom read 861
- pluggable application client 921
- plug-in configuration file
 - automated propagation 393
 - automatic regeneration 391
 - propagating 392
 - regenerating 386, 389
 - viewing 388
- plugin-cfg.xml 385
- PMI 145, 199
- pmt.bat 57
- Point-to-Point domain 419
- Point-to-Point messaging model 402–403, 451, 455, 559
- poison message 450
- policy type 597
- POP3 342–343, 349
- port
 - bootstrap 757, 760, 763, 778, 782
 - CELL_DISCOVERY_ADDRESS 29
 - NODE_DISCOVERY_ADDRESS 29
 - NODE_MULTICAST_DISCOVERY_ADDRESS 29
 - SSL 228
- port settings 164
- portlet 195
- portlet applications 7
- portlet container 195
- portlet fragment caching 195
- ports 60, 62, 68, 70, 78, 90, 113, 192, 199
 - generate unique 173, 224, 898
- pre-compile JSP 915
- prefer local 223
- preferred server 597–598, 603, 643, 649
- preferred servers only 598–599, 603, 644, 646
- prepared transactions 188
- Problems view 836, 838
- process definition 198
- process execution 163, 199
- process group assignment 163
- Process log 900
- processor partitioning 163
- PROCLIB 104
- product information
 - viewing 188
- profile
 - deleting 126
 - exporting and importing 246
- profile creation wizard 57, 59–60, 67–68, 75, 80, 83, 123–125, 131, 171
- profile directory 39
- Profile Management Tool 57, 98
- profile name 78–79
- profile registry 123, 126
- profile_home 39, 53
- ProfileCreator 57
- profiler support 198
- profileRegistry.xml 123
- profiles 48
 - about 48
 - creating 57, 95
 - types 50
- programming model extensions 198, 305
- properties file 38
- property files 799
- protocol adapters 21
 - JMX 21
- protocol provider 342, 347
- protocol switch rewriting 683
- provider endpoint 522, 581, 597, 655
- provider URL 761–762, 765
 - javax.naming.provider.url 763
- proxy_host 227
- pseudo-synchronous messaging 405
- publication point 551
- publish/subscribe broker profile 587
- publish/subscribe domain 407, 419
- Publish/Subscribe messaging model 402–403, 451, 455, 561
- publish/subscribe profile 663
- pull mode 404, 416
- Purge Policy 325
- push mode 404, 417

Q

queryNames 260–261, 278, 284
queue destination 451, 472, 474, 476, 494,
501–502, 504, 549–550, 593
 creating 637
queue manager 560, 566, 584–586, 588, 593, 656,
663
queue point 550
queue sharing group 592
QueueConnectionFactory object 410

R

ra.xml 332, 839
RACF 116, 118
random 395
Rational Application Developer 8–9
Rational Web Developer 9
read ahead 476
read committed 861
read uncommitted 861
read-ahead 868–869
read-ahead hint 869–870
read-only modifiers 860
Reap Time 324–325
Reap Timeout 325
receiver channel 586, 662–663
recycle 155
Redbooks Web site 944
 Contact us xxii
referenceable 356, 358, 361
RegenerateKey 705
relational resource adapter 308–310
reliability 552
Reloading enabled option 873
remote home interface 844
Remote Request Dispatcher 915
remove() 860
removeNode 126, 209–211
renameNode 212
repeatable read 861
replica 699, 706
replication client 702
replication domain 700, 703–705
replication mode 707
replication server 702
replicationType 270
replicator entry 703–704
repository 17, 29–31, 39–41

 application data 42
 application execution 44
 saving work to 157
request metrics 145
request routing 366
request-reply 404
request-reply pattern 405
reset 275
resource adapter 330–334, 337, 340, 425, 605, 802
 deployment descriptor 426
 installation 333
 life cycle management 423
 message inflow management 424
 packaging 426
 service integration bus 451, 454, 605
 transaction inflow management 424
 WebSphere Relational Resource Adapter 332
 work management 423
Resource Adapter Archive (RAR) 332
resource environment provider 356–358
resource provider
 J2C 329, 331, 333, 337
 JavaMail 343
 JDBC 311, 317
 URL 351, 354–355
 Configuring URLs 353
resource reference 233, 917, 919
resources 144
resources.xml 312, 333, 356–357
response content type 875
response encoding 875
response file 100, 125
response time 190
res-sharing-scope 329
restoreConfig 243–244
retry interval 395
ripplestart 226
RMI 203, 207, 253
RMI connector 21
RMI/IIOP 207
Rollout Update 928, 934–936, 938
rollout update 231
round robin 395
Runtime Performance Advisor 199

S

SAF 116
SAF EJBROLE 116, 118

- SAF keyring name 116
- sample applications 76, 112
- SAP 332
- SBBOEXEC 107
- SBBOLD2 107
- SBBOLoad 104, 106
- SBBOLPA 106
- SBBOMSG 107
- scalability 137
- schedulers 305
- scope 42, 77, 147–148, 153
 - default 146
- Secure Sockets Layer (SSL) 194
- security 160, 620
 - session 725
 - session management 725
- security contract 330
- security domain identifier 118
- security management 423
- security roles 918
- sender channel 585–586, 662–663
- serializable 861
- serialize session access 673
- serialver utility 838
- serialVersionUID 838, 913
- servant process 113
- servant region 168
- serve servlets by class name 873
- server
 - starting 129, 139
 - status 129
 - stopping 129, 183
- server ID 690
- server mode 699
- server root 742, 763
- server root context 765–768, 772, 783
- server startup 33
- server template 68, 897
- server weight 224
- server.xml 205
- server1 49–50
- ServerCloneID 396
- serverindex.xml 29, 31, 94
- servers
 - database 12
 - directory 12
 - Web 11
- serverStatus 75, 129, 139, 175–176
- service
 - run deployment manager as 64, 68, 78
- service integration 145
- service integration bus 41, 92, 145, 200, 203, 206–207, 452, 521, 540
 - architecture 540
 - bus member. See bus member
 - clustering 594
 - configuration 612
 - connecting to 520
 - controlling messaging engine selection 524
 - creating 616
 - creating with wsadmin 275
 - data store 568, 573
 - destination 549, 638
 - exception destination 580
 - foreign bus. See foreign bus
 - JMS activation specification 445, 455, 482–483
 - JMS connection factory 451, 462–464
 - JMS destination 472
 - JMS queue 474, 477
 - JMS topic 480
 - link 581–583
 - load balancing bootstrapped clients 534
 - message-driven beans 605, 607
 - messaging engine. See messaging engine
 - quality of service 466
 - reliability 551, 553
 - resource adapter 445, 451, 453, 455
 - run time components 561
 - scalability 546
 - security 609
 - SIB service. See SIB service
 - topic destination 481
 - topologies 601–602
 - transport chain 563
 - WebSphere MQ addressing 588
 - XA recovery 469
- service integration bus link 556
- service provider 342
- Service Provider Interface (SPI) 330, 745
- servlet 194
- Servlet 2.2 API 691
- servlet caching 191
- servlet request and response pooling 191
- servlets
 - finding the URL 238
 - serve by class name 873
- servlets by class name 386
- session 190

- performance 726
- session administrative object 343
- session affinity 676, 678, 688, 729
- session attributes 876
- session beans 842, 858
 - caching options 858
- session cache 683
- session cleanup settings 713
- session context 674
- session ID 676, 679, 683, 688–689
- session identifier 679, 682
- Session Initiation Protocol (SIP) 190
- session invalidation time 712
- session management 190–191
 - affinity 689–692
 - cleanup schedule 723
 - DB2 page sizes 716
 - HTTP 672
 - invalidating sessions 723, 729
 - last access time 709
 - local 683
 - maximum in-memory session count 686
 - multi-row schemas 717
 - overflow 686
 - overflow cache 684, 686, 728
 - persistent 692, 694, 708
 - properties 672–674
 - row type 717
 - security ID 725
 - security integration 687
 - serializable requirements 715
 - serialize session access 688
 - session affinity 690
 - session cache size 728
 - session listeners 723
 - session object size 727
 - session size 727
 - session timeout 729
 - session tracking mechanism 676
 - single-row schemas 717
 - single-row to multi-row migration 718
 - SSL ID tracking 678
 - time-based write frequency 711
 - write contents 719–720
 - URL rewriting
 - See also cookies
- session management properties
 - application 673
 - application server 673, 735–736
 - Overwrite Session Management 674
 - Web module 674
- session manager 684, 698, 703, 706, 725, 729, 731–733
 - overflow 684
- session object size 727
- session persistence 678
- session scope 674
- session store 690
- session timeout 142, 673
- session write interval 712
- SessionBeanTimeoutException 859
- SESSIONS table 717
- SessionTableName 694
- setAttribute 263, 286–287
- setCharacterEncoding 875
- setMessageDrivenContext 436, 438
- setupCmdLine 205
- share session attributes 876
- shared libraries 149, 233, 802, 809, 820, 825, 917
- shared queue group 592–593
- shared session context 674
- shared session context option 876
- short name 107–108, 173
- showall 272, 295
- showAttribute 272
- SIB JMS Resource Adapter 454, 605
- SIB service 521, 561–564, 597, 604, 611, 613–615, 655
- sib.client.ssl.properties 568
- SIB_ENDPOINT_ADDRESS 523, 563, 614
- SIB_ENDPOINT_SECURE_ADDRESS 564, 614
- SIB_MQ_ENDPOINT_ADDRESS 564
- SIB_MQ_ENDPOINT_SECURE_ADDRESS 564
- SIB_MQ_ENDPOINT_ADDRESS 663
- sibDDLGenerator 577, 629
- silent mode 125
- Simple WebSphere Authentication Mechanism 725
- Singleton pattern 809
- Singletons 809
- SIP applications 7
- SIP container 190
- SMS 110
- SMTP 342–343
- snoop 242
- snoop servlet 239
- SOAP 207
- SOAP connector address 86
- SOAP connector port 35, 63, 71, 79, 82, 92, 199

- SOAP_CONNECTOR_ADDRESS 201, 204, 206
- SOAP_CONNECTOR_PORT 202
- special header 396
- SPI 310, 330, 745
- spilling 548
- SQL Server 306
- SQL92 conditional expression syntax 414
- SSL 115–116, 145, 194, 496, 567–568, 583, 586, 678
- SSL ID tracking 672
- SSL session ID 676, 678
- SSLV3TIMEOUT 678
- stand-alone application server 170–171
- stand-alone server 50
- stand-alone server environment 16, 41, 55
- standalone server environment 16, 55
- startApplication 284
- startManager 67, 128, 133, 139, 166–167, 169
- startNode 85, 128
- startServer 75, 129, 179, 181, 183
- startserver 377
- startServer.log 167, 169, 180
- stateful session bean 830, 858
- stateful session bean failover 703, 734–737
- stateful session beans 859
 - timeout 859
- stateless session bean 830
- stateless session EJB 858
- StateManageable 25
- statement cache size 327
- StatisticsProvider 25
- STEPLIB 107
- sticky bean managed unit of work 739
- stopApplication 284
- stopManager 129, 169
- stopNode 85, 129, 213, 216–217
- stopServer 77, 129, 182–183, 185, 279
- stopServer.log 184
- storeUsingOCC 867
- Stream Handler Class 352
- subscription durability 485
- Sun ONE Directory Server 12
- sun.misc.Launcher\$AppClassLoader 796
- SWAM 725
- Sybase 12, 306
- synchronization 32, 34, 37, 213
 - forced 33
 - scheduling 33
- synchronize 31, 37, 147, 158

- synchronous messaging 401
- syncNode 37, 213, 215
- system administration 145
- system partition 747
- SystemOut.log 128, 287

T

- Table.ddl 850, 852
- target groups 531
- target mappings 235
- target server 383
- taskbar 143
- template 89, 125, 172, 224, 299, 897
 - application server (creating) 174
- test connection 911
- thin application client 921
- thread pool 190, 200
- tightly coupled 400
- time-based write 711
- time-based writes 712
- timeout 860
 - aged 324–325
 - connection 323–324
 - ConnectionWaitTimeoutException 323–324
 - reap 325
 - session 687, 723, 729
 - unused 324–325
- Tivoli Access Manager 6
- Tivoli Access Manager Servers for WebSphere Application Server 10
- Tivoli Directory Server 6, 12
- Tivoli Directory Server for WebSphere Application Server 9
- Tivoli Performance Viewer 145
- topic destination 451, 472, 474–475, 479, 494, 501, 503, 508
- topic mappings 587
- topic space destination 549–550
 - creating 639
- topic space mappings 583, 655, 663
- topic subscriber 407
- trace 899–900
 - enabling using wsadmin 286
- TraceService 286
- tracing 898
- transaction
 - bean managed 442
 - commit 855

- container managed 440
- isolation level 860–861, 864
- message-driven beans 439
- viewing 187
- transaction management 423
- transaction management contract 330
- transaction service 186, 197
- transactions 187–188
- transient partition 742
- transport 620
- transport chain 165, 190, 194, 523, 528, 561, 563–565, 567, 583, 585–586, 617, 619, 621, 626, 662
- TRIPLE_DES 705
- types 266

U

- UDDI 145
- UnauthorizedSessionRequestException 725
- unique ID 611
- unit of work 738
- Universal Unique Identifier (UUID) 571
- unmanaged node 368–369, 371
- unprotected 2.x methods 918
- UriGroup Name 386
- URL provider 351, 354
- URL rewriting 672, 676, 678, 682–683
- Use Binary Configuration field 46
- user rights 64, 72
- users and groups 145
- utility classes 892
- utility JAR 802, 808, 817, 820

V

- variable 156, 316, 895–896, 904
- variable scoped files 42
- variables 149, 316–317
- variables.xml 42
- virtual host 233, 289, 386, 888, 890, 901, 903, 914, 918
 - admin_host 227
 - and Web server plug-in 228
 - architectural overview 227
 - binding 914
 - creating 228
 - creating with wsadmin 294
 - default_host 227, 241–242
 - example 901

- finding the URL for a servlet or JSP 238, 241
- host alias 228, 242
- IBM HTTP Server 902
 - in a cluster 228
 - managing 227
 - MIME settings 229
 - modifying with wsadmin 294
 - proxy_host 227
 - scope 228
- virtual hosting 902
- Virtual hosts
 - See also* IBM HTTP Server VirtualHosts
- virtual hosts 902

W

- WAR class loader 803–804, 807–809, 814, 816, 818, 820, 826
- WAR class loader policy 804
- WAR classloader 813
- was.policy 43
- WAS_EXT_DIRS 801
- was_home 51
- WAS_USER_SCRIPT 370
- wasprofile 53, 123–124, 126
- WASService 130–132, 166, 174
- WC_defaulthost 901
- Web container 191, 875
- Web container inbound transport chain 366
- Web container transport chain 165
- Web module 872, 876
 - auto reload 873
 - default error page 874
 - directory browsing 874
 - file serving servlet 872
 - serve servlets by class name 873
- Web module extensions 872
- Web server 916, 929–930, 934
 - adding 372
 - configuration file 380
 - logs 382
 - starting and stopping 376
 - viewing status 376
- Web server definition 79
- Web server plug-in 6, 9, 164, 201, 227–228, 279, 286, 366–367, 688, 691–692, 729, 873, 916, 929–930, 934
 - generating 390
 - regenerating 387

- regenerating with wsadmin 286
- request routing 393
- Web server plug-in configuration service 391
- Web servers 11
- web.xml 356, 839
- WebSphere Application Server - Express 7
- WebSphere Application Server Enterprise 305
- WebSphere Application Server for z/OS Profile Management Tool (zPMT) 95
- WebSphere Application Server Network Deployment 8
- WebSphere Application Server V6.1 for z/OS 8
- WebSphere Business Integration Event Broker 455
- WebSphere Business Integration Message Broker 455
- WebSphere Business Integration Server Foundation 305
- WebSphere Information Integrator 12
- WebSphere MQ 6, 455, 556, 588
- WebSphere MQ Channel 635
- WebSphere MQ client 588
- WebSphere MQ client link 591
- WebSphere MQ connection factory 492, 498, 515
- WebSphere MQ link 584–589, 592–593, 612, 658
- WebSphere MQ Server 557, 563, 592–594, 612–613, 622, 633–635
- WebSphere Rapid Deployment 229
- WebSphere Relational Resource Adapter 332
- Windows Active Directory 13
- Windows service 64, 72, 94, 130, 132, 174, 207
- WLM 108, 855
- WLM APPLENV 108
- work area partition service 198
- work area service 198
- working directory 898
- workload management 50, 734
 - EJS WLM 854
 - messaging 600–601, 603–604, 608
 - messaging engine 594, 601
- workload-managed 596
- workspace 143, 145
 - auto-refresh 146
 - confirmation on discard 146
- ws.ext.dirs 801
- wsadmin 18, 229, 249–250
 - definition 250
 - getid 267
 - help 252
 - interactive 255
 - list 268
 - profile 254, 256
 - properties 252, 254, 256
 - script files 256
 - show 271
 - showattribute 272
 - starting session 251
- WSCallerHelper 309
- wsOptimisticRead 864, 866
- wsOptimisticUpdate 865
- wsPessimisticRead 864
- wsPessimisticUpdate 864
- wsPessimisticUpdate-NoCollision 868
- wsPessimisticUpdate-WeakestLockAtLoad 867
- wstemp 141

X

- X509 115

Z

- z/OS Security Server 12
- z/OS.e Security Server 12
- zFS 53, 108, 110
- zPMT 95–96, 98, 100, 120, 209



Redbooks

WebSphere Application Server V6.1: System Management and Configuration

(1.5" spine)
1.5" <-> 1.998"

789 <-> 1051 pages



WebSphere Application Server V6.1:

System Management and Configuration

**Learn about
WebSphere
Application Server**

**Configure and
administer a
WebSphere system**

Deploy applications

This IBM Redbook provides system administrators, developers, and architects with the knowledge to configure a WebSphere Application Server V6.1 run time environment, to package and deploy Web applications, and to perform ongoing management of the WebSphere environment.

One in a series of handbooks, the entire series is designed to give you in-depth information about the entire range of WebSphere Application Server products. In this IBM Redbook, we provide a detailed exploration of the WebSphere Application Server V6.1 run time environments and administration process.

The IBM Redbook includes configuration and administration information for WebSphere Application Server V6.1 and WebSphere Application Server Network Deployment V6.1 on distributed platforms (excluding iSeries) and WebSphere Application Server for z/OS V6.1.

**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