

WebSphere Application Server V6 System Management and Configuration Handbook

Read this book and others in the
WebSphere Handbook Series

Learn to design and administer
your own system

Customize profiles,
scripts and applications



Carla Sadtler
Lars Bek Laursen
Martin Phillips
Henrik Sjostrand
Martin Smithson
Kwan-Ming Wan

Redbooks



International Technical Support Organization

**WebSphere Application Server V6: System
Management and Configuration Handbook**

February 2005

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

First Edition (February 2005)

This edition applies to Version 6 of IBM WebSphere Application Server.

© Copyright International Business Machines Corporation 2005. All rights reserved.

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

Contents

Preface	xix
The team that wrote this redbook.....	xix
Become a published author	xxi
Comments welcome.....	xxi
Notices	xxiii
Trademarks	xxiv
Part 1. The basics	1
Chapter 1. WebSphere Application Server V6 for distributed platforms ..	3
1.1 WebSphere overview	4
1.2 WebSphere family.....	5
1.3 WebSphere Application Servers	6
1.4 WebSphere Application Server for distributed platforms.....	8
1.4.1 Packaging.....	8
1.4.2 System requirements and support for distributed platforms	11
1.4.3 New for V6	12
Chapter 2. WebSphere Application Server V6 architecture	19
2.1 Application server configurations	20
2.1.1 Stand-alone server configuration	20
2.1.2 Distributed server configuration	21
2.2 Application servers, nodes, and cells	22
2.2.1 Application servers	23
2.2.2 Nodes, node groups, and node agents.....	23
2.2.3 Cells	23
2.3 Servers	24
2.3.1 Application servers	24
2.3.2 Clusters.....	24
2.3.3 JMS servers (V5)	25
2.3.4 External servers	25
2.4 Containers.....	26
2.4.1 Web container	26
2.4.2 Enterprise JavaBeans container.....	28
2.4.3 Application client container	28
2.5 Application server services	28
2.5.1 J2EE Connector Architecture services	30
2.5.2 Transaction service	30

2.5.3	Dynamic cache service	31
2.5.4	Message listener service	32
2.5.5	Object Request Broker service	32
2.5.6	Administrative service	33
2.5.7	Name service	33
2.5.8	Performance Monitoring Infrastructure service	35
2.5.9	Security service	36
2.6	Data Replication Service	36
2.7	Virtual hosts	37
2.8	Session management	38
2.8.1	HTTP Session persistence	39
2.8.2	Stateful session EJB persistence	40
2.9	Web services	40
2.9.1	Enterprise services (JCA Web services)	42
2.9.2	Web service client	43
2.9.3	Web service provider	43
2.9.4	Enterprise Web Services	43
2.9.5	IBM WebSphere UDDI Registry	44
2.9.6	Web Services Gateway	44
2.10	Service integration bus	46
2.10.1	Application support	48
2.10.2	Service integration bus and messaging	48
2.10.3	Web services and the service integration bus	50
2.11	Security	51
2.11.1	User registry	53
2.11.2	Authentication	54
2.11.3	Authorization	55
2.11.4	Security components	56
2.11.5	Security flows	57
2.12	Resource providers	58
2.12.1	JDBC resources	59
2.12.2	Mail providers	60
2.12.3	JCA resource adapters	61
2.12.4	URL providers	62
2.12.5	JMS providers	62
2.12.6	Resource environment providers	63
2.13	Workload management	64
2.14	High availability	66
2.15	Administration	67
2.15.1	Administration tools	68
2.15.2	Configuration repository	69
2.15.3	Centralized administration	69
2.16	The flow of an application	71

2.17 Developing and deploying applications	72
2.17.1 Application design	73
2.17.2 Application development	73
2.17.3 Application packaging	74
2.17.4 Application deployment	74
2.17.5 WebSphere Rapid Deployment	75
2.18 Technology support summary	76
Chapter 3. System management: A technical overview	81
3.1 System management overview	82
3.1.1 System management tools	82
3.1.2 System management in a standalone server environment	83
3.1.3 System management in a distributed server environment	84
3.1.4 Role-based administration	85
3.2 Java Management Extensions (JMX)	85
3.2.1 JMX architecture	86
3.2.2 JMX distributed administration	88
3.2.3 JMX MBeans	90
3.2.4 JMX usage scenarios	90
3.2.5 J2EE management	91
3.3 Distributed administration	92
3.3.1 Distributed process discovery	94
3.3.2 Centralized changes to configuration and application data	97
3.3.3 File synchronization	98
3.4 Configuration and application data repository	104
3.4.1 Repository directory structure	104
3.4.2 Variable scoped files	107
3.4.3 Application data files	107
Chapter 4. Getting started with profiles	113
4.1 Understanding profiles	114
4.1.1 Types of profiles	116
4.1.2 Directory structure and default profiles	117
4.2 Building a system using profiles	119
4.2.1 Standalone server environment	119
4.2.2 Distributed server environment	119
4.3 Creating profiles	121
4.3.1 Creating a deployment manager profile	123
4.3.2 Creating an application server profile	130
4.3.3 Creating a custom profile	138
4.3.4 Federating a custom node to a cell	145
4.3.5 Creating a new application server on an existing node	146
4.3.6 Federating an application server profile to a cell	149

4.4	Creating profiles manually.....	151
4.4.1	Using the wasprofile command.....	151
4.4.2	Creating a profile.....	153
4.5	Managing the processes	154
4.5.1	Starting a distributed server environment	154
4.5.2	Stopping the distributed server environment.....	156
4.5.3	Enabling process restart on failure	157
Chapter 5.	Administration basics	161
5.1	Introducing the WebSphere administrative console	162
5.1.1	Starting the administrative console	162
5.1.2	Logging in to the administrative console.....	164
5.1.3	Changing the administrative console session timeout	165
5.1.4	The graphical interface	166
5.1.5	Finding an item in the console	169
5.1.6	Updating existing items.....	174
5.1.7	Adding new items	176
5.1.8	Removing items	177
5.1.9	Starting and stopping items.....	177
5.1.10	Using variables	179
5.1.11	Saving work.....	180
5.1.12	Getting help.....	181
5.2	Securing the administrative console	182
5.3	Working with the deployment manager.....	183
5.3.1	Deployment manager configuration settings.....	183
5.3.2	Starting and stopping the deployment manager	187
5.4	Working with application servers.....	190
5.4.1	Creating an application server	191
5.4.2	Viewing the status of an application server.....	194
5.4.3	Starting an application server	197
5.4.4	Stopping an application server	200
5.4.5	Viewing runtime attributes of an application server.....	203
5.4.6	Customizing application servers	206
5.5	Working with nodes.....	217
5.5.1	Adding a node.....	217
5.5.2	Removing a node	224
5.5.3	Node agent synchronization	227
5.5.4	Starting and stopping nodes	230
5.5.5	Node groups	233
5.6	Working with clusters	235
5.6.1	Creating clusters	236
5.6.2	Viewing cluster topology	238
5.6.3	Managing clusters	239

5.7 Working with virtual hosts	239
5.7.1 Creating a virtual host	240
5.8 Managing applications	242
5.8.1 Using the administrative console to manage applications	242
5.8.2 Installing an enterprise application	244
5.8.3 Uninstalling an enterprise application	246
5.8.4 Exporting an enterprise application	246
5.8.5 Starting an enterprise application	247
5.8.6 Stopping an enterprise application	247
5.8.7 Preventing an enterprise application from starting on a server	247
5.8.8 Viewing installed applications	248
5.8.9 Viewing EJB modules	250
5.8.10 Viewing Web modules	252
5.8.11 Finding a URL for a servlet or JSP	253
5.9 Managing your configuration files	258
5.9.1 Backing up a profile configuration	259
5.9.2 Restoring a node configuration	260
5.9.3 Exporting and importing profiles	262
5.9.4 Deleting profiles	263
Chapter 6. Administration with scripting	267
6.1 Overview of WebSphere scripting	268
6.2 Using wsadmin	268
6.2.1 Launching wsadmin	268
6.2.2 Configuring wsadmin	270
6.2.3 Commands and scripts invocation	271
6.2.4 Overview of wsadmin objects	274
6.2.5 Management using wsadmin objects	276
6.3 Common operational administrative tasks using wsadmin	292
6.3.1 General approach for operational tasks	292
6.3.2 Examples of common administrative tasks	293
6.3.3 Managing the deployment manager	293
6.3.4 Managing nodes	294
6.3.5 Managing application servers	295
6.3.6 Managing enterprise applications	297
6.3.7 Managing clusters	299
6.3.8 Generating the Web server plug-in configuration	300
6.3.9 Enabling tracing for WebSphere components	300
6.4 Common configuration tasks	302
6.4.1 General approach for configuration tasks	302
6.4.2 Specific examples of WebSphere configuration tasks	302
6.5 Differences from WebSphere V5	315
6.6 End-to-end examples	316

6.7 Using Java for administration	316
Online resources	317
Chapter 7. Configuring WebSphere resources.....	319
7.1 WebSphere resources.....	320
7.2 JDBC resources	321
7.2.1 What are JDBC providers and data sources?.....	321
7.2.2 WebSphere support for data sources	322
7.2.3 Creating a data source	326
7.2.4 Creating a JDBC provider.....	326
7.2.5 Creating JDBC data source	331
7.3 JCA resources.....	341
7.3.1 WebSphere Application Server JCA support	344
7.3.2 Installing and configuring resource adapters	345
7.3.3 Configuring J2C connection factories	349
7.3.4 Using resource adapters from an application	353
7.4 JavaMail resources	354
7.4.1 JavaMail sessions.....	356
7.4.2 Configuring the mail provider	356
7.4.3 Configuring JavaMail sessions	360
7.4.4 Example code	363
7.5 URL providers	364
7.5.1 Configuring URL providers	364
7.5.2 Configuring URLs	366
7.5.3 URL provider sample	368
7.6 Resource environment providers	369
7.6.1 Resource environment references	370
7.6.2 Configuring the resource environment provider	371
7.7 Resource authentication	374
7.8 More information	376
Chapter 8. Managing Web servers.....	377
8.1 Web server support overview	378
8.1.1 Request routing using the plug-in	379
8.1.2 Web server and plug-in management.....	380
8.2 Web server installation examples	383
8.2.1 Standalone server environment	384
8.2.2 Distributed server environment.....	386
8.3 Working with Web servers.....	389
8.3.1 Defining nodes and Web servers	389
8.3.2 Viewing the status of a Web server.....	394
8.3.3 Starting and stopping a Web server	395
8.3.4 IBM HTTP Server remote administration	397

8.3.5	Mapping modules to servers	402
8.4	Working with the plug-in configuration file	404
8.4.1	Regenerating the plug-in configuration file	406
8.4.2	Propagating the plug-in configuration file	411
8.4.3	Modifying the plug-in request routing options	412
Chapter 9.	Problem determination	417
9.1	Resources for identifying problems	418
9.2	Administrative console messages	419
9.3	Log files	420
9.3.1	JVM (standard) logs	421
9.3.2	Process (native) logs	428
9.3.3	IBM service (activity) log	428
9.4	Traces	430
9.4.1	Diagnostic trace service	431
9.4.2	Web server logs and traces	439
9.5	Log Analyzer	443
9.5.1	Using Log Analyzer	444
9.5.2	Merging logs on multiple application servers	449
9.5.3	Updating the symptom database	450
9.6	Collector tool	451
9.7	First Failure Data Capture logs	452
9.8	Dumping the contents of the name space	453
9.9	HTTP session monitoring	454
9.10	Application debugging and tracing	455
9.10.1	Application Server Toolkit	456
9.10.2	Java logging interface	456
9.11	Product installation information	456
9.11.1	Using the administrative console to find product information	457
9.11.2	Locating WebSphere Application Server version information	457
9.11.3	Finding the JDK version	459
9.11.4	Finding the IBM HTTP Server version	459
9.12	Resources for problem determination	459
Part 2.	Messaging with WebSphere	461
Chapter 10.	Asynchronous messaging	463
10.1	Messaging concepts	464
10.1.1	Loose coupling	464
10.1.2	Messaging types	465
10.1.3	Destinations	465
10.1.4	Messaging models	466
10.1.5	Messaging patterns	467
10.2	Java Message Service	470

10.2.1	JMS API history	470
10.2.2	JMS providers	471
10.2.3	JMS domains	471
10.2.4	JMS administered objects	472
10.2.5	JMS and JNDI	473
10.2.6	JMS connections	475
10.2.7	JMS sessions	476
10.2.8	JMS messages	476
10.2.9	JMS message producers	478
10.2.10	JMS message consumers	479
10.2.11	JMS exception handling	482
10.2.12	Application Server Facilities	484
10.2.13	JMS and J2EE	485
10.3	Messaging in the J2EE Connector Architecture	485
10.3.1	Message endpoints	488
10.3.2	MessageEndpointFactory	488
10.3.3	Resource adapters	488
10.3.4	JMS ActivationSpec JavaBean	491
10.3.5	Message endpoint deployment	494
10.3.6	Message endpoint activation	495
10.3.7	Message delivery	496
10.3.8	Administered objects	497
10.4	Message-driven beans	497
10.4.1	Message-driven bean types	498
10.4.2	Client view of a message-driven bean	498
10.4.3	Message-driven bean implementation	499
10.4.4	Message-driven bean life cycle	501
10.4.5	Message-driven beans and transactions	502
10.4.6	Message-driven bean activation configuration properties	507
10.4.7	Associating a message-driven bean with a destination	509
10.4.8	Message-driven bean best practices	511
10.5	Managing WebSphere JMS providers	514
10.5.1	Managing the default messaging JMS provider	514
10.5.2	Managing the WebSphere MQ JMS provider	519
10.5.3	Managing a generic JMS provider	522
10.6	Configuring WebSphere JMS administered objects	526
10.6.1	Common administration properties	526
10.6.2	Configuring the default messaging JMS provider	527
10.6.3	Configuring the WebSphere MQ JMS provider	552
10.6.4	Configuring listener ports	568
10.6.5	Configuring the generic JMS provider	572
10.7	Connecting to a service integration bus	576
10.7.1	JMS client runtime environment	576

10.7.2	Controlling messaging engine selection	579
10.7.3	Load balancing bootstrapped clients.	588
10.8	References and resources	590
Chapter 11.	Default messaging provider.	593
11.1	Concepts and architecture	594
11.1.1	Buses	594
11.1.2	Bus members	595
11.1.3	Messaging engines	595
11.1.4	Data stores	601
11.1.5	Destinations	602
11.1.6	Mediations.	606
11.1.7	Foreign buses	606
11.2	Runtime components	612
11.2.1	SIB service	612
11.2.2	Service integration bus transport chains	614
11.2.3	Data stores	620
11.2.4	Exception destinations	625
11.2.5	Service integration bus links	627
11.2.6	WebSphere MQ links	629
11.3	High availability and workload management.	638
11.3.1	Cluster bus members for high availability	638
11.3.2	Cluster bus members for workload management	638
11.3.3	Partitioned queues	639
11.3.4	JMS clients connecting into a cluster of messaging engines	640
11.3.5	Preferred servers and core group policies	641
11.3.6	Best practices	644
11.4	Service integration bus topologies	645
11.4.1	One server in the cell is a member of one bus	645
11.4.2	Every server in the cell is a member of the same bus	646
11.4.3	A single cluster bus member and one messaging engine.	646
11.4.4	A cluster bus member with multiple messaging engines	647
11.4.5	Mixture of cluster and server bus members	647
11.4.6	Multiple buses in a cell	648
11.5	Service integration bus and message-driven beans	649
11.5.1	Message-driven beans connecting to the bus.	649
11.5.2	MDBs and clusters	651
11.6	Service integration bus security	652
11.7	Problem determination	653
11.8	Configuration and management	655
11.8.1	SIB service configuration	655
11.8.2	Creating a bus.	658
11.8.3	Adding a bus member using a default data store	660

11.8.4 Adding a bus member with a different data store	661
11.8.5 Creating a queue destination	666
11.8.6 Creating a topic space destination	668
11.8.7 Creating an alias destination.....	668
11.8.8 Adding messaging engines to a cluster	670
11.8.9 Setting up preferred servers	672
11.8.10 Setting up a foreign bus link to a service integration bus	678
11.8.11 Setting up a foreign bus link to an MQ queue manager	683
11.8.12 Creating a foreign destination.....	692
Part 3. Working with applications	695
Chapter 12. Session management.....	697
12.1 What is new?.....	698
12.2 HTTP session management	698
12.3 Session manager configuration.....	699
12.3.1 Session management properties	699
12.3.2 Accessing session management properties	700
12.4 Session scope.....	700
12.5 Session identifiers	702
12.5.1 Choosing a session tracking mechanism	703
12.5.2 SSL ID tracking	704
12.5.3 Cookies	705
12.5.4 URL rewriting	709
12.6 Local sessions.....	710
12.7 General properties for session management	711
12.8 Session affinity	715
12.8.1 Session affinity and failover	717
12.9 Persistent session management	719
12.9.1 Enabling database persistence.....	721
12.9.2 Memory-to-memory replication	723
12.9.3 Session management tuning.....	734
12.9.4 Persistent sessions and non-serializable J2EE objects	741
12.9.5 Larger DB2 page sizes and database persistence	742
12.9.6 Single and multi-row schemas (database persistence).....	743
12.9.7 Contents written to the persistent store using a database	745
12.10 Invalidating sessions	749
12.10.1 Session listeners.....	749
12.11 Session security	751
12.12 Session performance considerations	752
12.12.1 Session size	753
12.12.2 Reducing persistent store I/O	756
12.12.3 Multirow persistent sessions: Database persistence	757

12.12.4	Managing your session database connection pool	758
12.12.5	Session database tuning	759
12.13	Stateful session bean failover	760
12.13.1	Enabling stateful session bean failover	760
12.13.2	Stateful session bean failover considerations	764
Chapter 13.	WebSphere naming implementation	769
13.1	Features	770
13.2	WebSphere naming architecture	771
13.2.1	Components	771
13.2.2	JNDI support	772
13.2.3	JNDI bindings	773
13.2.4	Federated name space	774
13.2.5	Local name space structure	777
13.3	Interoperable Naming Service (INS)	785
13.3.1	Bootstrap ports	785
13.3.2	CORBA URLs	785
13.4	Distributed CosNaming	787
13.5	Configured bindings	788
13.5.1	Types of objects	789
13.5.2	Types of binding references	789
13.6	Initial contexts	791
13.6.1	Setting initial root context	793
13.7	Federation of name spaces	797
13.8	Interoperability	798
13.8.1	WebSphere V4.0 EJB clients	798
13.8.2	WebSphere V4.0 server	800
13.8.3	EJB clients hosted by non-WebSphere environment	800
13.9	Examples	801
13.9.1	Single server	801
13.9.2	Two single servers on the same box	803
13.9.3	Network Deployment application servers on the same box	804
13.9.4	WebSphere Application Server V4 client	807
13.10	Naming tools	808
13.10.1	dumpNameSpace	808
13.11	Configuration	811
13.11.1	Name space bindings	811
13.11.2	CORBA naming service users and groups	815
Chapter 14.	Understanding class loaders	821
14.1	A brief introduction to Java class loaders	822
14.2	WebSphere class loaders overview	825
14.2.1	WebSphere extensions class loader	826

14.2.2 Application and Web module class loaders	827
14.2.3 Handling JNI code	828
14.3 Configuring WebSphere for class loaders	828
14.3.1 Class loader policies	828
14.3.2 Class loader/delegation mode	831
14.3.3 Class preloading	833
14.3.4 Shared libraries	834
14.4 Learning class loaders by example	835
14.4.1 Step 1: Simple WAR packaging	837
14.4.2 Step 2: Sharing the utility JAR among multiple modules	838
14.4.3 Step 3: Changing the WAR class loader delegation mode	839
14.4.4 Step 4: Sharing utility JARs among multiple applications	840
Chapter 15. Packaging applications	847
15.1 WebSphere Bank sample application	848
15.1.1 WebSphere Bank resources used	849
15.2 Packaging using the Application Server Toolkit	850
15.2.1 Preparing the sample code	851
15.2.2 Importing an EAR file	851
15.2.3 Working with deployment descriptors	857
15.3 Setting application bindings	860
15.3.1 Defining EJB JNDI names	861
15.3.2 Binding EJB and resource references	862
15.3.3 Binding the message-driven bean to an ActivationSpec	864
15.3.4 Defining data sources for entity beans	865
15.4 IBM EJB extensions: EJB caching options	870
15.4.1 EJB container caching option for entity beans	870
15.4.2 EJB container caching option for stateful session beans	873
15.4.3 Stateful EJB timeout option	874
15.5 IBM EJB extensions: EJB access intents	875
15.5.1 Transaction isolation levels overview	876
15.5.2 Concurrency control	877
15.5.3 Using EJB 2.x access intents	878
15.5.4 Using read-ahead hints	883
15.5.5 Tracing access intents behavior	885
15.6 IBM EJB extensions: Inheritance relationships	885
15.7 IBM Web module extensions	886
15.7.1 File serving servlet	886
15.7.2 Web application auto reload	887
15.7.3 Serve servlets by class name	887
15.7.4 Default error page	888
15.7.5 Directory browsing	888
15.7.6 JSP attributes	888

15.7.7 Automatic HTTP request and response encoding	888
15.8 IBM EAR extensions: Sharing session context.....	889
15.9 Exporting WebSphere Bank EAR file	891
15.10 WebSphere Enhanced EAR	891
15.10.1 Configuring a WebSphere Enhanced EAR.....	892
15.11 Packaging recommendations	904
Chapter 16. Deploying applications.....	907
16.1 Preparing the environment	908
16.1.1 Creating the WebSphere Bank DB2 database	908
16.1.2 Creating a WEBSPHEREBANK_ROOT environment variable.....	909
16.1.3 Creating the WebSphere Bank application server	910
16.1.4 Defining the WebSphere Bank virtual host	914
16.1.5 Creating the virtual host for IBM HTTP Server and Apache	915
16.1.6 Creating a DB2 JDBC provider and data source	917
16.1.7 Configuring the messaging resources.....	925
16.2 Generating deployment code	928
16.2.1 Using EJBDeploy command line tool	928
16.3 Deploying the application	930
16.3.1 Using a bindings file	936
16.4 Deploying application clients.....	937
16.4.1 Defining application client bindings.....	940
16.4.2 Launching the J2EE client.....	941
16.5 Updating applications	944
16.5.1 Replacing an entire application EAR file.....	945
16.5.2 Replacing or adding an application module	945
16.5.3 Replacing or adding single files in an application or module	946
16.5.4 Removing application content.....	947
16.5.5 Performing multiple updates to an application or module	948
16.5.6 Rolling out application updates to a cluster.....	951
16.5.7 Hot deployment and dynamic reloading	954
Chapter 17. WebSphere Rapid Deployment	957
17.1 Annotation-based programming	958
17.2 Rapid deployment tools	960
17.3 Using rapid deployment commands	962
17.3.1 wrd-config command.....	962
17.3.2 wrd command	966
17.4 Free-form projects	967
17.5 Free-form development example	970
17.5.1 Setting up the environment for free-form development.....	971
17.5.2 Adding application source code	974
17.5.3 Terminating the WebSphere Rapid Deployment session	981

17.5.4 Verifying results.....	981
17.6 Automatic application installation projects.....	983
17.7 Automatic application installation example	983
17.7.1 Setting up an automatic application installation session.....	984
17.7.2 Managing applications	986
Related publications	989
IBM Redbooks	989
Other publications	989
Online resources	990
How to get IBM Redbooks	991
Help from IBM	992
Index	993

Preface

This IBM Redbook provides system administrators, developers, and architects with the knowledge to configure a WebSphere Application Server V6 runtime 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 book, we provide a detailed exploration of the WebSphere Application Server V6 runtime environments and administration process.

The team that wrote this redbook

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

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.

Lars Bek Laursen is an Advisory IT Specialist at the Integrated Technology Services division of IBM Global Services in Lyngby, Denmark. 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 runtime environments. Lars holds a Master of Science in Engineering degree from the Technical University of Denmark.

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.

Henrik Sjostrand is a Senior IT Specialist and has worked for IBM Sweden for ten years. He has 12 years of experience in the IT field. During his time with IBM he has had a number of different positions, from consulting and education to pre-sales activities. He is currently working as a technical consultant for the Nordic IBM Software Services for WebSphere team. The last four years, he has focused on e-business application development, and WebSphere Application Server architecture and deployment. He is certified in WebSphere Application Server v4 and v5 and WebSphere Studio v5. Henrik holds a Master of Science in Electrical Engineering from Chalmers University of Technology in Gothenburg, Sweden, where he lives.

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 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:

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

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

Sam Cleveland
WebSphere Application Server Samples Development

The authors of *IBM WebSphere Application Server V5.1 System Management and Configuration*, SG24-6195:

Lee Clifford
Jeff Heyward
Arihiro Iwamoto
Noelle Jakusz
Lars Bek Laursen
WonYoung Lee
Isabelle Mauny
Shafkat Rabbi
Ascension Sanchez
Authors of the V5 book

Become a published author

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

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbook@us.ibm.com

- ▶ Mail your comments to:

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

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

Trademarks

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

AIX®	eServer™	Rational®
alphaWorks®	ETE™	Redbooks (logo)™
Balance®	ibm.com®	Redbooks (logo)  ™
CICS®	IBM®	Redbooks™
ClearCase®	IMS™	S/390®
Cloudscape™	Informix®	SAA®
DB2 Universal Database™	iSeries™	SLC™
DB2®	Lotus®	SupportPac™
developerWorks®	Notes®	Tivoli®
Domino®	OS/400®	WebSphere®
e-business on demand™	Perform™	XDE™
®server®	pSeries®	z/OS®
®server®	Rational Rose®	zSeries®

The following terms are trademarks of other companies:

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

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

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

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

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

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



Part 1

The basics

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

This part includes the following:

- ▶ Chapter 1, “WebSphere Application Server V6 for distributed platforms” on page 3
- ▶ Chapter 2, “WebSphere Application Server V6 architecture” on page 19
- ▶ Chapter 3, “System management: A technical overview” on page 81
- ▶ Chapter 4, “Getting started with profiles” on page 113
- ▶ Chapter 5, “Administration basics” on page 161
- ▶ Chapter 6, “Administration with scripting” on page 267
- ▶ Chapter 7, “Configuring WebSphere resources” on page 319
- ▶ Chapter 8, “Managing Web servers” on page 377
- ▶ Chapter 9, “Problem determination” on page 417



WebSphere Application Server V6 for distributed platforms

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 for distributed platforms.

1.1 WebSphere 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 past several years, many software vendors have collaborated on a set of server-side application programming technologies that help build Web accessible, distributed, 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 family

The WebSphere platform forms the foundation of a comprehensive business solutions framework. Its extensive offerings are designed to solve the problems of companies of all different sizes. For example, the technologies and tools at the heart of the WebSphere platform can be used to build and deploy the core of an international financial trading application. Yet, it also fits very nicely as the Web site solution for a neighborhood restaurant that simply wants to publish an online menu, hours of operation, and perhaps provide a Web-based table reservation or food delivery system. WebSphere's complete and versatile nature can sometimes be the source of confusion for people who are trying to make important decisions about platforms and developer toolkits for their business or departmental projects. So, the goal of this paper is to help you get started with understanding the technologies, tools, and offerings of the WebSphere platform.

Figure 1-1 on page 6 shows a high-level overview of the WebSphere platform.

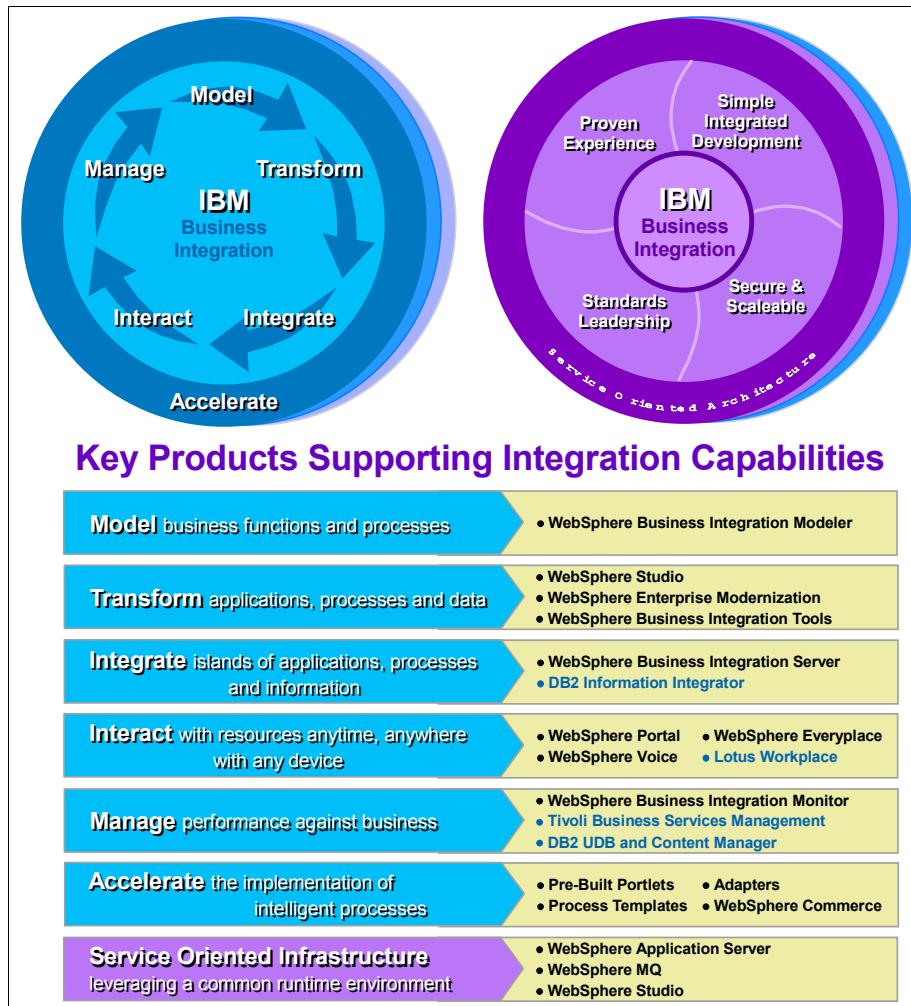


Figure 1-1 WebSphere Product family

1.3 WebSphere Application Servers

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

The primary component of the WebSphere Application Server products is the application server, which provides the environment to run your Web-enabled

e-business applications. You can think of an application server as *Web middleware*, the middle tier in a three-tier e-business environment. The first tier is the Web server that handles requests from the browser client. The third tier is the business database, for example DB2® UDB, and the business logic, for example, traditional business applications such as order processing. The middle tier is IBM WebSphere Application Server, which provides a framework for consistent, architected linkage between the HTTP requests and the business data and logic.

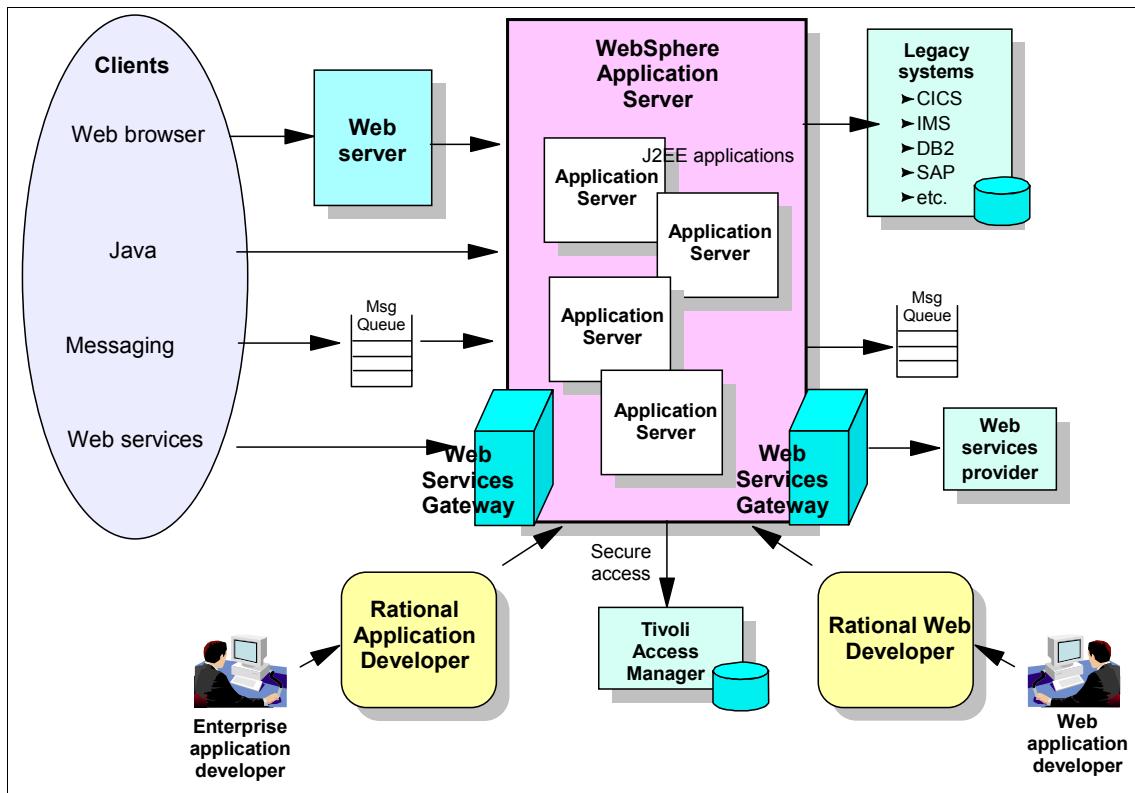


Figure 1-2 WebSphere Application Server product overview

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

WebSphere Application Servers are available on a wide range of platforms, including UNIX®-based platforms, Microsoft® operating systems, IBM z/OS®, and IBM @server® iSeries™.

1.4 WebSphere Application Server for distributed platforms

The latest product in the WebSphere Application Server family is IBM WebSphere Application Server V6. It features:

- ▶ Full J2EE 1.4 support
- ▶ High-performance connectors to many common back-end systems
 - These connectors reduce the coding effort required to link dynamic Web pages to real line-of-business data.
- ▶ Application services for session and state management
- ▶ Web services
 - Web services enable businesses to connect applications to other business applications, deliver business functions to a broader set of clients and partners, interact with marketplaces more efficiently, and create new business models dynamically.
- ▶ A fully integrated JMS 1.1 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.
- ▶ Many of the programming model extensions (PMEs) available in WebSphere Business Integration Server Foundation

1.4.1 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

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

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. The development tool included is a trial version of Rational Application Developer, full J2EE 1.4 compliant development tool.

To avoid confusion with the Express package in this document, 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 customer 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.

Packaging summary

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

Table 1-1 WebSphere Application Server packaging

Platform	Express V6 ¹	Base V6	ND V6
WebSphere Application Server	Yes	Yes	Yes
Deployment manager	No	No	Yes
IBM HTTP Server V6 Web server plug-ins	Yes	Yes	Yes
IBM HTTP Server	Yes	Yes	Yes
Application Client (not on zLinux)	Yes	Yes	Yes
Application Server Toolkit	Yes	Yes	Yes
DataDirect Technologies JDBC Drivers for WebSphere Application Server	Yes	Yes	Yes
Development tools	Rational Web Developer (single use license)	Rational Application Developer Trial	Rational Application Developer Trial
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

Platform	Express V6 ¹	Base V6	ND V6
Production ready applications	IBM Business Solutions	No	No
Tivoli® Directory Server for WebSphere Application Server (LDAP server)	No	No	Yes
Tivoli Access Manager Servers for WebSphere Application Server	No	No	Yes
Edge Components	No	No	Yes
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.

1.4.2 System requirements and support for distributed platforms

Note: The information in this section was current at the time this IBM Redbooks was published. For a current list of supported operating system levels and requirements, see the WebSphere Application Server Web site:

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

WebSphere Application Server V6 is supported on the distributed platforms shown in Table 1-2.

Table 1-2 WebSphere Application Server features

Operating system	Express V6	Base V6	ND V6
AIX®	Yes	Yes	Yes
H-UX	Yes	Yes	Yes
Linux® (Intel®)	Yes	Yes	Yes
Linux on iSeries	Yes	Yes	Yes
Linux on pSeries®	Yes	Yes	Yes
Linux on zSeries®	No	Yes	Yes

Operating system	Express V6	Base V6	ND V6
Solaris	Yes	Yes	Yes
Windows®	Yes	Yes	Yes

WebSphere Application Server V6 supports the following database servers:

- ▶ Cloudscape™
- ▶ IBM DB2
- ▶ Informix® Dynamic Server
- ▶ Oracle
- ▶ Microsoft SQL Server
- ▶ Sybase Adaptive Server Enterprise

WebSphere Application Server currently supports the following Web servers:

- ▶ Apache Server
- ▶ IBM HTTP Server
- ▶ Microsoft Internet Information Services
- ▶ Lotus® Domino® Enterprise Server
- ▶ Sun Java System Web Server
- ▶ Covalent Enterprise Ready Server

1.4.3 New for V6

WebSphere Application Server V6 continues with the tradition of providing support for the current J2EE specifications. In addition, it focuses on features that provide ease-of-use, simplification of application development and deployment, high availability, and flexibility. The following sections give you the highlights of the new features and functionality provided with WebSphere Application Server V6.

Programming support

The following are highlights of the new application programming features for WebSphere Application Server V6:

J2EE 1.4 support

WebSphere Application Server V6 provides full support for J2EE 1.4. The J2EE specification requires a certain set of specifications to be supported. Among these are EJB 2.1, JMS 1.1, JCA 1.5, Servlet 2.4, and JSP 2.0.

WebSphere Application Server V6 also provides support for J2EE 1.2 and 1.3 to ease migration.

WebSphere Application Server V6 is shipped with JDK 1.4.2, which includes the new Java Web Start feature. Java Web Start is an application-deployment technology that includes the portability of applets, the maintainability of servlets and JavaServer Pages (JSP) file technology, and the simplicity of mark-up languages such as XML and HTML. It is a Java application that allows full-featured Java 2 client applications to be launched, deployed and updated from a standard Web server.

Web services

Web services support has been updated to include the latest in technology options, including:

- ▶ *Java API for XML-based RPC (JAX-RPC) 1.1* enables you to develop SOAP-based interoperable and portable Web services and Web service clients. The JAX-RPC programming model is defined by the Web services standard JSR 101.
- ▶ *Web services for Java 2 Platform, Enterprise Edition* defines the programming model and run-time architecture for implementing Web services based on the Java language.JSR 109 - WSEE.
- ▶ *SOAP with Attachments API for Java (SAAJ) 1.2* is used for SOAP messaging that works behind the scenes in the JAX-RPC implementation.
- ▶ *Web Services Security (WS-Security)* proposes a standard set of SOAP extensions that you can use to build secure Web services.
- ▶ *Web Services-Interoperability (WS-I) Basic Profile 1.1* is a set of non-proprietary Web services specifications that promote interoperability. The *Web Services-Interoperability (WS-I) Attachments Profile* compliments the WS-I Basic Profile 1.1 to add support for interoperable SOAP messages with attachments-based Web services.
- ▶ *Java API for XML Registries (JAXR) 1.0* defines a Java client API for accessing both UDDI (Version 2 only) and ebXML registries.
- ▶ *Universal Description, Discovery and Integration (UDDI) V3* defines a way to publish and discover information about Web services.

In addition, WebSphere Application Server V6 adds value to the standards in these ways:

- ▶ Custom bindings to supplement JAX-RPC features, allowing you to create your own custom bindings to map Java to XML and XML to Java conversions.
- ▶ Support for generic SOAP elements
- ▶ Multi-protocol support for a stateless session Enterprise JavaBean (EJB) as the Web service provider for enhanced performance without changes to JAX-RPC clients

- ▶ Caching for Web services clients running in the application server, including the Web Services Gateway, in addition to the server-side Web service caching for Web services

The private UDDI Registry previously shipped with the V5 Network Deployment package, is now available in all packages and implements V3.0 of the UDDI specification.

And finally, the Web Services Gateway, in the Network Deployment package only, has been fully integrated into the application server. The function of the Web Services Gateway is provided by the service integration features new to V6. Installation is automatic when you install WebSphere Application Server. Administration is fully integrated within the administration tools.

Service Data Objects (SDO)

SDO, formerly WebSphere Data Objects, provide unified data access and representation across heterogeneous data stores. With SDO, data mediators perform the real work of accessing the data stores. Clients query a data mediator service and get a data graph in response. The data graph consists of structured data objects representing the data store. Clients update the data graph and send it back to the mediator service to have the updates applied.

SDO is not intended to replace other data access technologies, but rather to provide an alternate choice. It has the advantage of simplifying the application programming tasks required to access data stores.

SDO support is included in WebSphere Studio Application Developer 5.1.1 and in Rational Application Developer 6.0. This support includes:

- ▶ Wizards and views for working with data objects
- ▶ Relational data lists
- ▶ Relational data objects

WebSphere Application Server 6.0 support for SDO includes:

- ▶ Support for SDO naming and packaging
- ▶ Externalization of the following APIs
- ▶ SDO Core APIs
- ▶ EJB Mediator for entity EJBs
- ▶ JDBC Data Mediator for relational databases supported by WebSphere Application Server

SDO is defined by JSR 235. For more information, see:

<ftp://www6.software.ibm.com/software/developer/library/j-commonj-sdowmt/Commonj-SDO-Specification-v1.0.doc>

JavaServer Faces (JSF) v1.0

JavaServer Faces (JSF) is a user interface framework or API that eases the development of Java based Web applications. JSF makes J2EE more approachable to non-Java application developers with HTML, scripting, and page layout skills.

WebSphere Application Server version 6.0 supports JavaServer Faces 1.0 at a runtime level.

Programming Model Extensions (PMEs)

PMEs formerly part of more advanced WebSphere Application Server packaging options are now available in the Express, Base, and Network Deployment packages:

- ▶ Last Participant Support
- ▶ Internationalization Service
- ▶ WorkArea Service
- ▶ ActivitySession Service
- ▶ Extended JTA Support
- ▶ Startup Beans
- ▶ Asynchronous Beans
- ▶ Scheduler Service (Timer Service)
- ▶ Object Pools
- ▶ Dynamic Query
- ▶ Application Profiling

System management

WebSphere Application Server V6 has enhanced the usability of the WebSphere administration tools and has introduced features for managing multiple instances of WebSphere. There is also a focus on enhanced application deployment features.

The following sections highlight of the new system management features for WebSphere Application Server V6.

Improved system management model

Several improvements have been made to the basic system management features of WebSphere Application Server V6:

- ▶ Mixed cell support enables you to migrate an existing WebSphere Application Server V5 Network Deployment environment to V6. By migrating the Deployment Manager to V6 as a first step, you can continue to run V5 application servers until you can migrate each of them.

- ▶ Configuration archiving allows you to create a complete or partial archive of an existing WebSphere Application Server configuration. This archive is portable and can be used to create new configurations based on the archive.
- ▶ Defining a WebSphere Application Server V6 instance by a profile allows you to easily configure multiple runtimes with one set of install libraries.. After installing the product, you create the runtime environment by building profiles.
- ▶ Defining a generic server as an application server instance in the administration tools allows you to associate it with a non-WebSphere server or process that is needed to support the application server environment.
- ▶ By defining external Web servers as managed servers, you can start and stop the Web server and automatically push the plug-in configuration to it. This requires a node agent to be installed on the machine and is typically used when the Web server is behind a firewall.

You can also define a Web server as an unmanaged server for placement outside the firewall. This allows you to create custom plug-ins for the Web server, but you must manually move the plug-in configuration to the Web server machine.

As a special case, you can define the IBM HTTP server as an unmanaged server, but treat it as a managed server. This does not require a node agent because the commands are sent directly to the IBM HTTP server administration process.

- ▶ You can use node groups to define a boundary for server cluster formation. With WebSphere Application Server V6, you can now have nodes in cells with different capabilities, for example, a cell can contain both WebSphere Application Server on distributed systems and on z/OS. Node groups are created to group nodes of similar capability together to allow validation during system administration processes.

Administrative console updates

The administrative console has been updated with ease of use in mind. New panels have been added to facilitate the new V6 features such as service integration, the integrated UDDI Registry and Web Services Gateway, and the new Web server options. The navigation has been reworked to reduce the number of clicks required to reach most configuration settings.

The Tivoli Performance View monitor has also been integrated into the administrative console.

Application management

Improvements in application management techniques have been added to facilitate rapid application deployment and efficient update procedures.These improvements include the following items:

- ▶ Enhanced Enterprise Archive (EAR) files can now be built using Rational Application Developer or the Application Server Toolkit. The Enhanced EAR contains bindings and server configuration settings previously done at deployment time. This allows developers to predefine known runtime settings and can speed up deployment.
- ▶ Fine grain application update capabilities allow you to make small delta changes to applications without doing a full application update and restart.
- ▶ WebSphere Rapid Deployment provides the ability for developers to use annotation based programming. This is step forward in the automation of application development and deployment.

Service integration

The service integration functionality within WebSphere Application Server V6 supports both message-oriented and service-oriented applications. The primary component of this functionality is the service integration bus, which provides the support for messaging and Web services applications. One or more application servers or clusters join a bus to become bus members. The service integration bus becomes a component of the Enterprise Service Bus (ESB).

The service integration functionality provides:

- ▶ A fully compliant J2EE 1.4 JMS messaging provider, integrated within the application server. This messaging provider is the default messaging provider for the application server. It can support multi-server configurations and can be linked to WebSphere MQ, appearing as a queue manager. This new JMS provider replaces the embedded JMS provider available in WebSphere Application Server V5.
- ▶ An integrated Web services infrastructure and support for the Web Services Gateway, which provides you with a single point of control, access and validation of Web service requests, and allows you to control which Web services are available to different groups of Web service users.

The service integration bus is fully integrated with the administration tools, WebSphere security, installation processes, and provides support for clustering enablement.

Clustering enhancements

New enhancements in the area of clustering, failover and workload management include:

- ▶ Failover of stateful session EJBs is now possible. Each EJB container provides a method for stateful session beans to fail over to other servers. This feature uses the same memory-to-memory replication provided by the data replication services component used for HTTP session persistence.

- ▶ A new High Availability Manager has been added with the intent of eliminating single points of failure. It is responsible for running key services on available application servers, rather than on a dedicated one such as the deployment manager. The High Availability Manager takes advantage of fault tolerant storage technologies such as Network Attached Storage (NAS), significantly lowering the cost and complexity of high availability configurations. It also offers hot standby and peer failover for critical services.
- ▶ To simplify use, we made improvements to the administration of clusters.

Security enhancements

Updates to security features in WebSphere Application Server V6 include:

- ▶ Java Authorization Contract with Containers (JACC) 1.0 support details the contract requirements for J2EE containers and authorization providers. With this detail, authorization providers can perform the access decisions for resources in J2EE 1.4 application servers such as WebSphere Application Server. This support facilitates the plug-in of third-party authorization servers.
- ▶ WebSphere Application Server V6 provides an embedded IBM Tivoli Access Manager (TAM) client that is JACC compliant. The TAM can be used to access a Tivoli Access Manager server for authentication and authorization.
- ▶ Tivoli Access Manager (TAM) server is bundled in the Network Deployment package.



WebSphere Application Server V6 architecture

WebSphere Application Server is the implementation by IBM of the Java 2 Enterprise Edition (J2EE) platform. It conforms to the J2EE 1.4 specification. WebSphere Application Server is available in three unique packages that are designed to meet a wide range of client requirements. At the heart of each package is a WebSphere Application Server that provides the runtime environment for enterprise applications.

This discussion centers on the runtime server component of the following packaging options of WebSphere Application Server for distributed platforms:

- ▶ IBM WebSphere Application Server - Express V6, referred to as *Express*
- ▶ IBM WebSphere Application Server V6, referred to as *Base*
- ▶ IBM WebSphere Application Server Network Deployment V6, referred to as *Network Deployment*

2.1 Application server configurations

At the heart of each member of the WebSphere Application Server family is an application server. Each family has essentially the same architectural structure. Although the application server structure for Base and Express is identical, there are differences in licensing terms, the provided development tool, and platform support. With Base and Express, you are limited to stand-alone application servers. Each stand-alone application server provides a fully functional J2EE 1.4 environment.

Network Deployment has additional elements that allow for more advanced topologies such as workload management, scalability, high availability, and central management of multiple application servers.

2.1.1 Stand-alone server configuration

Express, Base, and Network Deployment all support a single stand-alone server environment. With a stand-alone configuration, each application server acts as a unique entity. An application server runs one or more J2EE applications and provides the services required to run those applications.

Multiple stand-alone application servers can exist on a machine, either through independent installations of the WebSphere Application Server code or through multiple configuration profiles within one installation. However, WebSphere Application Server does not provide for common management or administration for multiple application servers. Stand-alone application servers do not provide workload management or failover capabilities.

Figure 2-1 on page 21 shows an architectural overview of a stand-alone application server.

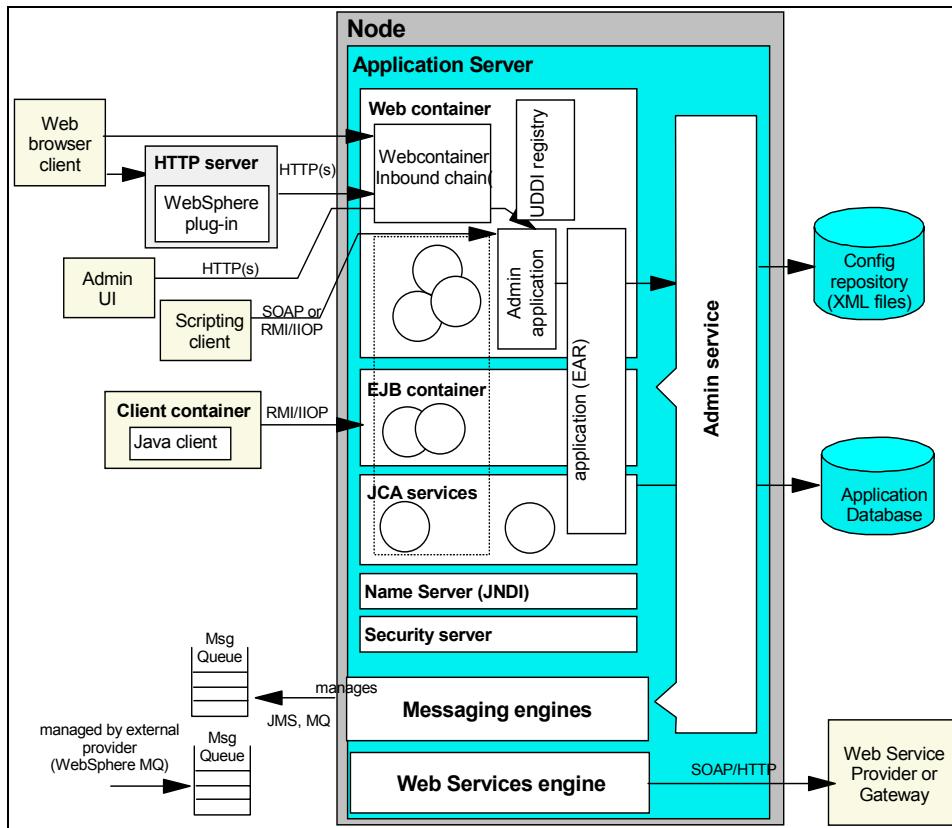


Figure 2-1 Architectural overview for a stand-alone server

2.1.2 Distributed server configuration

With Network Deployment, you can build a distributed server configuration, which enables central administration, workload management, and failover. In this environment, you integrate one or more application servers into a cell that is managed by a deployment manager. The application servers can reside on the same machine as the deployment manager or on multiple separate machines. Administration and management are handled centrally from the administration interfaces through the deployment manager.

With this configuration, you can create multiple application servers to run unique sets of applications and then manage those applications from a central location. More importantly, you can cluster application servers to allow for workload management and failover capabilities. Applications that you install in the cluster

are replicated across the application servers. When one server fails, another server in the cluster continues processing. Workload is distributed among Web containers and EJB containers in a cluster using a weighted round-robin scheme.

Figure 2-2 illustrates the basic components of an application server in a distributed server environment.

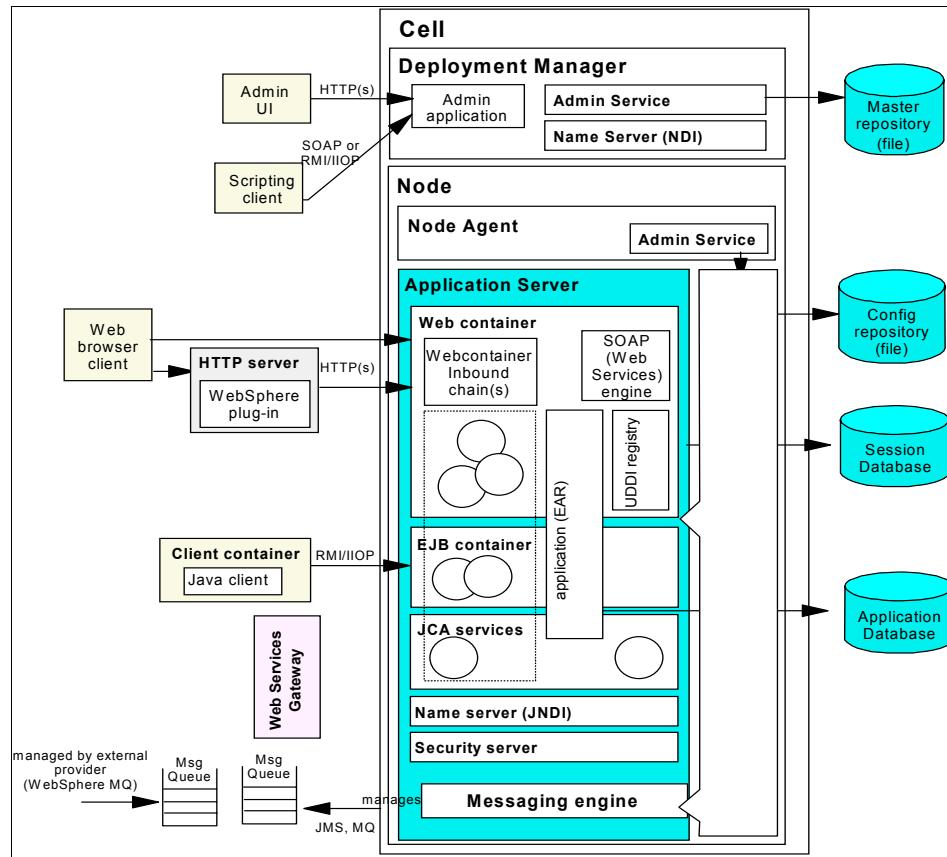


Figure 2-2 Distributed server environment

2.2 Application servers, nodes, and cells

Regardless of the configuration, the WebSphere Application Server is organized based on the concept of cells, nodes, and servers. While all of these elements are present in each configuration, cells and nodes do not play an important role until you take advantage of the features provided with Network Deployment.

2.2.1 Application servers

The application server is the primary runtime component in all configurations. It is where an application executes. All WebSphere Application Server configurations can have one or more application servers. In the Express and Base configurations, each application server functions as a separate entity. There is no workload distribution or common administration among application servers. With Network Deployment, you can build a distributed server environment consisting of multiple application servers maintained from a central administration point. In a distributed server environment, you can cluster application servers for workload distribution.

2.2.2 Nodes, node groups, and node agents

A *node* is a logical grouping of server processes managed by WebSphere and that share common configuration and operation control. A node is associated with one physical installation of WebSphere Application Server. In a stand-alone application server configuration, there is only one node.

With Network Deployment, you can configure multiple nodes to manage from one common administration server. In these centralized management configurations, each node has a node agent that works with a deployment manager to manage administration processes.

A node group is a new concept introduced with WebSphere Application Server V6. A *node group* is a grouping of nodes within a cell that have similar capabilities. A node group validates that the node is capable of performing certain functions before allowing those functions. For example, a cluster cannot contain both z/OS nodes and non-z/OS nodes. In this case, you can define two node groups, one for the z/OS nodes and one for nodes other than z/OS. A DefaultNodeGroup is automatically created based on the deployment manager platform. This node group contains the deployment manager and any new nodes with the same platform type.

2.2.3 Cells

A *cell* is a grouping of nodes into a single administrative domain. In the Base and Express configurations, a cell contains one node. That node might have multiple servers, but the configuration files for each server are stored and maintained individually.

In a distributed server configuration, a cell can consist of multiple nodes which are all administered from a single point. The configuration and application files for all nodes in the cell are centralized into a cell master configuration repository.

The deployment manager process manages the central repository and synchronizes with local copies that are held on each of the nodes.

2.3 Servers

WebSphere Application Server supplies application servers. They provide the functions required to host applications. WebSphere Application Server also provides the ability to define external servers to the administration process. Table 2-1 shows which types of servers you can define to the WebSphere Application Server administration tools.

Table 2-1 WebSphere Application Server server support

Server type	Express and Base	Network Deployment
Application server	Yes	Yes
Application server clustering	No	Yes
External Web server	Yes	Yes
External generic server	No	Yes
WebSphere V5 JMS servers	No	Yes

2.3.1 Application servers

Application servers provide the runtime environment for application code. They provide containers and services that specialize in enabling the execution of specific Java application components. Each application server runs in its own Java Virtual Machine (JVM).

2.3.2 Clusters

With Network Deployment, you can use application server clustering to enhance workload distribution. A *cluster* is a logical collection of application server processes that provides workload balancing and high availability.

Application servers in a cluster are members of that cluster and must all have identical application components on them. Other than the applications on them, cluster members do not have to share any other configuration data.

For example, one cluster member might run on a large multi-processor server while another member of that same cluster might run on a small mobile computer. The server configuration settings for each of these two cluster

members is very different, except the application components that are assigned to them. In that area of configuration, they are identical.

The members of a cluster can be located on a single node (*vertical cluster*), across multiple nodes (*horizontal cluster*), or on a combination of the two.

When you install, update, or delete an application, the updates are automatically distributed to all members in the cluster. In WebSphere Application Server V5, if you updated an application on a cluster, you had to stop the application on every server in the cluster, install the update, and then restart the server. With WebSphere Application Server V6, the Rollout Update option allows you to update and restart the application servers on each node, one node at a time. This provides continuous availability of the application.

2.3.3 JMS servers (V5)

In WebSphere Application Server V5, JMS servers provide the default messaging support for WebSphere Application Server. For migration purposes, Network Deployment in V6 supports cells that contain both V5 and V6 nodes (the deployment manager must be at V6), and by extension, Network Deployment supports existing JMS servers in V5 application servers in the cell.

2.3.4 External servers

You can define servers other than WebSphere application servers to the administrative process. You can define:

- ▶ Generic servers

A *generic* server is a server that is managed in the WebSphere administrative domain, but is not a server that is supplied by WebSphere Application Server. The generic server can be any server or process that is necessary to support the application server environment, including a Java server, a C or C++ server or process, a CORBA server or a Remote Method Invocation server.

- ▶ Web servers

Web servers can be defined to the administration process as Web server nodes, allowing applications to be associated with one or more defined Web servers.

Web server nodes can be managed or unmanaged. *Managed* nodes have a node agent on the Web server machine that allows the deployment manager to administer the Web server. You can start or stop the Web server from the deployment manager, generate the Web server plug-in for the node, and automatically push it to the Web server. Managed Web server nodes are usually behind the firewall with WebSphere Application Server installations.

Unmanaged Web server nodes, as the name implies, are not managed by WebSphere. You normally find these outside the firewall, or in the demilitarized zone. You must manually copy or FTP Web server plug-in files to the Web server. However, if you define the Web server as a node, you can generate custom plug-in files for it.

Note: As a special case, if the unmanaged Web server is an IBM HTTP Server, you can administer the Web server from the WebSphere administrative console. Then, you can automatically push the plug-in configuration file to the Web server with the deployment manager using HTTP commands to the IBM HTTP Server administration process. This configuration does not require a node agent.

2.4 Containers

The J2EE 1.4 specification defines the concept of containers to provide runtime support for applications. There are two types of containers in the application server implementation:

- ▶ A Web container, which processes HTTP requests, servlets, and JavaServer Pages (JSPs)
- ▶ An EJB container, which processes Enterprise JavaBeans (EJBs)

In addition, there is an application client container that can run on the client machine. Table 2-2 shows the containers that each packaging option supports.

Table 2-2 WebSphere Application Server container support

Container type	Express and Base	Network Deployment
Web container	Yes	Yes
EJB container	Yes	Yes
Application client container	Yes	Yes

2.4.1 Web container

The Web container processes servlets, JSP files, and other types of server-side includes. Each application server runtime has one logical Web container, which can be modified, but not created or removed. Each Web container provides the following:

- ▶ Web container transport chains

Requests are directed to the Web container using the Web container inbound transport chain. The chain consists of a TCP inbound channel that provides the connection to the network, an HTTP inbound channel that serves HTTP 1.0 and 1.1 requests, and a Web container channel over which requests for servlets and JSPs are sent to the Web container for processing.

- ▶ **Servlet processing**

When handling servlets, the Web container creates a request object and a response object, then invokes the servlet service method. The Web container invokes the servlet's destroy method when appropriate and unloads the servlet, after which the JVM performs garbage collection.

- ▶ **HTML and other static content processing**

Requests for HTML and other static content that are directed to the Web container are served by the Web container inbound chain. However, in most cases, using an external Web server and Web server plug-in as a front-end to a Web container is more appropriate for a production environment.

- ▶ **Session management**

Support is provided for the javax.servlet.http.HttpSession interface as described in the Servlet application program interface (API) specification.

- ▶ **Web services engine**

Web services are provided as a set of APIs in cooperation with the J2EE applications. Web services engines are provided to support Simple Object Access Protocol (SOAP).

Web server plug-ins

Although the Web container can serve static content, a more likely scenario is that you will use an external Web server to receive client requests. The Web server can serve requests that do not require any dynamic content, for example, HTML pages. However, when a request requires dynamic content, such as JSP or servlet processing, it must be forwarded to WebSphere Application Server for handling.

To forward a request, you use a Web server plug-in that is included with the WebSphere Application Server packages for installation on a Web server. You copy an Extensible Markup Language (XML) configuration file, located on the WebSphere Application Server, to the Web server plug-in directory. The plug-in uses the configuration file to determine whether a request should be handled by the Web server or an application server. When WebSphere Application Server receives a request for an application server, it forwards the request to the appropriate Web container in the application server. The plug-in can use HTTP or HTTPS to transmit the request.

2.4.2 Enterprise JavaBeans container

The Enterprise JavaBeans (EJB) container provides all the runtime services that are needed to deploy and manage enterprise beans. It is a server process that handles requests for both session and entity beans.

The enterprise beans, packaged in EJB modules, installed in an application server do not communicate directly with the server. Instead, the EJB container provides an interface between the enterprise beans and the server. Together, the container and the server provide the enterprise bean runtime environment.

The container provides many low-level services, including threading and transaction support. From an administrative viewpoint, the container manages data storage and retrieval for the contained enterprise beans. A single container can host more than one EJB Java archive (JAR) file.

2.4.3 Application client container

The application client container is a separately installed component on the client's machine. It allows the client to run applications in a J2EE environment that is compatible with EJB.

To launch the application client along with its client container runtime, execute the following command: `LaunchClient`.

2.5 Application server services

The application server provides services in addition to the containers, as shown in Table 2-3 on page 29.

Table 2-3 WebSphere Application Server services

Service	Express and Base	Network Deployment
J2EE Connector Architecture services	Yes	Yes
Transaction service	Yes	Yes
Dynamic cache service	Yes	Yes
Message listener service	Yes	Yes
Object Request Broker service	Yes	Yes
Administrative service (Java Management Extensions)	Yes	Yes
Diagnostic trace service	Yes	Yes
Debugging service	Yes	Yes
Name service (Java Naming Directory Interface)	Yes	Yes
Performance Monitoring Interface service	Yes	Yes
Security service (JAAS and Java 2 security)	Yes	Yes
Service Integration Bus service	Yes	Yes

For further information, see the sections starting with 2.5.1, “J2EE Connector Architecture services” on page 30.

Table 2-4 shows the services that are provided to support the programming model extensions (PMEs).

Table 2-4 services that support programming model extensions

PME support service	Express and Base	Network Deployment
Application profiling service	Yes	Yes
Compensation service	Yes	Yes
Internationalization service	Yes	Yes
Object pool service	Yes	Yes
Startup beans service	Yes	Yes
Activity session service	Yes	Yes
Work area partition service	Yes	Yes

PME support service	Express and Base	Network Deployment
Work area service	Yes	Yes

The sections that follow discuss the WebSphere Application Server services that are listed in Table 2-3 on page 29.

2.5.1 J2EE Connector Architecture services

Connection management for access to enterprise information systems (EIS) in WebSphere Application Server is based on the J2EE Connector Architecture (JCA) specification, also sometimes referred to as J2C. The connection between the enterprise application and the EIS is done through the use of EIS-provided resource adapters, which are plugged into the application server. The architecture specifies the connection management, transaction management, and security contracts that exist between the application server and the EIS.

Within the application server, the Connection Manager pools and manages connections. The Connection Manager administers connections that are obtained through both resource adapters defined by the JCA specification and data sources defined by the JDBC 2.0 Extensions, and later, specification.

2.5.2 Transaction service

WebSphere applications use transactions to coordinate multiple updates to resources as one unit of work such that all or none of the updates are made permanent. Transactions are started and ended by applications or the container in which the applications are deployed.

WebSphere Application Server is a transaction manager that supports the coordination of resource managers through the XAResource interface and participates in distributed global transactions with transaction managers that support the CORBA Object Transaction Service protocol (for example, application servers) or the Web Service Atomic Transaction protocol.

WebSphere Application Server also participates in transactions imported through J2EE Connector 1.5 resource adapters. You can also configure WebSphere applications to interact with, or direct the WebSphere transaction service to interact with, databases, Java Message Service (JMS) queues, and JCA connectors through their local transaction support when distributed transaction coordination is not required.

How applications use transactions depends on the type of application component, for example:

- ▶ A session bean can either use container-managed transactions where the bean delegates management of transactions to the container, or bean-managed transactions where the bean manages transactions itself.
- ▶ Entity beans use container-managed transactions.
- ▶ Web components, or servlets, use bean-managed transactions.

WebSphere Application Server handles transactions with three main components:

- ▶ A transaction manager supports the enlistment of recoverable XAResources and ensures that each such resource is driven to a consistent outcome, either at the end of a transaction, or after a failure and restart of the application server.
- ▶ A container in which the J2EE application runs manages the enlistment of XAResources on behalf of the application when it performs updates to transactional resource managers such as databases. Optionally, the container can control the demarcation of transactions for enterprise beans that are configured for container-managed transactions.
- ▶ A UserTransaction API handles bean-managed enterprise beans and servlets. UserTransaction allows such application components to control the demarcation of their own transactions.

2.5.3 Dynamic cache service

The dynamic cache service improves performance by caching the output of servlets, commands, Web services, and JSP files. The dynamic cache works within an application server, intercepting calls to objects that can be cached, for example, through a servlet's service() method or a command's execute() method. The dynamic cache either stores the object's output to or serves the object's content from the dynamic cache.

Because J2EE applications have high read-write ratios and can tolerate small degrees of latency in the currency of their data, the dynamic cache can create significant gains in server response time, throughput, and scalability.

The following caching features are available in WebSphere Application Server:

- ▶ Cache replication

Cache replication among cluster members takes place using the WebSphere data replication service. Data is generated one time and then copied or replicated to other servers in the cluster, saving execution time and resources.

- ▶ Cache disk offload

By default, when the number of cache entries reaches the configured limit for a given WebSphere server, eviction of cache entries occurs, allowing new entries to enter the cache service. The dynamic cache includes a disk offload feature that copies the evicted cache entries to disk for potential future access.
- ▶ Edge Side Include caching

The Web server plug-in contains a built-in Edge Side Include (ESI) processor. The ESI processor caches whole pages, as well as fragments, providing a higher cache hit ratio. The cache implemented by the ESI processor is an in-memory cache, not a disk cache. Therefore, the cache entries are not saved when the Web server is restarted.
- ▶ External caching

The dynamic cache controls caches outside of the application server, such as that provided by the Edge components, an IBM HTTP Server's FRCA cache that is not z/OS, and a WebSphere HTTP Server plug-in ESI Fragment Processor that is not z/OS. When external cache groups are defined, the dynamic cache matches external cache entries with those groups and pushes out cache entries and invalidations to those groups. This external caching allows WebSphere to manage dynamic content beyond the application server. The content can then be served from the external cache, instead of the application server, improving performance.

2.5.4 Message listener service

With EJB 2.1, an ActivationSpec is used to connect message-driven beans to destinations. However, you can deploy existing EJB 2.0 message-driven beans against a listener port as in WebSphere Application Server V5. For those message-driven beans, the message listener service provides a listener manager that controls and monitors one or more JMS listeners. Each listener monitors a JMS destination on behalf of a deployed message-driven bean.

2.5.5 Object Request Broker service

An Object Request Broker (ORB) manages the interaction between clients and servers, using Internet Inter-ORB Protocol (IIOP). The ORB service enables clients to make requests and receive responses from servers in a network-distributed environment.

The ORB service provides a framework for clients to locate objects in the network and call operations on those objects as though the remote objects were located in the same running process as the client. The ORB service provides

location transparency. The client calls an operation on a local object, known as a *stub*. Then the stub forwards the request to the desired remote object, where the operation is run, and the results are returned to the client.

The client-side ORB is responsible for creating an IIOP request that contains the operation and any required parameters, and for sending the request in the network. The server-side ORB receives the IIOP request, locates the target object, invokes the requested operation, and returns the results to the client. The client-side ORB demarshals the returned results and passes the result to the stub, which returns the result to the client application, as though the operation had been run locally.

WebSphere Application Server uses an ORB to manage communication between client applications and server applications as well as communication among product components.

2.5.6 Administrative service

The administrative service runs within each server JVM. In Base and Express, the administrative service runs in the application server. In Network Deployment, each of the following hosts an administrative service:

- ▶ Deployment manager
- ▶ Node agent
- ▶ Application server

The administrative service provides the necessary functions to manipulate configuration data for the server and its components. The configuration is stored in a repository in the server's file system.

The administrative service has a security control and filtering functionality that provides different levels of administration to certain users or groups using the following administrative roles:

- ▶ Administrator
- ▶ Monitor
- ▶ Configurator
- ▶ Operator

2.5.7 Name service

Each application server hosts a name service that provides a Java Naming and Directory Interface (JNDI) name space. The service is used to register resources hosted by the application server. The JNDI implementation in WebSphere Application Server is built on top of a Common Object Request Broker Architecture (CORBA) naming service (CosNaming).

JNDI provides the client-side access to naming and presents the programming model that application developers use. 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 naming architecture is used by clients of WebSphere applications to obtain references to objects related to those applications. These objects are bound into a mostly hierarchical structure, referred to as a *name space*. The name space structure consists of a set of name bindings, each containing a name relative to a specific context and the object bound with that name. The name space can be accessed and manipulated through a name server.

The following are features of a WebSphere Application Server name space:

- ▶ **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 that is used for cell-scoped persistent bindings and a node-persistent root that is 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.

In a federated name space, the real location of each context is transparent to client applications. Clients have no knowledge that multiple name servers are handling resolution requests for a particular requested object.

In a Network Deployment distributed server configuration, the name space for the cell is federated among the deployment manager, node agents, and application servers of the cell. Each such server 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

You can use the configuration graphical interface and script interfaces to configure bindings in various root contexts within the name space. These bindings are read-only and are bound by the system at server startup.

- ▶ Support for CORBA Interoperable Naming Service (INS) object Uniform Resource Locator (URL)

WebSphere Application Server contains support for CORBA object URLs (corbaloc and corbaname) as JNDI provider URLs and lookup names.

Figure 2-3 summarizes the naming architecture and its components.

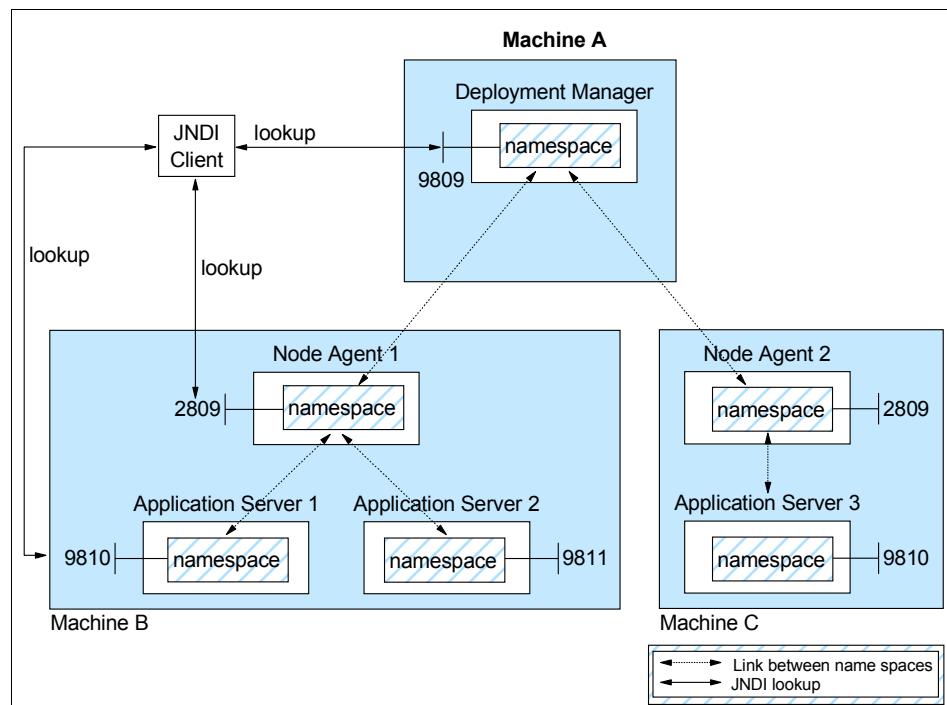


Figure 2-3 Naming topology

2.5.8 Performance Monitoring Infrastructure service

WebSphere Application Server collects data on runtime and applications through the Performance Monitoring Infrastructure (PMI). This infrastructure is compatible with and extends the JSR-077 specification.

PMI uses a client-server architecture. The server collects performance data from various WebSphere Application Server components and stores it in memory. This data consists of counters such as servlet response time and data connection pool usage. The data can then be retrieved using a Web client, Java client, or Java Management Extensions (JMX) client. WebSphere Application Server contains Tivoli Performance Viewer, which is integrated into the WebSphere administrative console and displays and monitors performance data.

WebSphere Application Server also collects data by timing requests as they travel through the product components. PMI request metrics log the time spent in major components, such as Web containers, EJB containers, and databases. These data points are recorded in logs and can be written to Application Response Time agents that Tivoli monitoring tools use.

2.5.9 Security service

Each application server JVM hosts a security service. The security service uses the security settings held in the configuration repository to provide authentication and authorization functionality.

2.6 Data Replication Service

The Data Replication Service (DRS) is responsible for replicating in-memory data among WebSphere processes. You can use DRS for:

- ▶ Stateful session EJB persistence and failover (new in V6.0)
- ▶ HTTP session persistence and failover
- ▶ Dynamic cache replication

Replication domains, consisting of server or cluster members that have a need to share internal data, perform the replication. Multiple domains can be used, each for a specific task among a set of servers or clusters. While HTTP session replication and EJB state replication can (and should) share a domain, you need a separate domain for dynamic cache replication.

You can define a domain so that each domain member has a single replicator that sends data to another domain member. You can also define a domain so that each member has multiple replicators that send data to multiple domain members.

WebSphere Application Server offers two topologies when setting up data replication among servers:

- ▶ Peer-to-peer topology

Each application server stores sessions in its own memory and retrieves sessions from other application servers. In other words, each application server acts as a *client* by retrieving sessions from other application servers. Each application server also acts as a *server* by providing sessions to other application servers. This mode, working in conjunction with the workload manager, provides hot failover capabilities.
- ▶ Client/server topology

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. 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 store sessions but do not respond to user requests.

2.7 Virtual hosts

A *virtual host* is a configuration that enables a single host machine to resemble multiple host machines. This configuration allows a single physical machine to support several independently configured and administered applications. A virtual host is not associated with a particular node. It is a configuration, rather than a live object, which is why you can create it but you cannot start or stop it.

Each virtual host has a logical name and a list of one or more Domain Name Server (DNS) aliases by which it is known. A DNS alias is the TCP/IP host name and port number that is used to request the servlet, for example, yourHostName:80. When a servlet request is made, the server name and port number entered into the browser are compared to a list of all known aliases in an effort to locate the correct virtual host and serve the servlet. If no match is found, an HTTP 404 error is returned to the browser.

WebSphere Application Server provides two default virtual hosts:

- ▶ default_host

This virtual host is used for accessing most applications. The default settings for default_host map to all requests for any alias on ports 80, 9443, and 9080. For example:

```
http://localhost:80/snoop  
http://localhost:9080/snoop
```

- ▶ admin_host

This virtual host is configured specifically for accessing the WebSphere Application Server administrative console. Other applications are not accessible through this virtual host. The default settings for admin_host map to requests on ports 9060 and 9043, as in `http://localhost:9060/admin`.

2.8 Session management

In many Web applications, users dynamically collect data as they move through the site based on a series of selections on the pages that they visit. Where the user goes and what the application displays can depend on what the user has chosen previously from the site. To maintain this data, the application stores it in a *session*.

WebSphere supports three approaches to track sessions:

- ▶ Secure Sockets Layer (SSL) session identifiers, where SSL session information is used to track the HTTP session ID.
- ▶ Cookies, where the application server session support generates a unique session ID for each user and returns this ID to the user's browser using a cookie. The default name for the session management cookie is JSESSIONID. Using cookies is the most common method of session management.
- ▶ URL rewriting

Session data can be kept in local memory cache, stored externally on a database, or kept in memory and replicated among application servers. Table 2-5 on page 38 shows the session support for each WebSphere Application Server configuration.

Table 2-5 WebSphere Application Server session management support

Session type	Express and Base	Network Deployment
Cookies	Yes	Yes
URL rewriting	Yes	Yes
SSL session identifiers	Yes	Yes
In memory cache	Yes	Yes
Session persistence using a database	Yes	Yes

Session type	Express and Base	Network Deployment
Memory-to-memory session persistence	No	Yes

The Servlet 2.4 specification defines the session scope at the Web application level, meaning that session information can only be accessed by a single Web application. However, there can be times when there is a logical reason for multiple Web applications to share information, sharing a user name, for example. WebSphere Application Server provides an IBM extension to the specification that allows 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. You specify this option during application assembling.

2.8.1 HTTP Session persistence

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, which is 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.

By default, WebSphere Application Server places session objects in memory. However, the administrator has the option of enabling persistent session management. This option instructs WebSphere to place session objects in a persistent store. Using a persistent store allows an application server to recover the user session data on restart or another cluster member after a cluster member in a cluster fails or is shut down. Two options for HTTP session persistence are available:

- ▶ Database

Session information is stored in a central session database for session persistence.

In a single-server environment, the session can be persisted when the user's session data must be maintained across a server restart or when the user's session data is too valuable to lose through an unexpected server failure.

In a multi-server environment, the multiple application servers hosting a particular application need to share this database information to maintain session states for the stateful components.

- ▶ Memory-to-memory using data replication services

In a Network Deployment distributed server environment, WebSphere internal replication enables sessions to be shared among application servers 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.

2.8.2 Stateful session EJB persistence

With WebSphere Application Server V6, you now have failover capability of stateful session EJBs. This function uses data replication services and interacts with the workload manager component during a failover situation.

2.9 Web services

Web services are self-contained, modular applications that can be described, published, located, and invoked over a network. WebSphere Application Server can act as both a Web service provider and as a requester. As a provider, it hosts Web services that are published for use by clients. As a requester, it hosts applications that invoke Web services from other locations.

WebSphere Application Server supports SOAP-based Web service hosting and invocation.

Table 2-6 WebSphere Application Server server support

Service	Express and Base	Network Deployment
Web services support	Yes	Yes
Private UDDI v3 Registry	Yes	Yes
Web Services Gateway	No	Yes
Enterprise Web services	Yes	Yes

Web services support includes the following:

- ▶ Web Services Description Language (WSDL), an XML-based description language, provides a way to catalog and describe services. WSDL describes

the interface of Web services (parameters and result), the binding (SOAP, EJB), and the implementation location.

- ▶ Universal Discovery Description and Integration (UDDI), a global platform-independent, open framework, enables businesses to discover each other, define their interaction, and share information in a global registry.

UDDI support in WebSphere Application Server V6 includes UDDI V3 APIs, some UDDI V1 and V2 APIs, UDDI V3 client for Java, and UDDI4J for compatibility with UDDI V2 registries. It also provides a UDDI V3 Registry that is integrated in WebSphere Application Server.
- ▶ SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment.
- ▶ XML is a common language for exchanging information.
- ▶ JAX-RPC (JSR 101) is the core programming model and bindings for developing and deploying Web services on the Java platform. It is a Java API for XML-based RPC and supports JavaBeans and enterprise beans as Web service providers.
- ▶ Enterprise Web services (JSR 109) adds EJBs and XML deployment descriptors to JSR 101.
- ▶ WS-Security is the specification that covers a standard set of SOAP extensions and can be used when building secure Web services to provide integrity and confidentiality. It is designed to be open to other security models including PKI, Kerberos, and SSL. WS-Security provides support for multiple security tokens, multiple signature formats, multiple trust domains, and multiple encryption technologies. It includes security token propagation, message integrity, and message confidentiality. The specification is proposed by IBM, Microsoft, and VeriSign for review and evaluation. In the future, WS-Security will replace existing Web services security specifications from IBM and Microsoft, including SOAP Security Extensions (SOAP-SEC), WS-Security and WS-License from Microsoft, as well as security token and encryption documents from IBM.
- ▶ JAXR is an API that standardizes access to Web services registries from within Java. JAXR 1.0 defines access to ebXML and UDDI V2 registries. WebSphere Application Server provides JAXR level 0 support, meaning that it supports UDDI registries.

JAXR does not map precisely to UDDI. For a precise API mapping to UDDI V2, IBM provides UDDI4J and IBM Java Client for UDDI V3.
- ▶ The SOAP with Attachments API for Java (SAAJ) is a standard for sending XML documents over the Internet from the Java platform.

IBM adds value: In addition to the requirements of the specifications, IBM has added the following features to its Web services support:

- ▶ Custom bindings

JAX-RPC does not support all XML schema types. Custom bindings allow developers to map Java to XML and XML to Java conversions.

- ▶ Support for generic SOAP elements

In cases where you want generic mapping, this support allows you to eliminate binding and use the generic SOAPElement type.

- ▶ Multi-protocol support

This features allows a stateless session EJB as the Web service provider, which provides enhanced performance without changes to the JAX-RPC client.

- ▶ Client caching

In WebSphere Application Server V5, there was support for server side Web service caching for Web services providers running within the application server. In addition to this server side caching, WebSphere Application Server V6 introduces caching for Web services clients running within a V6 application server, including the Web Services Gateway.

2.9.1 Enterprise services (JCA Web services)

Enterprise services offer access over the Internet to applications in a platform-neutral and language-neutral fashion. They offer access to enterprise information systems (EIS) and message queues and can be used in a client/server configuration without the Internet. Enterprise services can access applications and data on a variety of platforms and in a variety of formats.

An enterprise service wraps a software component in a common services interface. The software component is typically a Java class, EJB, or JCA resource adapter for an EIS. In services terminology, this software component is known as the implementation. Enterprise services primarily use WSDL and Web Services Invocation Framework (WSIF) to expose an implementation as a service.

Using the Integrated Edition of WebSphere Studio, you can turn Customer Information Control System (CICS®) and Information Management System (IMS™) transactions into Web services using JCA.

2.9.2 Web service client

Applications that invoke Web services are known as *Web service clients* or *Web service requestors*. An application that acts as a Web service client is deployed to WebSphere Application Server like any other enterprise application. No additional configuration or software is needed for the Web services client to function. Web services clients can also be stand-alone applications.

A Web service client binds to a Web service server to invoke the Web service. The binding is done using a service proxy, or stub, which is generated based on the WSDL document for the Web service. This service proxy contains all the information that is needed to invoke the Web service and is used locally by the clients to access the business service. The binding can also be done dynamically using WSIF.

2.9.3 Web service provider

An application that acts as a Web service is deployed to WebSphere Application Server like any other enterprise application. The Web services are contained in Web modules or EJB modules.

Publishing the Web service to a UDDI registry makes it available to anyone searching for it. Web services can be published to a UDDI registry using the Web Services Explorer provided with Rational Application Developer.

When using Rational Application Developer to package the application for deployment, no additional configuration or software is needed for the Web services client to function. The SOAP servlets are automatically added, and a SOAP administrative tool is included in a Web module.

If not, you can use the endptEnabler tool found in the WebSphere bin directory to enable the SOAP services within the Enterprise Application Archive (EAR) file and to add the SOAP administrative tool.

2.9.4 Enterprise Web Services

The Enterprise Web Services, based on the JSR 109 specification request, uses JAX-RPC in a J2EE environment that defines the runtime architecture as well as implements and deploys Web services in a generic J2EE server. The specification defines the programming model and architecture for implementing Web services in Java based on JSRs 67, 93, 101, and future JSRs related to Web services standards. You can find the list of JSRs at:

<http://www.jcp.org/en/jsr/all>

2.9.5 IBM WebSphere UDDI Registry

WebSphere Application Server V6 provides a private UDDI registry that implements V3.0 of the UDDI specification. This registry enables the enterprise to run its own Web services broker within the company or to provide brokering services to the outside world. The UDDI registry installation and management is now integrated in with WebSphere Application Server.

You can access the registry for inquiry and publish through the UDDI:

- ▶ SOAP API
- ▶ EJB client interface
- ▶ User console

You can use this Web-based graphical user interface to publish and inquire about UDDI entities. However, it provides only a subset of the UDDI API functions.

Security for the UDDI registry is handled using WebSphere security. To support the use of secure access with the IBM WebSphere UDDI Registry, you need to configure WebSphere to use HTTPS and SSL.

A relational database is used to store registry data.

2.9.6 Web Services Gateway

The Web Services Gateway bridges the gap between Internet and intranet environments during Web service invocations. The gateway builds upon the WSDL and the WSIF for deployment and invocation.

With WebSphere Application Server V6, the Web Services Gateway is fully integrated into the integration service technologies, which provides the runtime. The administration is done directly from the WebSphere administrative console.

The primary function of the Web Services Gateway is to map an existing WSDL-defined Web service, the target service, to a new service, the gateway service, that is offered by the gateway to others. The gateway thus acts as a proxy. Each target service, whether internal or external, is available at a service integration bus destination.

The role formerly played by filters in the V5 Web Services Gateway is now provided through JAX-RPC handlers. Using JAX-RPC handlers provides a standard approach for intercepting and filtering service messages. JAX-RPC handlers interact with messages as they pass in and out of the service integration bus. Handlers monitor messages at ports and take appropriate action, depending upon the sender and content of each message.

Exposing internal Web services to the outside world

A Web service hosted internally and made available through the service integration bus is called an *inbound service*. Inbound services are associated with a service destination. Service requests and responses are passed to the service through an endpoint listener and associated inbound port.

From the gateway's point of view, the inbound service is the target service. To expose the target service for outside consumption, the gateway takes the WSDL file for the inbound service and generates a new WSDL file that can be shared with outside requestors. The interface described in the WSDL is exactly the same. However, the service endpoint is changed to the gateway, which is now the official endpoint for the service client. Figure 2-4 diagrams the configuration for exposing Web services through a gateway.

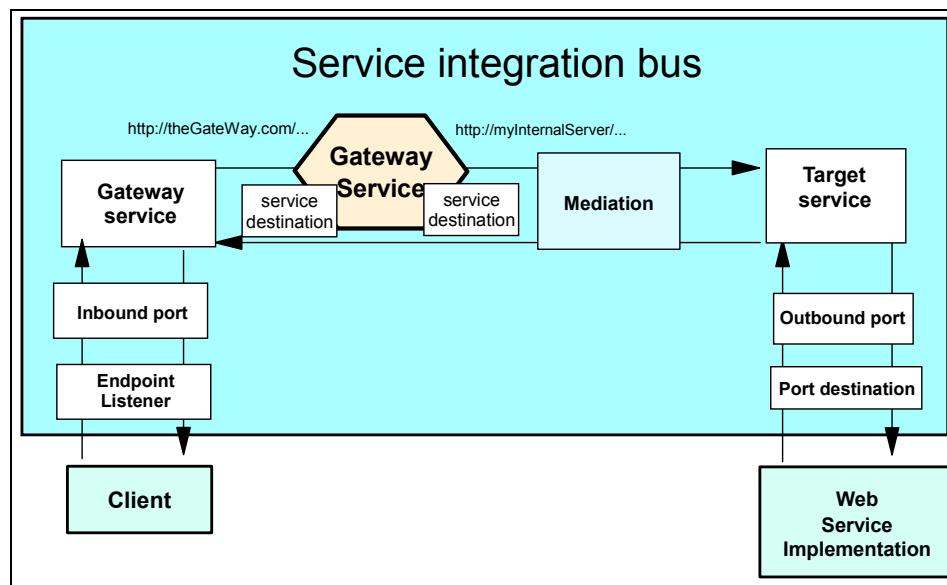


Figure 2-4 Exposing Web services through a gateway

Externally-hosted Web services

A Web service that is hosted externally and made available through the service integration bus is called an *outbound service*. To configure an externally-hosted service for a bus, do the following:

1. Associate it with a service destination.
2. Configure one or more port destinations, one for each type of binding (for example, SOAP over HTTP or SOAP over JMS) through which service requests and responses are passed to the external service.

From the gateway's point of view, the outbound service is the target service. Mapping a gateway service to the target service allows internal service requestors to invoke the service as though it were running on the gateway. Again, a new WSDL is generated by the gateway that shows the same interface but that names the gateway as service provider rather than the real internal server. All requests to the gateway service are rerouted to the implementation specified in the original WSDL.

Of course, every client could access external Web services by traditional means, but if you add the gateway as an additional layer in between, clients do not have to change anything if the service implementor changes. This scenario is very similar to that shown in Figure 2-4 on page 45. The difference is the Web service implementation is located at a site on the Internet.

UDDI publication and lookup

The gateway facilitates working with UDDI registries. As you map a service for external consumption using the gateway, you can publish the exported WSDL in the UDDI registry. When the services in the gateway are modified, the UDDI registry is updated with the latest changes.

2.10 Service integration bus

The service integration bus provides the communication infrastructure for messaging and service-oriented applications, thus unifying this support into a common component. The service integration bus is a JMS provider that is JMS 1.1 compliant for reliable message transport and that has the capability of intermediary logic to adapt message flow intelligently in the network. It also supports the attachment of Web services requestors and providers. Service integration bus capabilities have been fully integrated within WebSphere Application Server, enabling it to take advantage of WebSphere security, administration, performance monitoring, trace capabilities, and problem determination tools.

The service integration bus is often referred to as just a bus. When used to host JMS applications, it is also often referred to as a *messaging bus*.

Figure 2-5 on page 47 illustrates the service integration bus and how it fits into the larger picture of an Enterprise Service Bus.

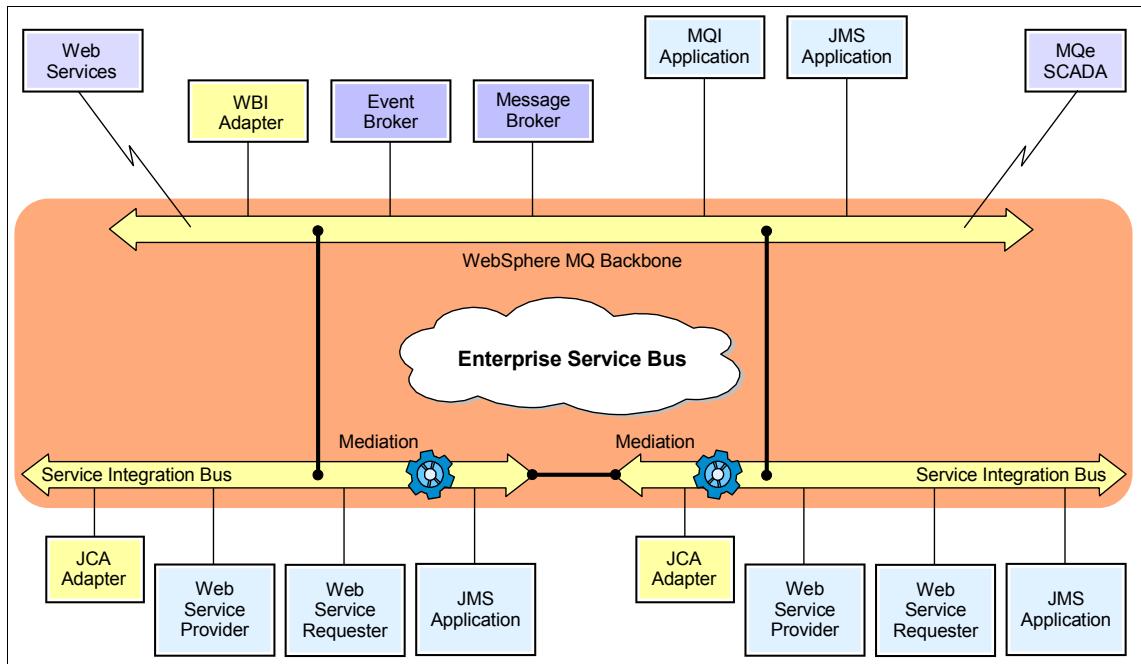


Figure 2-5 The Enterprise Service Bus

A service integration bus consists of the following:

- ▶ Bus members
Application servers or clusters that have been added to the bus.
- ▶ Messaging engine

The application server or cluster component that manages bus resources. When a bus member is defined, a messaging engine is automatically created on the application server or cluster. The messaging engine provides a connection point for clients to produce or consume messages.

An application server has one messaging engine per bus of which it is a member. A cluster has at least one messaging engine per bus and can have more. In this case, the cluster owns the messaging engine and determines on which application server the messaging engine will run. You can have multiple messaging engines and application servers.

- ▶ Destinations

The place within the bus to which applications attach to exchange messages. Destinations can represent Web service endpoints, messaging point-to-point queues, or messaging publish/subscribe topics. Destinations are created on a bus and hosted on a messaging engine.
- ▶ Message store

Each messaging engine uses a set of tables in a data store, such as a JDBC database, to hold information such as messages, subscription information, and transaction states. Messaging engines can share a database, each using its own set of tables. The message store can be backed by any JDBC database supported by WebSphere Application Server.

2.10.1 Application support

The service integration bus supports the following application attachments:

- ▶ Web services
 - Requestors using the JAX-RPC API
 - Providers running in WebSphere Application Server as stateless session beans and servlets (JSR-109)
 - Requestors or providers attaching via SOAP/HTTP or SOAP/JMS
- ▶ Messaging applications
 - Inbound messaging using JFAP-TCP/IP (or wrapped in SSL for secure messaging)

JFAP is a proprietary format and protocol used for service integration bus messaging providers.
 - MQ application in an MQ network using MQ channel protocol
 - JMS applications in WebSphere Application Server V5 using WebSphere MQ client protocol
 - JMS applications in WebSphere Application Server V6

2.10.2 Service integration bus and messaging

With Express or Base, you typically have one stand-alone server with one messaging engine on one service integration bus. With Network Deployment, however, you have more flexibility by using service integration bus and messaging.

Figure 2-6 illustrates two application servers, each with a messaging engine on a service integration bus.

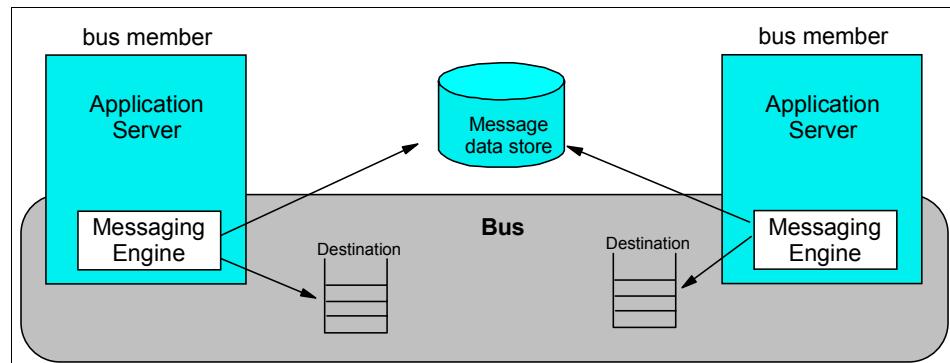


Figure 2-6 Service Integration Bus

The following are valid topologies:

- ▶ One bus and one messaging engine (application server or cluster)
- ▶ One bus with multiple messaging engines
- ▶ Multiple buses within a cell, which might or might not be connected to each other
- ▶ Buses connected between cells
- ▶ One application server that is a member of multiple buses and that has one messaging engine per bus.
- ▶ A connection between a bus and a WebSphere message queue manager

When using this type of topology, consider the following:

- WebSphere message queue can coexist on the same machine as the WebSphere default messaging provider. In V5, the embedded JMS server and WebSphere MQ cannot coexist on the same machine.
- A messaging engine cannot participate in a WebSphere MQ cluster.
- You can configure the messaging engine to look like another queue manager to WebSphere MQ.
- WebSphere applications can send messages directly to WebSphere MQ or through the service integration bus.
- You can have multiple connections to WebSphere MQ, but each connection must be to a different queue manager.
- WebSphere Application Server V5 JMS client can connect to V6 destinations. Also, a V6 JMS application can connect to an embedded

messaging provider in a V5 server if configured. However, you cannot connect a V5 embedded JMS server to a V6 bus.

Mediation

Mediation manipulates a message as it traverses the messaging bus (destination). For example, mediation:

- ▶ Transforms the message
- ▶ Reroutes the message
- ▶ Copies and routes the message to additional destinations
- ▶ Interacts with non-messaging resource managers (for example, databases)

You control mediation using a mediation handler list. The list is a collection of Java programs that perform the function of a mediation that are invoked in sequence.

Clustering

In a distributed server environment, you can use clustering for high availability and scalability. You can add a cluster as a bus member and achieve the following:

- ▶ High availability
 - One messaging engine is active in the cluster. In the event that the messaging engine or server fails, the messaging engine on a standby server is activated.
- ▶ Scalability
 - A single messaging destination can be partitioned across multiple active messaging engines in the cluster. Messaging order is not preserved.

Quality of service

You can define quality of service on a destination basis to determine how messages are (or are not) persisted. You can also specify quality of service within the application.

Message-driven beans

With EJB 2.1, message-driven beans (MDB) in the application server listen to queues. Topics are linked to the appropriate destinations on the service integration bus using JCA connectors (ActivationSpec objects). Support is also included for EJB 2.0 MDBs to be deployed against a listener port.

2.10.3 Web services and the service integration bus

Through the service integration bus Web services enablement, you can:

- ▶ Make an internal service that is already available at a service destination available as a Web service.
- ▶ Make an external Web service available at a service destination.
- ▶ Use the Web Services Gateway to map an existing service, either an internal service or an external Web service, to a new Web service that appears to be provided by the gateway.

2.11 Security

Table 2-7 shows the security features that the WebSphere Application Server configurations support.

Table 2-7 WebSphere Application Server security support

Security feature	Express and Base	Network Deployment
Java 2 security	Yes	Yes
J2EE security (role mapping)	Yes	Yes
JAAS	Yes	Yes
CSIV2	Yes	Yes
JACC	Yes	Yes
Security authentication	LTPA, SWAM	LTPA
User registry	Local OS, LDAP, custom registry	Local OS, LDAP, custom registry

Figure 2-7 on page 52 presents a general view of the logical layered security architecture model of WebSphere Application Server. The flexibility of that architecture model lies in pluggable modules that you can configure according to your requirements and existing IT resources.

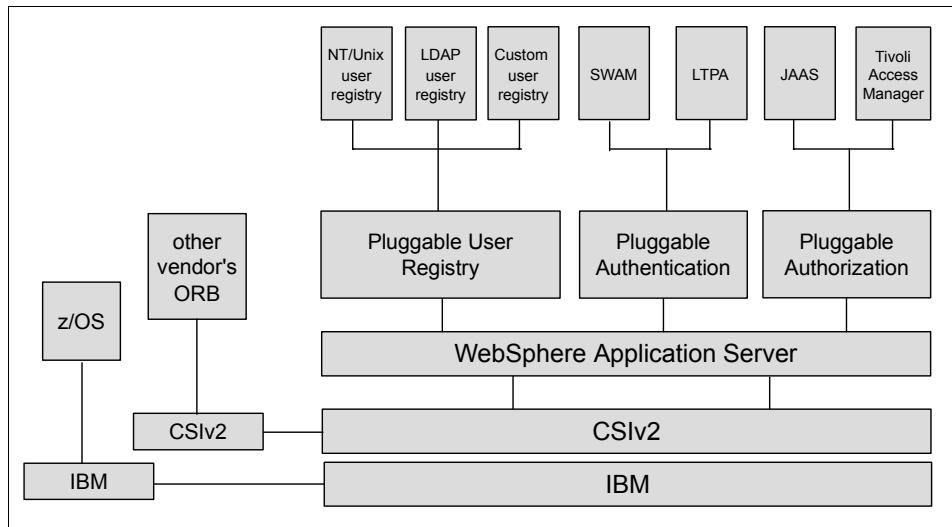


Figure 2-7 WebSphere Application Server security architecture

WebSphere Application Server security sits on top of the operating system security and security features of other components, including the Java language. This architecture provides the following layers of security:

- ▶ Operating system security protects sensitive WebSphere configuration files and authenticates users when the operating system user registry is used for authentication.
- ▶ Standard Java security is provided through the JVM that WebSphere and the Java security classes use.
- ▶ The Java 2 Security API provides a means to enforce access control, based on the location of the code and who signed it. Java 2 security guards access to system resources such as file I/O, sockets, and properties. WebSphere global security settings allow you to enable or disable Java 2 security and provide a default set of policies. You can activate or deactivate Java 2 security independently from WebSphere global security.

The current principal of the thread of execution is not considered in the Java 2 security authorization. There are instances where it is useful for the authorization to be based on the principal, rather than the code base and the signer.

- ▶ The Java Authentication and Authorization Services (JAAS) is a standard Java API that allows the Java 2 security authorization to be extended to the code base on the principal as well as the code base and signers. The JAAS programming model allows the developer to design application authentication

in a pluggable fashion, which makes the application independent from the underlying authentication technology. JAAS does not require Java 2 security to be enabled.

- ▶ The Common Secure Interoperability protocol adds additional security features that enable interoperable authentication, delegation and privileges in a CORBA environment. It supports interoperability with the EJB 2.1 specification and can be used with SSL.
- ▶ J2EE security uses the security collaborator to enforce security policies based on J2EE and to support J2EE security APIs. WebSphere applications use security APIs to access the security mechanisms and implement security policies. J2EE security guards access to Web resources such as servlets/JSPs and EJB methods based on roles that the application developer defines. Users and groups are assigned to these roles during application deployment.
- ▶ Java Contract for Containers (JACC) support allows the use of third-party authorization providers for access decisions. The default JACC provider for WebSphere Application Server is the Tivoli Access Manager that is bundled with Network Deployment. The Tivoli Access Manager client functions are integrated in WebSphere Application Server.
- ▶ IBM Java Secure Socket Extension is the SSL implementation that WebSphere Application Server uses. It is a set of Java packages that enable secure Internet communications. It implements a Java version of SSL and Transport Layer Security protocols and includes functionality for data encryption, server authentication, message integrity, and client authentication.

WebSphere Application Server security relies on and enhances all the above mentioned layers. It implements security policies in a unified manner for both Web and EJB resources. WebSphere global security options are defined at the cell level. However, individual servers can override a subset of the security configuration. When using mixed z/OS and distributed nodes, the security domain features are merged.

2.11.1 User registry

The pluggable user registry allows you to configure different databases to store user IDs and passwords that are used for authentication and authorization. Only one single registry can be active at a time. There are three options for user registries:

- ▶ Local operating system user registry

When configured, WebSphere uses the operating system's users and groups for authentication.

- ▶ LDAP user registry

An LDAP user registry is often the best solution for large scale Web implementations. Most LDAP servers on the market are well equipped with security mechanisms that you can use to securely communicate with WebSphere Application Server. The flexibility of search parameters that an administrator can set to adapt WebSphere to different LDAP schemas is considerable.

- ▶ Custom user registry

A custom user registry leaves an open door for any custom implementation of a user registry database. You should use the UserRegistry Java interface that the WebSphere API provides to write a custom registry. You can use this interface to access virtually any relational database, flat files, and so on.

2.11.2 Authentication

Authentication is the process of establishing whether a client is valid in a particular context. A client can be either a user, a machine, or an application. The pluggable authentication module allows you to choose whether WebSphere authenticates the user or accepts the credentials from external authentication mechanisms.

An authentication mechanism in WebSphere typically collaborates closely with a user registry when performing authentication. The authentication mechanism is responsible for creating a credential, which is a WebSphere internal representation of a successfully authenticated client user. Not all credentials are created equal. The abilities of the credential are determined by the configured authentication mechanism.

Although WebSphere provides several authentication mechanisms, only a single active authentication mechanism can be configured at once. The active authentication mechanism is selected when configuring WebSphere global security.

WebSphere provides two authentication mechanisms that differ primarily in the distributed security features each supports:

- ▶ Simple WebSphere Authentication Mechanism (SWAM) is intended for simple, non-distributed, single application server type runtime environments. The single application server restriction is because this mechanism does not support forwardable credentials. So, if a servlet or EJB in application server process 1 invokes a remote method on an EJB living in another application server process 2, the identity of the caller identity in process 1 is not transmitted to server process 2. What is transmitted is an unauthenticated

credential, which depending on the security permissions configured on the EJB methods, might cause authorization failures.

Because the Simple WebSphere Authentication Mechanism is intended for a single application server process, single sign-on is not supported.

This type of authentication is suitable for simple environments, software development environments, or other environments that do not require a distributed security solution.

SWAM relies on the session ID and is not as secure as LTPA. For this reason, we strongly recommend using SSL with this type of authentication.

- ▶ Light Weight Third Party Authentication (LTPA)

LTPA is intended for distributed, multiple application servers and machine environments. It supports forwardable credentials and single sign-on.

Lightweight Third Party Authentication is able to support security in a distributed environment through the use of cryptography. This allows it to encrypt, digitally sign, and securely transmit authentication related data and later decrypt and verify the signature.

This type of authentication requires that the configured user registry be a central, shared repository such as LDAP or a Windows domain type registry.

2.11.3 Authorization

WebSphere Application Server standard authorization features are as follows:

- ▶ Java 2 security architecture, which uses a security policy to specify who is allowed to execute code in the application. Code characteristics, such as a code signature, signer ID, or source server, determine whether the code is granted access to be executed.
- ▶ JAAS, which extends the Java 2 approach with role-based access control. Permission to execute a code is granted based on the code characteristics and also on the user running it. JAAS programming models allow the developer to design application authentication in a pluggable fashion, which makes the application independent from the underlying authentication technology.

For each authenticated user, a Subject class is created and a set of Principals is included in the subject to identify that user. Security policies are granted based on possessed principals.

WebSphere Application Server provides an internal authorization mechanism that is used by default. As an alternative, you can define external JACC providers to handle authorization decisions. During application installation, security policy information is stored in the JACC provider server using standard interfaces

defined by JACC. Subsequent authorization decisions are made using this policy information. An exception is that the WebSphere Application Server default authorization engine makes all administrative security authorization decisions.

2.11.4 Security components

Figure 2-8 shows an overview of the security components that come into play in WebSphere Application Security.

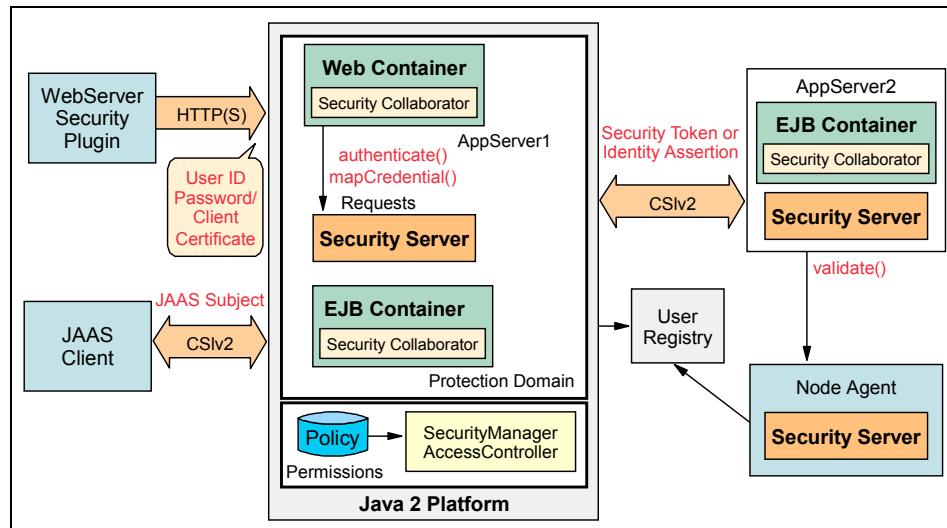


Figure 2-8 WebSphere Application Security components

Security server

The security server is a component of WebSphere Application Server that runs in each application server process. If multiple application server instances are executed on a single node, then multiple security servers exist on that node.

The security server component is responsible for managing authentication and for collaborating with the authorization engine and the user registry.

Security collaborators

Security collaborators are application server processes that enforce security constraints specified by the deployment descriptors. These processes communicate with the security server every time authentication and authorization actions are required. The following security collaborators are identified:

- ▶ The Web security collaborator resides in the Web container and provides the following services to the application:
 - Checks authentication
 - Performs authorization according to the constraint specified in the deployment descriptor
 - Logs security tracing information
- ▶ The EJB security collaborator resides in the EJB container. The EJB security collaborator uses Common Secure Interoperability Version 2 (CSlv2) and Secure Authentication Service (SAS) to authenticate Java client requests to enterprise beans. The EJB security collaborator works with the security server to perform the following functions:
 - Checks authorizations according to the specified security constraint
 - Supports communication with local user registry
 - Logs security tracing information
 - Communicates with external ORBs using CSlv2 when a request for a remote bean is issued

2.11.5 Security flows

The following sections outline the general security flow.

Web browser communication

When a Web browser sends a request to a WebSphere application, the following security interactions occur:

1. The Web user requests a Web resource that is protected by WebSphere Application Server.
2. The Web server receives the request and recognizes that the requested resource is on the application server.
3. Using the Web server plug-in, the Web server redirects the request to the Web security collaborator, which performs user authentication.
4. After successful authentication, the Web request reaches the Web container. The Web security collaborator passes the user's credentials and the security information contained in the deployment descriptor to the security server for authorization.

- Upon subsequent requests, authorization checks are performed either by the Web collaborator or the EJB collaborator, depending on what the user is requesting. User credentials are extracted from the established security context.

Administrative tasks

Administrative tasks are issued using either the Web-based administrative console or the wsadmin scripting tool. The following tasks are executed:

- The administration client generates a request that reaches the server side ORB and JMX MBeans. The JMX MBeans represent managed resources.
- The JMX MBeans contact the security server for authentication purposes. JMX beans have dedicated roles assigned and do not use user registry for authentication and authorization.

Java client communication

When a Java client interacts with a WebSphere application, the following occurs:

- A Java client generates a request that reaches the server side ORB.
- The CSIV2 or IBM SAS interceptor performs authentication on the server side on behalf of the ORB, and sets the security context.
- The server side ORB passes the request to the EJB container.
- After submitting a request to the access-protected EJB method, the EJB container passes the request to the EJB collaborator.
- The EJB collaborator reads the deployment descriptor from the EAR file and reads the user credentials from the security context.
- Credentials and security information are passed to the security server, which validates user access rights and passes this information back to the collaborator.
- After receiving a response from the security server, the EJB collaborator authorizes or denies access to the user to the requested resource.

2.12 Resource providers

Resource providers define resources that running J2EE applications need. Table 2-8 on page 59 shows the resource provider support of the WebSphere Application Server configuration.

Table 2-8 WebSphere Application Server resource provider support

Resource	Express and Base	Network Deployment
JDBC provider	Yes	Yes
Mail providers (JavaMail)	Yes	Yes
JMS providers	Yes	Yes
Resource environment providers	Yes	Yes
URL providers	Yes	Yes
Resource adapters	Yes	Yes

2.12.1 JDBC resources

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 technique makes an application more portable, because the application does not need to hard code a driver name, which often includes the name of a particular vendor. The technique also makes maintaining the code easier. If, for example, you move the data source to a different server, all you need to do is update the relevant property in the data source. You do not need to touch the code using that data source.

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 is usually a pooled connection. That is, 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, you are providing information about the set of classes that are used to implement the data source and the database driver. That is, the JDBC provider holds the environment settings for the `DataSource` object.

Data sources

In WebSphere Application Server, connection pooling is provided by two parts, a JCA Connection Manager and a relational resource adapter, as shown in Figure 2-9.

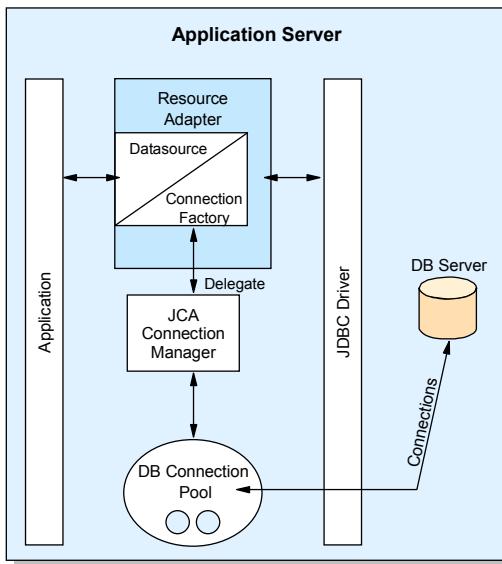


Figure 2-9 Resource adapter in J2EE connector architecture

The JCA Connection Manager provides the connection pooling, local transaction, and security supports. The relational resource adapter provides the JDBC wrappers and JCA CCI implementation that allow applications using bean-managed persistence, JDBC calls, and container-managed persistence beans to access the database JDBC Driver.

Important: WebSphere Version 4.0 provided its own JDBC connection manager to handle connection pooling and JDBC access. This support is included with WebSphere Application Server V5 and 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 WebSphere Version 4.0.

2.12.2 Mail providers

The JavaMail APIs provide a platform and protocol-independent framework for building Java-based mail client applications. The JavaMail APIs require service

providers, known in WebSphere as protocol providers, to interact with mail servers that run the appropriate protocols.

A mail provider encapsulates a collection of protocol providers. WebSphere Application Server has a Built-in Mail Provider that encompasses the following protocol providers:

- ▶ Simple Mail Transfer Protocol (SMTP) 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) is the standard protocol for receiving mail.
- ▶ Internet Message Access Protocol (IMAP) is an alternative protocol to POP3 for receiving mail.

These protocol providers are installed as the default and should be sufficient for most applications. To use other protocols, you must install the appropriate service provider for those protocols.

In addition to service providers, JavaMail requires the JavaBeans Activation Framework (JAF) as the underlying framework to deal with complex data types that are not plain text, such as Multipurpose Internet Mail Extensions (MIME), 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`, containing the JavaMail APIs as well as the SMTP, IMAP, and POP3 service providers
- ▶ `activation.jar`, containing the JAF

2.12.3 JCA resource adapters

The JCA defines a standard architecture for connecting the J2EE platform to heterogeneous EIS. Imagine an ERP, mainframe transaction processing, database systems, and legacy applications not written in the Java programming language.

The JCA resource adapter is a system-level software driver supplied by EIS vendors or other third-party vendors. It provides the connectivity between J2EE components (an application server or an application client) and an EIS.

To use a resource adapter, install the resource adapter code and create connection factories that use the adapter.

One resource adapter, the WebSphere Relational Resource Adapter, is predefined for handling data access to relational databases. This resource adapter provides data access through JDBC calls to access databases dynamically. It provides connection pooling, local transaction, and security support. The WebSphere persistence manager uses this adapter to access data for container-managed persistence beans.

2.12.4 URL providers

URL providers implement the functionality for a particular URL protocol, such as HTTP, by extending the `java.net.URLStreamHandler` and `java.netURLConnection` 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 uses the URL support that the IBM JDK provides. 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 providers that implement other protocols that the JDK does not support.

2.12.5 JMS providers

The JMS functionality that WebSphere provides includes support for three types of JMS providers:

- ▶ Default messaging provider (service integration bus)
- ▶ WebSphere MQ provider
- ▶ Generic JMS providers
- ▶ V5 default messaging provider (for migration)

There can be more than one JMS provider per node. That is, you can configure a node to use any combination or all of the providers, including the default messaging provider, WebSphere MQ JMS provider, and a generic JMS provider concurrently. In addition, WebSphere MQ and the default messaging provider can coexist on the same machine.

The support provided by WebSphere administration tools for configuration of JMS providers differs depending upon the provider. Table 2-9 on page 63 provides a summary of the support.

Table 2-9 WebSphere administration support for JMS provider configuration

Configurable objects	Default messaging provider	WebSphere MQ JMS provider	Generic JMS provider	V5 default messaging, WebSphere JMS provider
Messaging system objects (queues/topics)	Yes	No	No	Yes
JMS administered objects (JMS connection factory and JMS destination)	Yes	Yes	No	Yes

Default messaging provider

The default messaging provider for WebSphere Application Server uses the service integration bus for transport. The default message provider provides point-to-point as well as publish and subscribe functions. Within this provider, you define JMS connection factories and JMS destinations that correspond to service integration bus destinations.

WebSphere MQ messaging provider

WebSphere Application Server supports the use of full WebSphere MQ as the JMS provider. The product is tightly integrated with the WebSphere installation. WebSphere provides the JMS client classes and administration interface, while WebSphere MQ provides the queue-based messaging system.

Generic messaging providers

WebSphere Application Server supports the use of generic messaging providers, as long as they implement the ASF component of the JMS 1.0.2 specification. JMS resources for generic messaging providers are not configurable using WebSphere administration.

V5 default messaging provider

For backwards compatibility with earlier releases, WebSphere Application Server V6 also includes support for the V5 default messaging provider, which enables you to configure resources for use with V5 embedded messaging. You can also use the V5 default messaging provider with a service integration bus.

2.12.6 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 looked up. WebSphere Application Server provides a number of local environment entries by default.

The J2EE specification also provides a mechanism for defining custom (non-default) environment entries using <resource-env-ref> entries that are defined in an application's standard deployment descriptors. The J2EE 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 to be accessible via JNDI in the application server's local name space (java:comp/env).
- ▶ Specifying the administrative object's JNDI lookup name and the expected returned object type in <resource-env-ref>

WebSphere Application Server supports the <resource-env-ref> mechanism by providing administration objects for the following:

- ▶ *Resource environment provider* defines an administrative object that groups together the referenceable, resource environment entry administrative objects and any required custom properties.
- ▶ *Referenceable* defines the class name of the factory class that returns object instances implementing a Java interface.
- ▶ *Resource environment entry* defines the binding target (JNDI name), factory class, and return object type (via the link to the referenceable) of the resource environment entry.

2.13 Workload management

Clustering application servers that host Web containers automatically enables plug-in workload management for the application servers and the servlets they host. Routing of servlet requests occurs between the Web server plug-in and the clustered application servers using HTTP or HTTPS, as shown in Figure 2-10 on page 65.

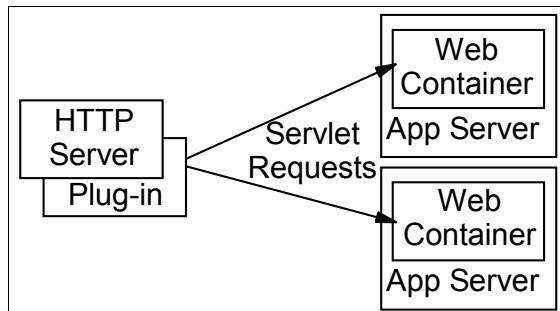


Figure 2-10 Plug-in (Web container) workload management

This routing is based on weights associated with the cluster members. If all cluster members have identical weights, the plug-in sends equal requests to all members of the cluster, assuming no strong affinity configurations. If the weights are scaled in the range from zero to twenty, the plug-in routes requests to those cluster members with the higher weight value more often.

A rule of thumb formula for determining routing preference is:

$$\% \text{ routed to Server1} = \text{weight1} / (\text{weight1} + \text{weight2} + \dots + \text{weightn})$$

In this statement, n is the number of cluster members in the cluster.

The Web server plug-in temporarily routes around unavailable cluster members.

Workload management for EJB containers can be performed by configuring the Web container and EJB containers on separate application servers. Multiple application servers with the EJB containers can be clustered, enabling the distribution of EJB requests between the EJB containers as shown in Figure 2-11.

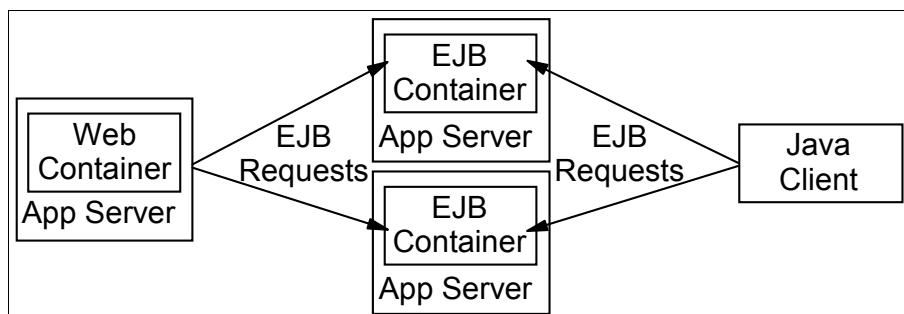


Figure 2-11 EJB workload management

In this configuration, EJB client requests are routed to available EJB containers in a round robin fashion based on assigned server weights. The EJB clients can be servlets operating within a Web container, stand-alone Java programs using RMI/IOP, or other EJBs.

The server weighted round robin routing policy ensures a distribution based on the set of server weights that have been assigned to the members of a cluster. For example, if all servers in the cluster have the same weight, the expected distribution for the cluster is that all servers receive the same number of requests. If the weights for the servers are not equal, the distribution mechanism sends more requests to the higher weight value servers than the lower weight value servers. The policy ensures the desired distribution, based on the weights assigned to the cluster members. In WebSphere Application Server V6, the balancing mechanism for weighted round robin is enhanced to ensure more balanced routing distribution among servers.

You can also choose to have requests sent to the node on which the client resides as the preferred routing. In this case, only cluster members on that node are chosen using the round robin weight method. Cluster members on remote nodes are chosen only if a local server is not available.

2.14 High availability

With Network Deployment V6, the high availability features are significantly improved. The following is a quick overview of the failover capabilities:

- ▶ HTTP server failover

The use of multiple HTTP servers, along with a load balancing product such as provided with the Edge components can be used to provide HTTP Server failover.

- ▶ Web container failover

The HTTP server plug-in in the Web server is aware of the configuration of all Web containers and can route around a failed Web container in a cluster. Sessions can be persisted to a database or in-memory using data replication services.

- ▶ EJB container failover

Client code and the ORB plug-in can route to the next EJB container in the cluster.

- ▶ Deployment manager and node agent

The need for failover in these two components has been reduced. Thus, no built-in failover capability is provided. The loss of the deployment manager

only affects configuration. We recommend that you use a process nanny to restart the Node Agent if it fails.

- ▶ Critical services failover

Hot standby and peer failover for critical services such as workload management routing, PMI aggregation, JMS messaging, transaction manager, and so on, is provided through the use of high availability domains.

A high availability domain defines a set of WebSphere processes, a core group, that provides high availability function to each other. Processes in the core group can be the deployment manager, node agents, application servers or cluster members.

One or more members of the core group can act as a high availability coordinator, managing the high availability activities within the core group processes. If a high availability coordinator server fails, another server in the core group takes over. High availability policies define how the failover occurs.

Workload management information is shared between core members and failover of critical services is done among them in a peer-to-peer fashion. Little configuration is necessary, and in many cases, this function works with the defaults that are created automatically as you create the processes.

- ▶ Transaction log hot standby

With V6, transaction logs can be maintained on Network Attached Storage. When a cluster member fails, another cluster member recovers the transaction log, thus enabling the failover 2PC transactions.

- ▶ JMS messaging failover

The messaging engine keeps messages in a remote database. When a server in a cluster fails, WebSphere selects an online server to run the Messaging Engine and the workload manager routes JMS connections to that server.

2.15 Administration

WebSphere Application Server's administration model is based on the Java Management Extensions (JMX) framework. JMX allows you to wrap hardware and software resources in Java and expose them in a distributed environment. JMX also provides a mapping framework for integrating existing management protocols, such as SNMP, into JMX's own management structures.

Each application server has an administration service that provides the necessary functions to manipulate configuration data for the server and its

components. The configuration is stored in a repository. The repository is a set of XML files that are stored in the server's file system.

2.15.1 Administration tools

Table 2-10 shows the administration tools that WebSphere Application Server supports by configuration.

Table 2-10 WebSphere Application Server administration tool support

Tool	Express and Base	Network Deployment
Administrative console	Yes	Yes
Commands	Yes	Yes
Scripting client, wsadmin	Yes	Yes

Administrative console

The administrative console is a Web-based interface that provides configuration and operation capability. The administrator connects to the application using a Web browser client. Users assigned to different administration roles can manage the application server and certain components and services using this interface.

The administrative console is a system application, crucial to the operation of WebSphere and, as such, is not exposed as an enterprise application on the console. In stand-alone application servers, the administrative console runs in the application server. In the Network Deployment distributed server environment, the administrative console application runs on the deployment manager. When a node is added to a cell, the administrative console application is deleted from the node and the configuration files are integrated into the master cell repository that the deployment manager maintains.

Commands

WebSphere Application Server provides a set of commands in the `<server_install>/bin` directory that allows you to perform a subset of administrative functions. For example, use the `startServer` command to start an application server.

Scripting client

The `wsadmin` scripting client provides extra flexibility over the Web-based administration application, allowing administration to use the command-line interface. Using the scripting client not only makes administration quicker, but it automates the administration of multiple application servers and nodes using scripts.

The scripting client uses the Bean Scripting Framework, which allows you to use a variety of scripting languages for configuration and control. WebSphere Application Server V6 supports two languages: jacl and jython (or jpython).

The wsadmin scripting interface is included in all WebSphere Application Server configurations but is targeted toward advanced users. The use of wsadmin requires in-depth familiarity with application server architecture and a scripting language.

2.15.2 Configuration repository

The configuration repository holds copies of the individual component configuration documents stored in XML files. The application server's administrative service takes care of the configuration and makes sure it is consistent during the runtime.

You can archive the configuration of unfederated nodes for export and import, making them portable among different WebSphere Application Server instances.

2.15.3 Centralized administration

The Network Deployment package allows multiple servers and nodes to be administered from a central location. This centralized administration uses a central deployment manager that handles the administration process and distributes the updated configuration to the node agent for each node. The node agent, in turn, maintains the configuration for the servers in the node. Table 2-11 on page 70 shows the distributed administration that WebSphere Application Server supports by configuration.

All operating system processes that are components of the WebSphere product are called managed servers or managed processes. JMX support is embedded in all managed processes. These processes are available to receive administration commands and to output administration information about the state of the managed resources within the processes.

WebSphere provides the following managed servers and processes:

- ▶ *Deployment manager* provides a single point to access configuration information and control for a cell. The deployment manager aggregates and communicates with the node agent processes on each node in the system.
- ▶ *Node agent* aggregates and controls the WebSphere managed processes on its node. There is one node agent per node.
- ▶ *Application server* is a managed server that hosts J2EE applications.

Table 2-11 shows the managed processes supported by each packaging option.

Table 2-11 WebSphere Application Server distributed administration support

Process	Express and Base	Network Deployment
Deployment manager	No	Yes
Node agent	No	Yes
Application servers	Stand-alone	Stand-alone or distributed server clustering

Deployment manager

The deployment manager process provides a single, central point of administrative control for all elements in the cell. It hosts the Web-based administrative console application. Administrative tools that need to access any managed resource in a cell usually connect to the deployment manager as the central point of control. Using the deployment manager, horizontal scaling, vertical scaling, and distributed applications are all easy to administer and manage. Application servers are managed by nodes, and one or more nodes is managed by a cell.

In a distributed server environment, the deployment manager maintains a master configuration repository that contains all of the cell's configuration data. The configuration repository at each node is a synchronized subset of the master repository. The node repositories are read-only for application server access. Only the deployment manager can initiate their update and push out configuration changes from the cell master configuration repository. It manages through communication with the node agent process resident on each node of the cell.

Node agent

The node agent is an administrative process and is not involved in application serving functions. It hosts important administrative functions such as:

- ▶ File transfer services
- ▶ Configuration synchronization
- ▶ Performance monitoring

The node agent aggregates and controls all the managed processes on its node by communicating with:

- ▶ The deployment manager to coordinate configuration synchronization and to perform management operations on behalf of the deployment manager.
- ▶ Application servers and managed Web servers to manage (start or stop) each server and to update its configuration and application binaries as required.

Only one node agent is defined and run on each node. In a stand-alone server environment, there is no node agent.

2.16 The flow of an application

Figure 2-12 shows the typical application flow for Web browser clients using either JDBC from a servlet or EJB to access application databases.

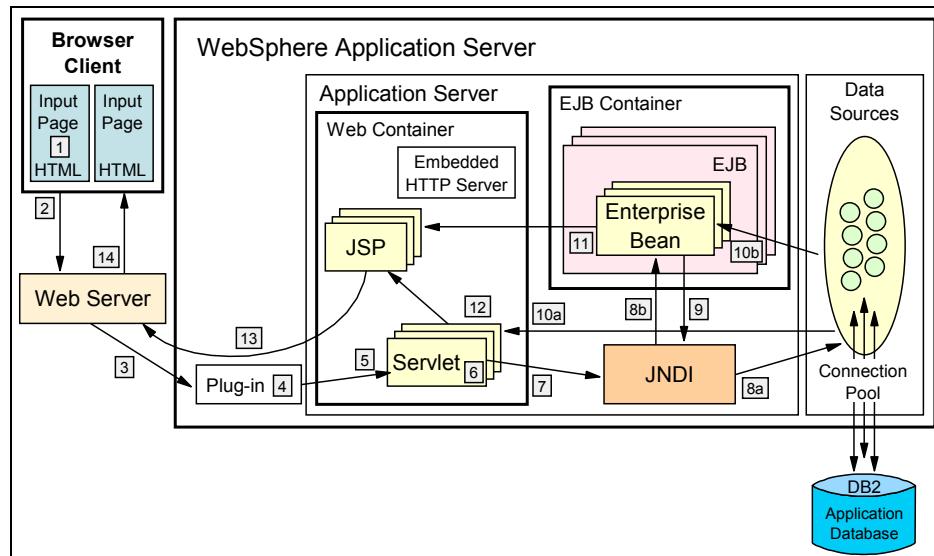


Figure 2-12 Application flow

The typical application flow is as follows:

1. A Web client requests a URL in the browser input page.
2. The request is routed to the Web server over the Internet.
3. The Web server immediately passes the request to the Web server plug-in. All requests go to the WebSphere plug-in first.
4. The Web server plug-in examines the URL, verifies the list of host name aliases from which it will accept traffic based on the virtual host information, and chooses a server to handle the request.
5. A *stream* is created. A *stream* is a connection to the Web container. It is possible to maintain a stream over a number of requests. The Web container receives the request and, based on the URL, dispatches it to the proper servlet.

6. If the servlet class is not loaded, the dynamic class loader loads the servlet (servlet init(), then doGet() or doPost()).
7. JNDI is used for lookup of either datasources or EJBs required by the servlet.
8. Depending upon whether a datasource is specified or an EJB is requested, the JNDI directs the servlet:
 - To the corresponding database and gets a connection from its connection pool in the case of a data source.
 - To the corresponding EJB container, which then instantiates the EJB when an EJB is requested.
9. If the EJB requested involves an SQL transaction, it goes back to the JNDI to look up the datasource.
10. The SQL statement is executed and the data retrieved is sent back either to the servlet or to the EJB.
11. Data beans are created and handed off to JSPs in the case of EJBs.
12. The servlet sends data to JSPs.
13. The JSP generates the HTML that is sent back through the WebSphere plug-in to the Web server.
14. The Web server sends the output HTML page to the browser.

2.17 Developing and deploying applications

Figure 2-13 on page 73 shows a high-level view of the stages of application development and deployment.

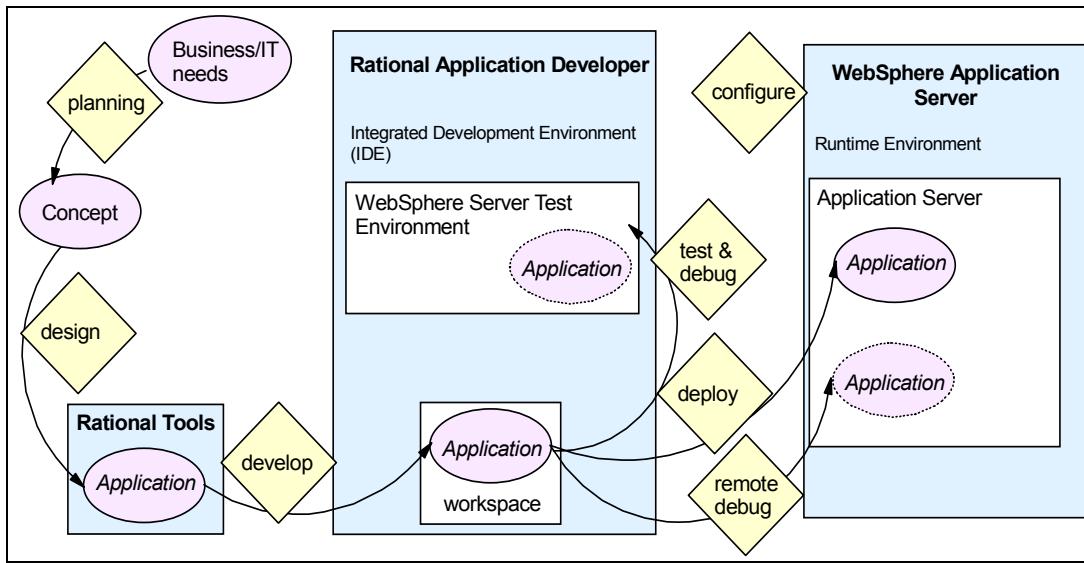


Figure 2-13 Develop and deploy

2.17.1 Application design

Design tools like Rational Rose® or Rational XDE™ can be used to model the application using the Unified Modeling Language. The output of the modeling generally consists of use-case scenarios, class diagrams, and starter code generated based on the model.

2.17.2 Application development

Application development is done using Rational Application Developer or a comparable IDE to create the enterprise application. You can start by importing pre-generated code from modeling tools, a sample application, an existing production application, or you can start from scratch.

Rational Application Developer provides many tools and aids to get you started quickly. It also supports team development using CVS or Rational ClearCase®, which allows multiple developers to share a single master source copy of the code.

During the development phase, you can do component testing using the built-in WebSphere Application Server test environment. Rational Application Developer provides server tools capable of creating and managing servers both in the test

environment and on remote server installations. The application is automatically packaged into an EAR file for deployment when you run the application on a server using Rational Application Developer.

2.17.3 Application packaging

J2EE applications are packaged into EAR files to be deployed to one or more application servers. A J2EE application contains any or all of the modules as shown in Table 2-12.

Table 2-12 J2EE 1.3 application modules

Module	Filename	Contents
Web module	<module>.war	Servlets, JSP files, and related code artifacts
EJB module	<module>.jar	Enterprise beans and related code artifacts
Application client module	<module>.jar	Application client code
Resource adapter module	<module>.rar	Library implementation code that your application uses to connect to enterprise information systems (EIS)

This packaging is done automatically in Rational Application Developer when you export an application for deployment. If you are using another IDE, WebSphere Application Server (with the exception of Express) provides the Application Server Toolkit for packaging applications.

WebSphere Enhanced EAR files

The WebSphere Enhanced EAR, introduced in WebSphere Application Server V6, is a regular J2EE EAR file with additional configuration information for resources usually 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.

When you deploy an enhanced EAR to a WebSphere Application Server V6 server, WebSphere can configure the resources specified in the enhanced EAR automatically. This automatic configuration reduces the number of steps that are required to set up the WebSphere environment to host the application.

2.17.4 Application deployment

Applications are installed on application servers using the administrative console or the wsadmin scripting interface. You can deploy an application to a single

server or a cluster. In a cluster, the application is installed on each application server in the cluster.

Installing an application involves the following tasks:

- ▶ Binding resource references, created during packaging, to real resources
 - For example, a data source would need to be bound to a real database.
- ▶ Defining JNDI names for EJB home objects
- ▶ Specifying data source entries for entity beans
- ▶ Binding EJB references to the real EJB JNDI names
- ▶ Mapping Web modules to virtual hosts
- ▶ Specifying listener ports for message-driven beans
- ▶ Mapping application modules to application servers
- ▶ Mapping security roles to users or groups

The use of an enhanced EAR file simplifies this installation process.

After a new application is deployed, the Web server plug-in configuration file needs to be regenerated and copied to the Web server.

Application update

In previous releases, deploying an update to an application required a complete EAR file to be deployed and the application to be restarted. WebSphere Application Server V6 allows partial updates to applications and makes it possible to restart only parts of an application.

Updates to an application can consist of individual application files, application modules, zipped files that contain application artifacts, or the complete application. All module types can be started (though only Web modules can be stopped).

In V6, you have a rollout start option for installing applications on a cluster that will stop, update, and start each cluster member in turn, ensuring availability.

2.17.5 WebSphere Rapid Deployment

WebSphere Rapid Deployment is designed to simplify the development and deployment of WebSphere applications. It is a collection of Eclipse plug-ins that can be integrated within development tools or run in a headless mode, without headers, from a user file system. WebSphere Rapid Deployment is currently integrated in Rational Web Developer, Rational Application Developer, and the

Application Server Toolkit. Initially, there are features that are only supported in headless mode.

During development, annotation-based programming is used. The developer adds metadata tags into the application source code that are used to generate artifacts needed by the code, thus reducing the number of artifacts the developer needs to create.

These applications are packaged into an enhanced EAR file that contains the J2EE EAR file along with deployment information, application resources, and properties (environment variables, JAAS authentication entries, shared libraries, classloader settings, and JDBC resources). During installation, this information is used to create the necessary resources. Moving an application from one server to another also moves the resources.

WebSphere Rapid Deployment automates installation of applications and modules onto a running application server by monitoring the workspace for changes and then driving the deployment process.

2.18 Technology support summary

Table 2-13 highlights the support that each WebSphere Application Server packaging option provides.

Table 2-13 WebSphere Application Server features and technology support

Feature	Base and Express V6	Network Deployment V6
Client and server support for the Software Development Kit for Java Technology Edition 1.4 (SDK 1.4.2)	Yes	Yes
J2EE 1.2, 1.3 programming support	Yes	Yes

Feature	Base and Express V6	Network Deployment V6
J2EE 14. programming support ¹ <ul style="list-style-type: none"> ▶ EJB 2.1 ▶ Servlet 2.4 ▶ JSP 2.0 ▶ JMS 1.1 ▶ JTA 1.0 ▶ JavaMail 1.3 ▶ JAF 1.0 ▶ JAXP 1.2 ▶ Connector 1.5 ▶ Web Services 1.1 ▶ JAX-RPC 1.1 ▶ SAAJ 1.2 ▶ JAXR 1.0 ▶ J2EE Management 1.0 ▶ JMX 1.2 ▶ JACC 1.0 ▶ JDBC 3.0 	Yes	Yes
WebSphere Rapid Deployment	Yes	Yes
Service Data Object (SDO)	Yes	Yes
Messaging support <ul style="list-style-type: none"> ▶ Integrated JMS 1.1 messaging provider ▶ Support for WebSphere MQ and generic messaging providers ▶ Message-driven beans 	Yes	Yes
Web services runtime support	Yes	Yes
Security support <ul style="list-style-type: none"> ▶ Java 2 ▶ J2EE ▶ JACC 1.0 ▶ JAAS 1.0 ▶ CSIV2 and SAS authentication protocols ▶ LDAP or local operating system user registry ▶ LTPA authentication mechanism ▶ Kerberos, Technology Preview 	Yes	Yes
▶ Simple WebSphere Authentication Mechanism (SWAM)	Yes	stand-alone server environment only

Feature	Base and Express V6	Network Deployment V6
Multi-node management and Edge components		
Workload management and failover	No	Yes
Deployment manager	No	Yes
Central administration of multiple nodes	No	Yes
Load Balancer	No	Yes
Caching Proxy	No	Yes
Dynamic caching	Yes	Yes
Performance and analysis tools		
Performance Monitoring Instrumentation (PMI)	Yes	Yes
Log Analyzer	Yes	Yes
Tivoli Performance Viewer (integrated in the administration console)	Yes	Yes
Administration and tools		
Administration and tools <ul style="list-style-type: none">▶ Web-based administration console▶ Integrated IBM HTTP Server and Application Server Administration Console▶ Administrative scripting▶ Java Management Extension (JMX) 1.2▶ J2EE Management (JSR-077)▶ J2EE Deployment (JSR-088)▶ Application Server Toolkit	Yes	Yes

Feature	Base and Express V6	Network Deployment V6
Web services <ul style="list-style-type: none"> ▶ JAX-RPC v1.0 for J2EE 1.3, v1.1 for J2EE 1.4 ▶ JSR 109 (Web services for J2EE) ▶ WS-I Basic Profile 1.1.2 support ▶ WS-I Simple SOAP Binding Profile 1.0.3 ▶ WS-I Attachments Profile 1.0 ▶ SAAJ 1.2 ▶ UDDI V2 and V3 ▶ JAXR ▶ WS-TX (transactions) ▶ SOAP 1.1 ▶ WSDL 1.1 for Web services ▶ WSIL 1.0 for Web services ▶ OASIS Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) ▶ OASIS Web Services Security: UsernameToken Profile 1.0 ▶ OASIS Web Services Security X.509 Certificate Token Profile 	Yes	Yes
Web Services Gateway	No	Yes
Private UDDI v3 Registry	Yes	Yes
Programming model extensions ²		
<ul style="list-style-type: none"> ▶ Activity sessions ▶ Application Profiling ▶ Asynchronous Beans (now called WorkManager) ▶ Dynamic caching ▶ Dynamic query ▶ Internationalization Service ▶ Object Pools ▶ Scheduler Service (now called Timer Service) ▶ Startup Beans ▶ WorkArea Service ▶ Extended JTA Support ▶ Last Participant Support 	Yes	Yes
▶ Back-up Cluster Support	No	Yes
1. You can see the APIs required for J2EE 1.4 in the Application Programming Interface section of the J2EE 1.4 specifications at: http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf 2. Business process choreography and business rule beans remain in WebSphere Business Integration Server Foundation.		



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

3.1 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:

- ▶ *Standalone server environment* refers to a single standalone 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 standalone 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.

3.1.1 System management tools

The IBM WebSphere Application Server administration tools include the following:

- ▶ WebSphere administrative console

The administrative console provides an HTML interface that allows you to configure and manage a WebSphere Application Server environment. The location and configuration of the console differs depending upon whether you have a standalone server or a distributed server environment.

- ▶ Command-line operational tools

A set of command line tools are available in the bin directory to perform specific actions on a targeted process. For example, you can use command line tools to start or stop a node, add (or remove) a node to a cell, to start or stop servers, or to view the status of a server.

- ▶ WebSphere scripting using wsadmin

The wsadmin scripting interface can be used to configure and manage WebSphere processes. The administrator can invoke wsadmin interactively, or create scripts to be run. Creating scripts can save time when you are

repeatedly performing the same set of actions on one or more WebSphere Application Server installations.

- ▶ Java-based JMX APIs that can be accessed directly by custom Java applications.

This book focuses primarily on using the administrative console, but will also provide information about using commands and wsadmin scripts.

3.1.2 System management in a standalone server environment

Each managed process has an administrative service that interacts with administration clients. In a standalone 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.

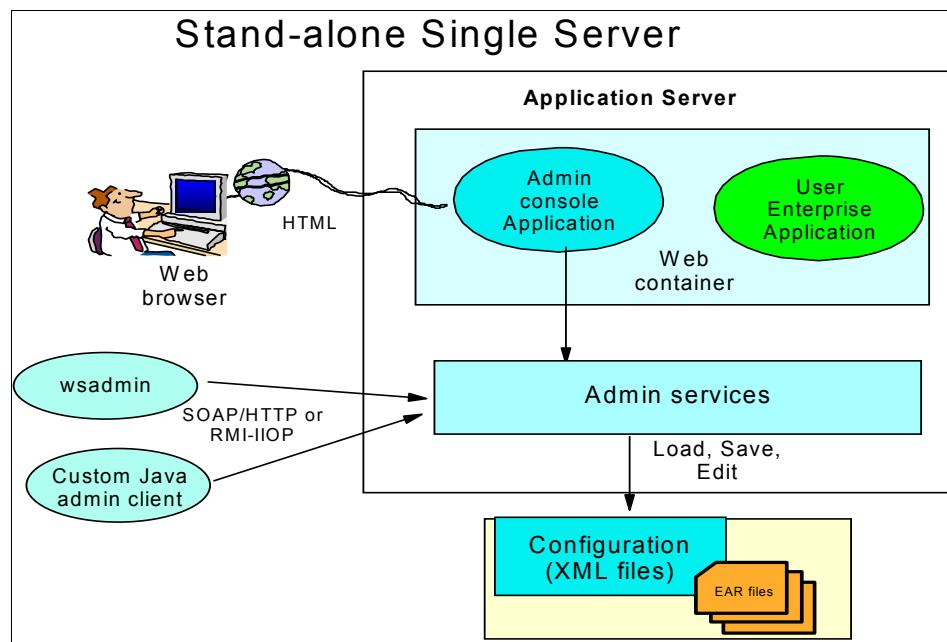


Figure 3-1 Managing a single-server installation

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.

3.1.3 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.

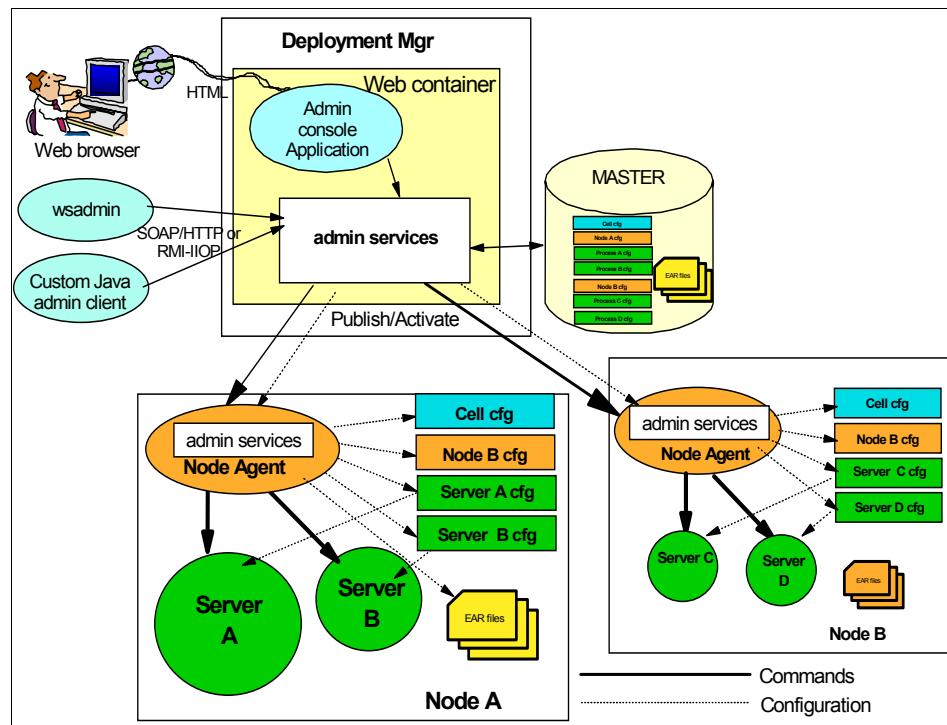


Figure 3-2 Managing a multi-server installation

Configuration should always be done at the deployment manager and synchronized out to the nodes. Although theoretically, it is 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.

3.1.4 Role-based administration

WebSphere Application Server provides a granularity of access control through the provision of four administrative security roles:

- ▶ *Monitor* can view the system state and configuration data, but cannot make any changes.
- ▶ *Operator* has all the functions of Monitor as well as ability to make operational changes, for example start/stop servers.
- ▶ *Configurator* has all the functions of Monitor as well as ability to make configurational changes.
- ▶ *Administrator* has all the functions of Operator and Configurator.

Using these roles requires that WebSphere global security be enabled. Users and groups can be assigned these roles through the administrative console.

3.2 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 1.4. 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.

3.2.1 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, etc.

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

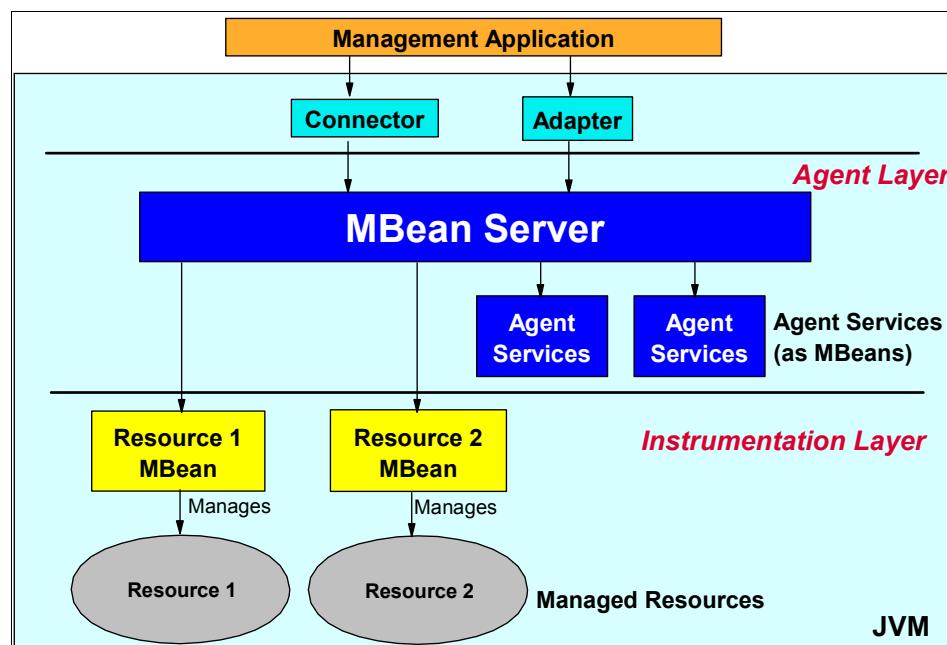


Figure 3-3 JMX architecture

How does JMX work?

Resources are managed by JMX MBeans. These are not EJBs, but simple JavaBeans 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 runtime resource. For instance, 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 JavaBeans 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 3-4.

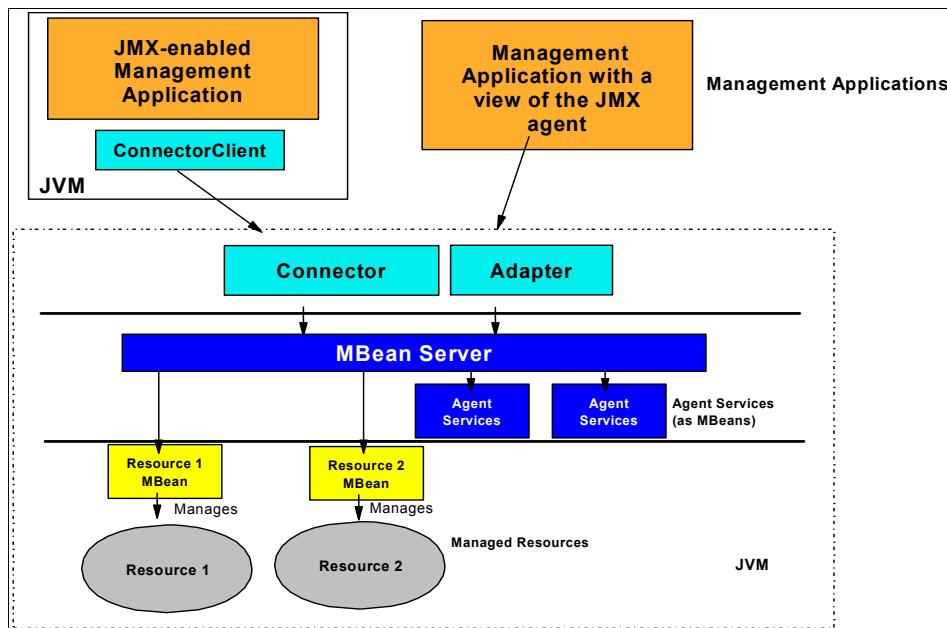


Figure 3-4 JMX connectors and adapters

3.2.2 JMX distributed administration

Figure 3-5 on page 89 shows how the JMX architecture fits into the overall distributed administration topology of a Network Deployment distributed server environment.

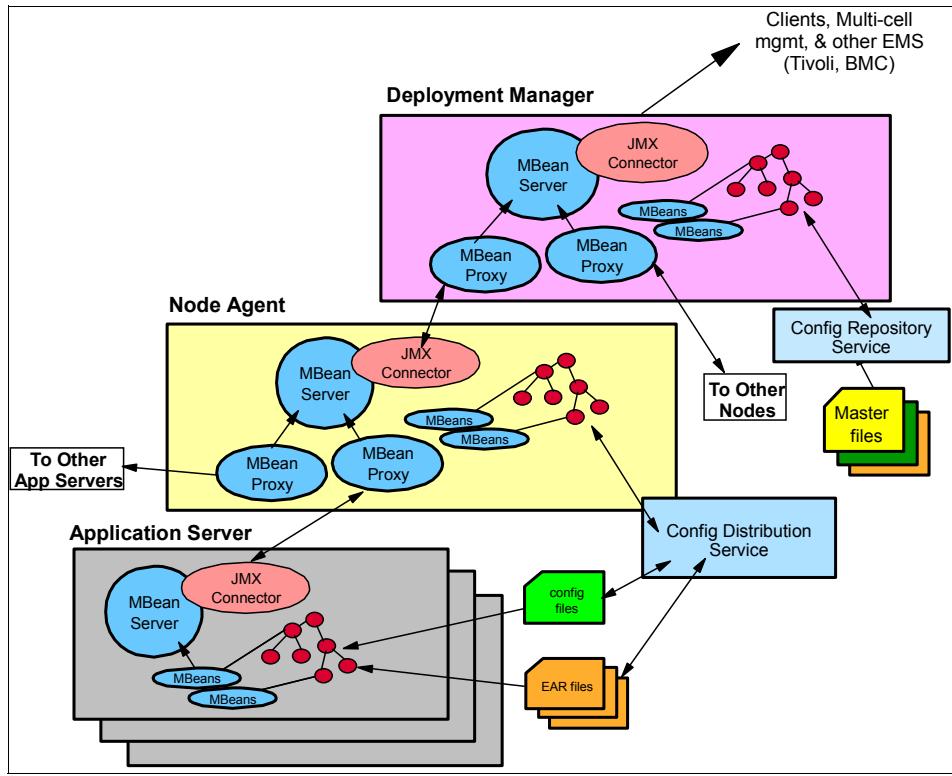


Figure 3-5 JMX distributed administration

The key points of this distributed administration architecture are:

- ▶ Internal MBeans which are 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 3-6 on page 90.

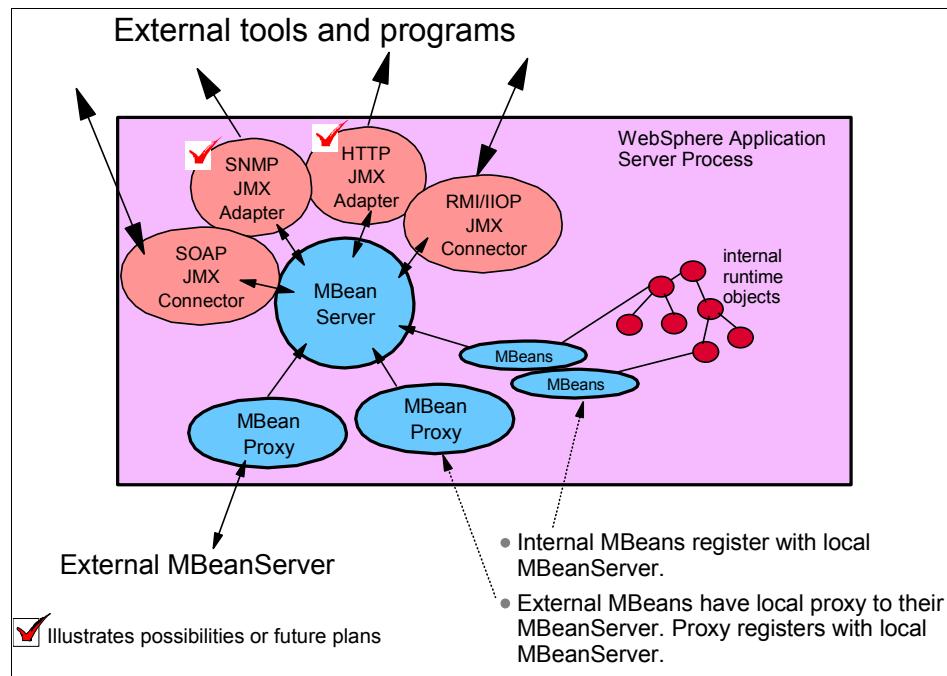


Figure 3-6 JMX architecture

3.2.3 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.

3.2.4 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 runtime 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.

3.2.5 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 the management standard MOs are essentially provided by mappings to existing WebSphere JMX MBeans. For instance, the specification require a J2EEServer managed object which 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 *Developing administrative programs for multiple Java 2 Platform, Server Foundation application servers* for a detailed description of the available objects and attributes).

A major requirement by the standard that does not easily map in to the existing WebSphere architecture is the ability to interoperate with management objects representing resources that have not been started in the WebSphere runtime environment. Consequently, a proxy mechanism has been introduced that runs in every application server in a standalone 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.

It is recommended 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.

3.3 Distributed administration

Administration in a distributed server environment is by necessity more complex than administration in a standalone 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 3-7.

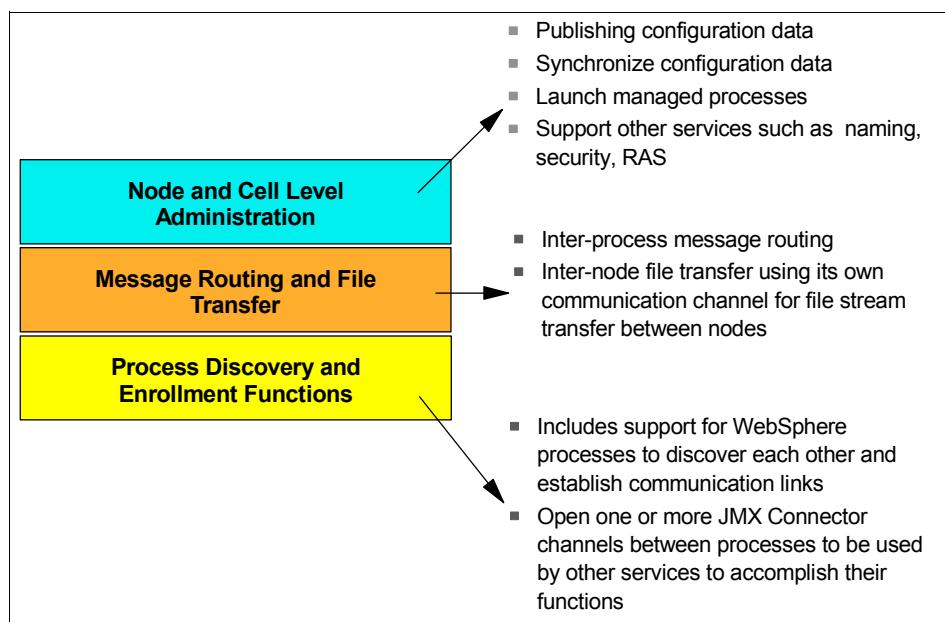


Figure 3-7 Layers of distributed administration services

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 3-8, where an operation on Node Y invokes a management method on a management bean (MBean) located on another node, Node X.

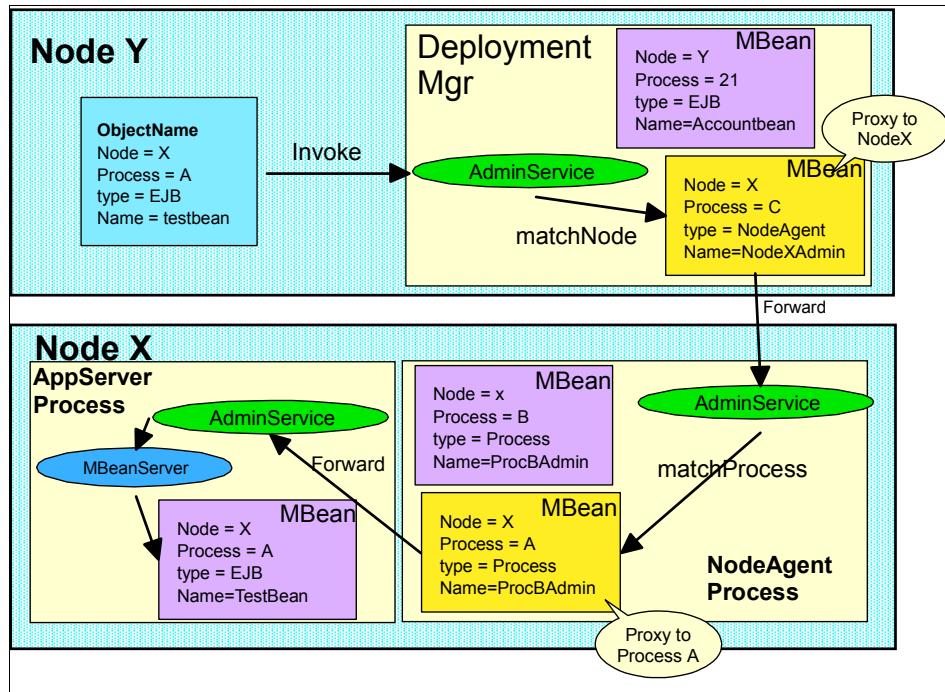


Figure 3-8 Distributed administration message routing

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.

3.3.1 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 3-9 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.

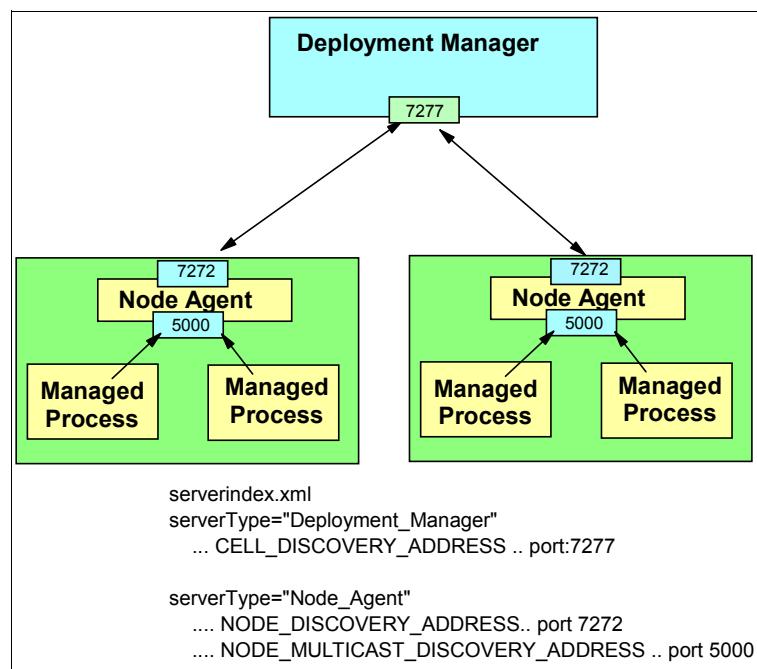


Figure 3-9 *Distributed discovery process*

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 when using the discovery service.

- ▶ 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 runtime 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.

3.3.2 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.

3.3.3 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 admin 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 admin 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 don't 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 6, “Administration with scripting” on page 267. The only thing you might need to know about wsadmin to complete these tasks is to start wsadmin to the SOAP connector port of the process you want to run the commands against. The default is to start to port 8879. If the process you are connecting to has a different port number specified, start the 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 c:\WebSphere\Application Server\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, 1047961605195, for example.

Example 3-1 Resetting the master cell repository

```
C:\WebSphere\AppServer\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=6.0.0.0,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 3-2 Resetting the node repository

```
C:\WebSphere\AppServer\profiles\appSrv01\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=6.0.0.0,name=repository,mbeanId  
entifier=repository,type=ConfigRepository,node=AppSrvrNode,process=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 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 standalone program which runs separately from the node agent. It has no cache of epoch values which could be used for an optimized synchronize, therefore performing a complete synchronize. For this same reason, if you restart a node agent, the very first synchronize it performs will always be a complete synchronize. 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 3-3 Using the syncNode command

```
C:\WebSphere\AppServer\profiles\AppSrv01\bin>stopnode  
ADMU0116I: Tool information is being logged in file  
  
C:\WebSphere\AppServer\profiles\AppSrv01\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.
```

```
C:\WebSphere\AppServer\profiles\AppSrv01\bin>syncnode carlavn2  
ADMU0116I: Tool information is being logged in file  
C:\WebSphere\AppServer\profiles\AppSrv01\logs\syncNode.log  
ADMU0128I: Starting tool with the AppSrv01 profile  
ADMU0401I: Begin syncNode operation for node AppSrvrNode with Deployment  
Manager carlavn2: 8879  
ADMU0016I: Synchronizing configuration between node and cell.  
ADMU0402I: The configuration for node AppSrvrNode has been synchronized with  
Deployment Manager carlavn2: 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.

3.4 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.

3.4.1 Repository directory structure

With V6, the directory structure of a WebSphere Application Server installation is slightly different than in previous releases. We will discuss this in detail in Chapter 4, “Getting started with profiles” on page 113, but for now, it is important to know configuration files defining a runtime environment are stored in profile directories. Each node, deployment manager, and standalone application server has its own profile directory under the <was_home>/profiles directory.

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 WebSphere cell. You can see this in Figure 3-10. The repository structure follows the same format, regardless of whether you have a standalone server environment or distributed server environment.

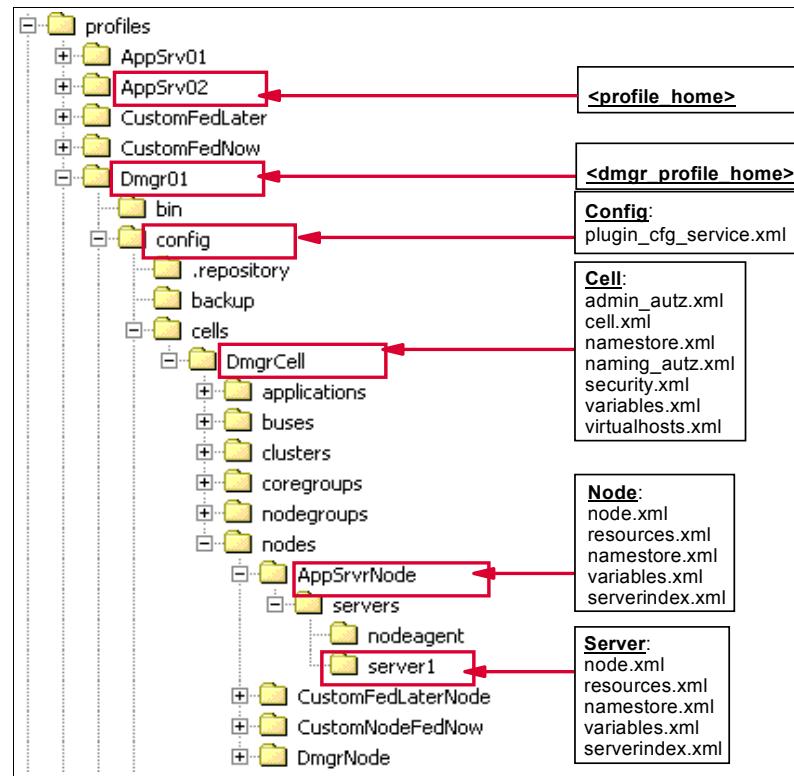


Figure 3-10 Repository directory structure

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 configuration for the cell. The directory contains a number of cell-level configuration settings files. Depending on the types of resources 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 standalone server environment and a distributed server environment. The differences are summarized in the following sections.

In a standalone 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:

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

3.4.2 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.

3.4.3 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 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.

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, 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.

However, 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, JSPs)

Figure 3-11 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.

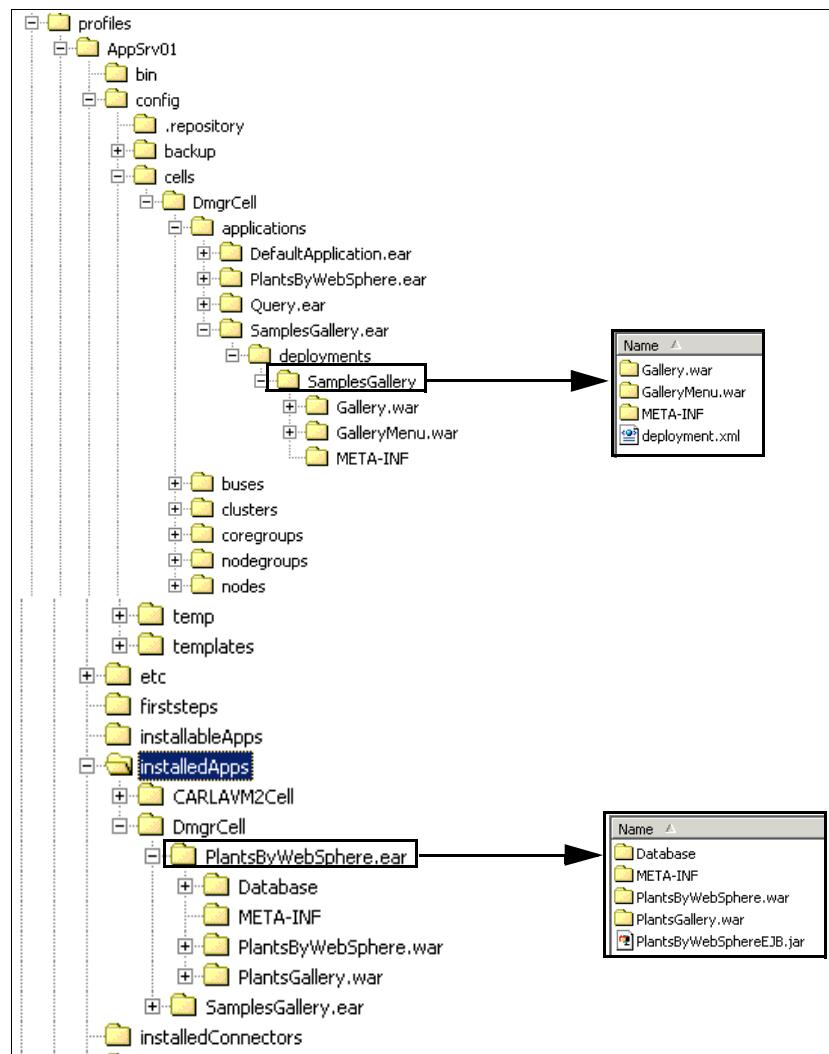


Figure 3-11 Location of application data files

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 standalone 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*, SG24-6446. 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:

- ▶ 4.1, “Understanding profiles” on page 114
- ▶ 4.2, “Building a system using profiles” on page 119
- ▶ 4.3, “Creating profiles” on page 121
- ▶ 4.4, “Creating profiles manually” on page 151
- ▶ 4.5, “Managing the processes” on page 154

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

4.1 Understanding profiles

New with WebSphere Application Server V6 is the concept of profiles. The WebSphere Application Server installation process simply lays down a set of core product files required for the runtime processes. After installation, you need to create one or more *profiles* that define the runtime to have a functional system. The core product files are shared among the runtime components defined by these profiles.

Note: In V5, the `wsinstance` command is used to create multiple runtime configurations using the same installation. With V6, you do this with profiles.

With Base and Express you can only have standalone application servers as shown in Figure 4-1. 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.

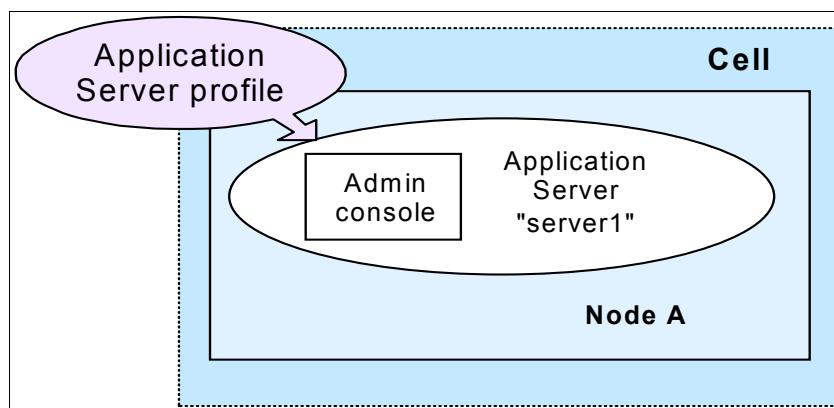


Figure 4-1 System management topology: Standalone server (Base and Express)

You can also create standalone 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.

With the Network Deployment package, you have the option of defining multiple application servers with central management capabilities as summarized in Figure 4-2 on page 115. 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 it is recommended that the deployment manager be placed on a separate dedicated machine.

The basis for this runtime 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.

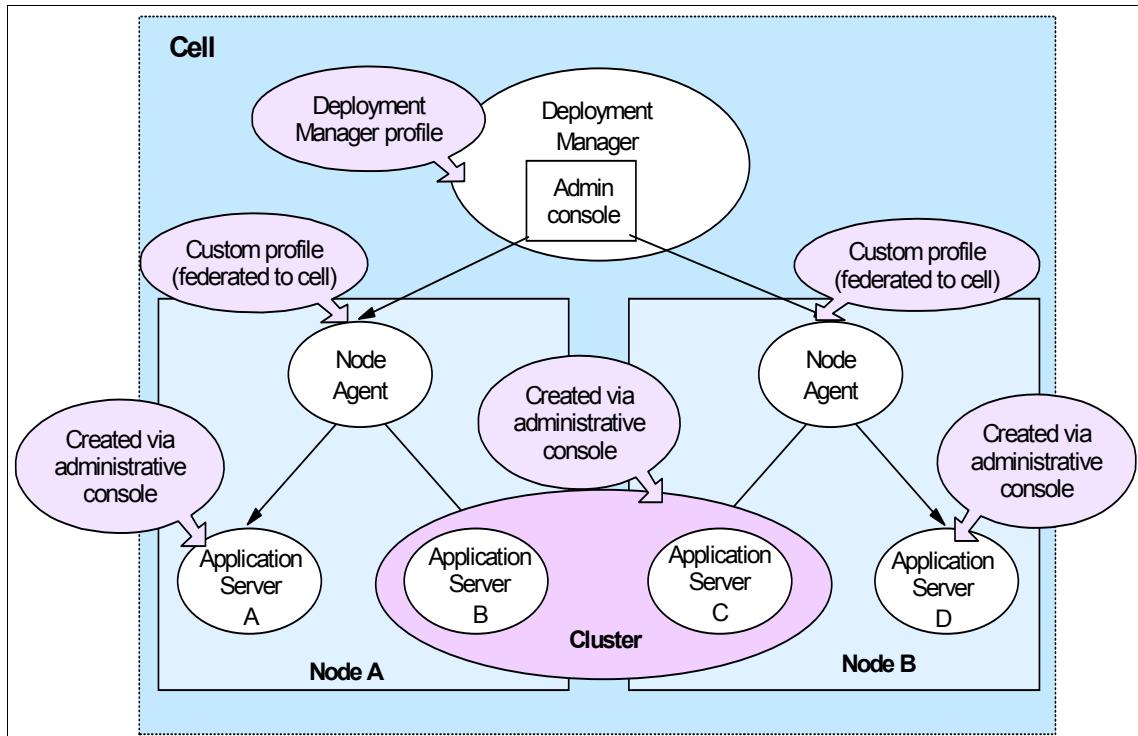


Figure 4-2 System management topology: Network Deployment

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

- ▶ You can define a custom profile to create an empty node for federation to the cell. After federation, you can further configure the node. For example, create application servers and clusters from the deployment manager administrative console.
- ▶ 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.

4.1.1 Types of profiles

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

Application server profile

The application server profile defines a single standalone application server. Using this profile gives you an application server that can run standalone, or unmanaged, with 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 automatically installed on the server.
- ▶ The server has a dedicated administrative console.

The primary use for this type of profile is:

- ▶ To build a server in a Base or Express installation.
- ▶ To build a standalone 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 as server1 and the administrative console is removed from the application server.

Deployment manager profile

The deployment manager profile defines a deployment manager in a Network Deployment installation. Although you could conceivably have the Network Deployment package and run only standalone 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.

4.1.2 Directory structure and default profiles

If you have worked with WebSphere Application Server before, 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:\WebSphere\AppServer. In this book, we refer to it 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. 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 runtime under each profile.

For example, if you installed on a Windows system, and created a profile named AppSrvr01, you would see a directory structure like that shown in Figure 4-3 on page 118.

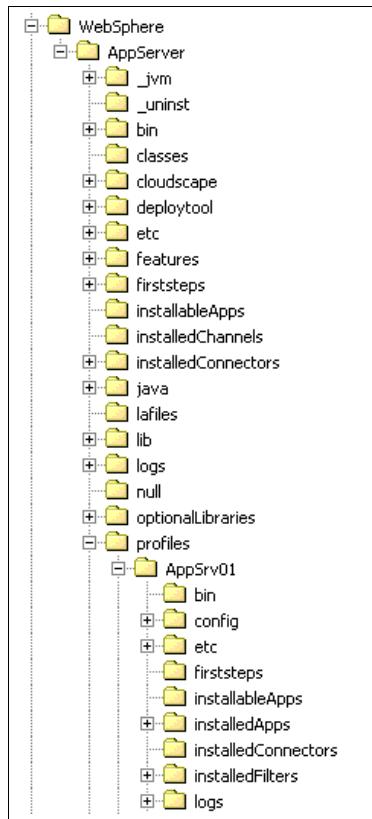


Figure 4-3 Directory structure

We refer to the root of each profile directory (`WebSphere/AppServer/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 runtime 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 `wasprofile` command to specify the default profile.
- ▶ If you have not specified the default profile, it will be the first profile you create.

To make sure command line operations are executed for the correct runtime, 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>` directory.

4.2 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. 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 a standalone server environment with one or more standalone application servers. We will refer to this as a *standalone server environment*.
- ▶ You plan to create a distributed server environment consisting of 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.

4.2.1 Standalone server environment

If you are creating a standalone application server, do the following:

1. Install your choice of Base, Express, or Network Deployment on the system.
2. Create an application server profile on that system.

4.2.2 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, you need to ensure that the ports you select for each are unique. For more information about ports, see *Planning and Designing for WebSphere Application Server V6*, SG24-6446.

Method 1

This method assumes that you do not have a standalone 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 server during creation. You can select a name for each server because 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.

1. Install WebSphere Application Server Network Deployment on server. If this is a multiple-machine install with 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 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 custom profile wizard, 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 WebSphere Application Server 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.

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

4.3 Creating profiles

This section shows to create profiles using the Profile creation wizard.

Silent install: You can also create profiles in silent mode using the `wasprofile` command (see , “Creating a profile in silent mode” on page 153).

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

1. From the Start menu in Windows only, select:
Start → Programs → IBM WebSphere → Application Server Network Deployment v6 → Profile creation wizard
2. Use the platform-specific command in the `<was_home>/bin/ProfileCreator` directory:
 - Windows: `pctWindows.exe`
 - AIX: `pctAIX.bin`
3. Check the box directly after installation from the install wizard to launch the Profile creation wizard.
4. When you start the wizard, the first screen you see is the Welcome screen. Click **Next** to select the type of profile you will create, as in Figure 4-4 on page 122.

Profile type selection

A profile defines a run-time environment. Choose the type of profile that is best for your needs.

Create a deployment manager profile

The first step in setting up a Network Deployment environment is to create a deployment manager. A deployment manager administers Application Servers that are federated into (made a part of) its cell.

The next step is to create additional profiles and federate them.

Create an Application Server profile

An Application Server runs your enterprise applications.

The Application Server can be managed from its own administrative console and function independent of other Application Servers and deployment managers. Alternatively, it can be federated into a cell and managed by the deployment manager.

Create a custom profile

A custom profile contains an empty node, which does not contain an administrative console or any servers. The typical use for a custom profile is to federate its node to a deployment manager.

After federating the node, use the deployment manager to create a server or a cluster of servers within the node.

Figure 4-4 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.

First Steps: At the end of the Profile creation wizard 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 creation wizard, and installation verification.

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

<was_install>\profiles\<profile_name>\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.

4.3.1 Creating a deployment manager profile

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

1. Start the Profile creation wizard and click **Next** on the Welcome page.
2. Select the **Create a deployment manager profile** option. Click **Next**.
3. 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 4-5. Click the box if you want this to be the default profile for receiving commands. Click **Next**.

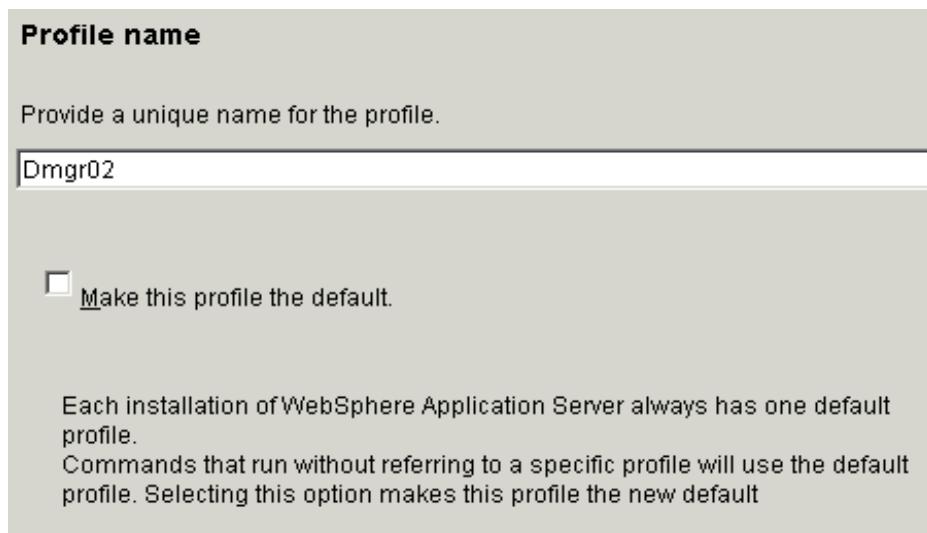


Figure 4-5 Creating a deployment manager profile: Enter a name

4. In the next screen, enter a directory in which to store the profile or accept the default. See Figure 4-6. Click **Next**.

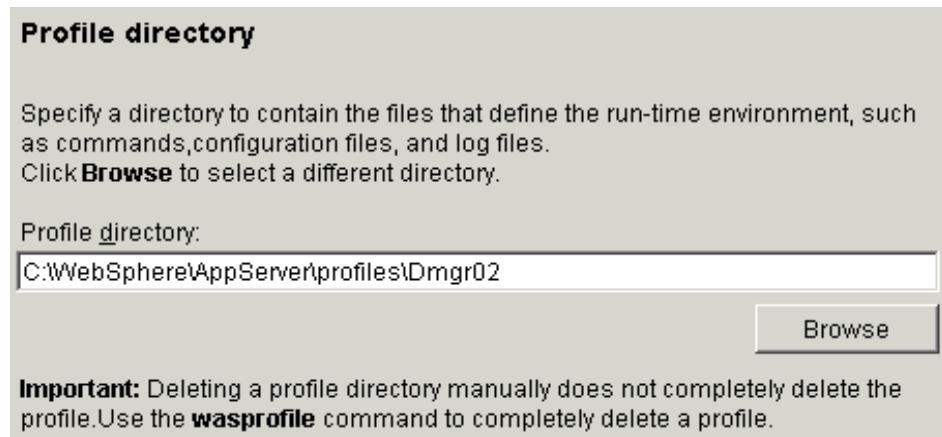


Figure 4-6 Creating a deployment manager profile: Enter a profile directory

5. Enter the node, host, and cell names. These default based on the hostname 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 4-7 on page 125.

Node, host, and cell names

Specify a node name,a host name, and a cell name for this profile. Refer to the installation guide for detailed field descriptions and migration considerations.

Node name:

CARLAVM2CellManager02

Host name:

CARLAVM2.itso.ral.ibm.com

Cell name:

CARLAVM2Cell02

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

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

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

Figure 4-7 Creating a deployment manager profile: Enter cell, host, and node names

Click **Next**.

6. 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 4-8 on page 126.

Port value assignment

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.

Administrative console port (Default 9060):	9060
Administrative console secure port (Default 9043):	9043
Bootstrap port (Default 9809):	9809
SOAP connector port (Default 8879):	8879
SAS SSL ServerAuth port (Default 9401):	9401
CSIV2 ServerAuth listener port (Default 9403):	9403
CSIV2 MultiAuth listener port (Default 9402):	9402
ORB listener port (Default 9100):	9100
Cell discovery port (Default 7277):	7277
High availability manager communication port (Default 9352):	9352

Figure 4-8 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*.

7. 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 4-9 on page 127,

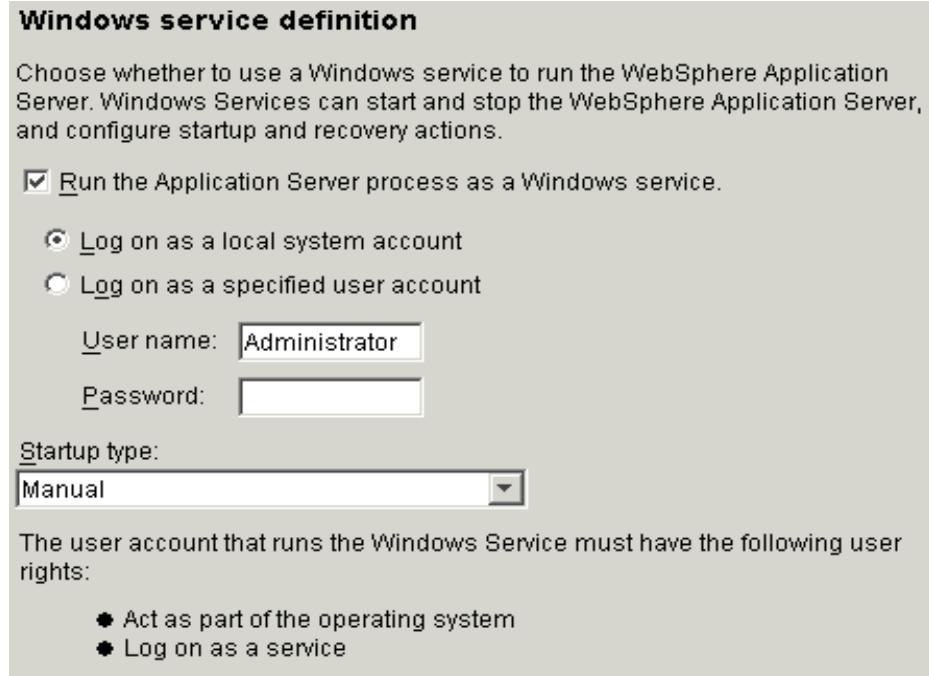


Figure 4-9 Creating a deployment manager profile: Run as a Windows service

Note that the panel lists the user rights the user ID you select needs to have. If the user ID doesn't have these rights, the wizard will automatically add them.

Click **Next**.

8. Review the options you have chosen and click **Next** to create the profile. After the wizard has finished, you will be presented with the screen in Figure 4-10 on page 128.

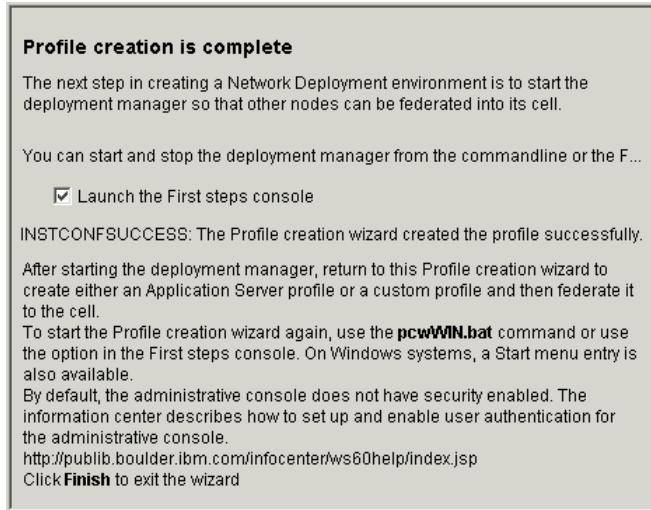


Figure 4-10 Creating a deployment manager profile: Finish

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

`<was_home>/logs/wasprofile/wasprofile_create_<profile_name>.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 as in Figure 4-11.

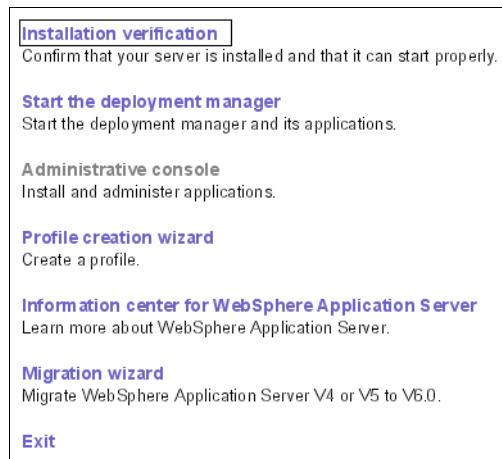


Figure 4-11 Deployment manager First Steps menu

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. You will find it at `<was_home>/profiles/<profile_name>`. In this book, we refer to it 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 creation wizard. See Figure 4-8 on page 126.

Click the **Log in** button. Because security is not active at this time, 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.

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 won't 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.

4.3.2 Creating an application server profile

An application server profile defines a new standalone application server. This server can be run standalone or can be later federated to a deployment manager cell for central management. This section takes you through the steps of creating the application server profile.

1. Start the Profile creation wizard. Click **Next** on the Welcome page.
2. Select the **Create an Application server profile** option. Click **Next**.
3. 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 4-12.

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

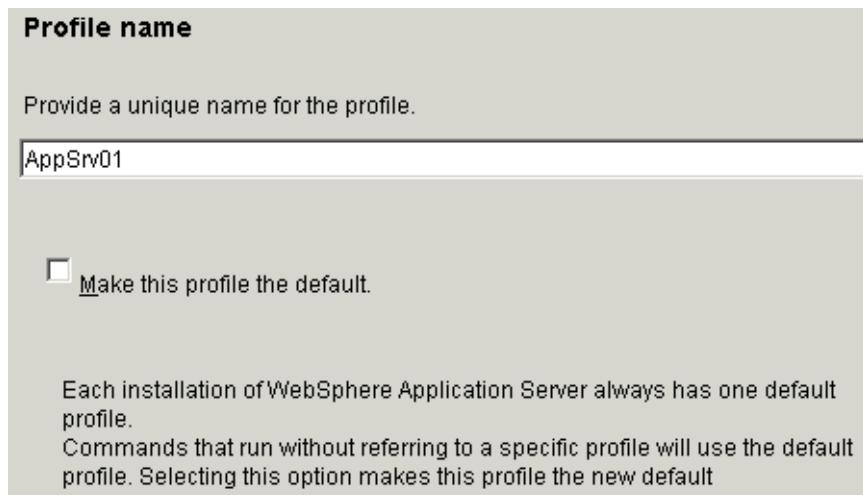


Figure 4-12 Creating an application server profile: Enter a name

4. In the next screen, enter a directory to store the profile in or accept the default and click **Next**. See Figure 4-13.

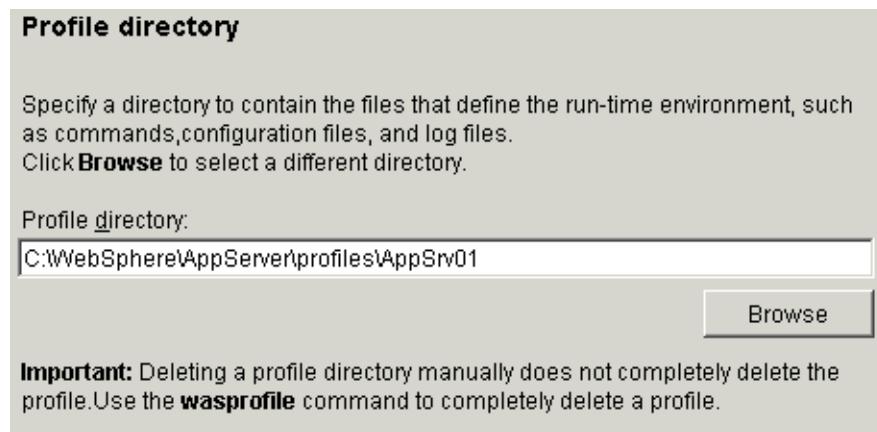


Figure 4-13 Creating an application server profile: Enter a profile directory

5. Enter the new node name and the system host name. See Figure 4-14. The node name will default based on the hostname 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**.

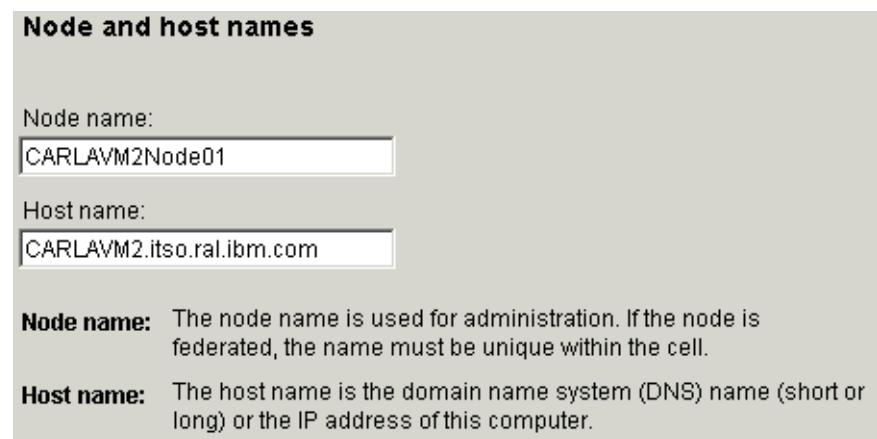


Figure 4-14 Creating an application server profile: Enter host and node names

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

6. The wizard will present a list of TCP/IP ports for use by the application server, as in Figure 4-15. If you already have existing profiles on the system, 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.

Port value assignment

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.

HTTP transport port (Default 9080):	9080
Administrative console port (Default 9060):	9061
HTTPS transport port (Default 9443):	9443
Administrative console secure port (Default 9043):	9044
Bootstrap port (Default 2809):	2809
SOAP connector port (Default 8880):	8880
SAS SSL ServerAuth port (Default 9401):	9404
CSIV2 ServerAuth listener port (Default 9403):	9405
CSIV2 MultiAuth listener port (Default 9402):	9406
ORB listener port (Default 9100):	9101
High availability manager communication port (Default 9353):	9353
Service Integration Port (Default 7276):	7276
Service Integration Secure Port (Default 7286):	7286
Service Integration MQ Interoperability Port (Default 5558):	5558
Service Integration MQ Interoperability Secure Port (Default 5578):	5578

Figure 4-15 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*.

7. 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 in Figure 4-16.

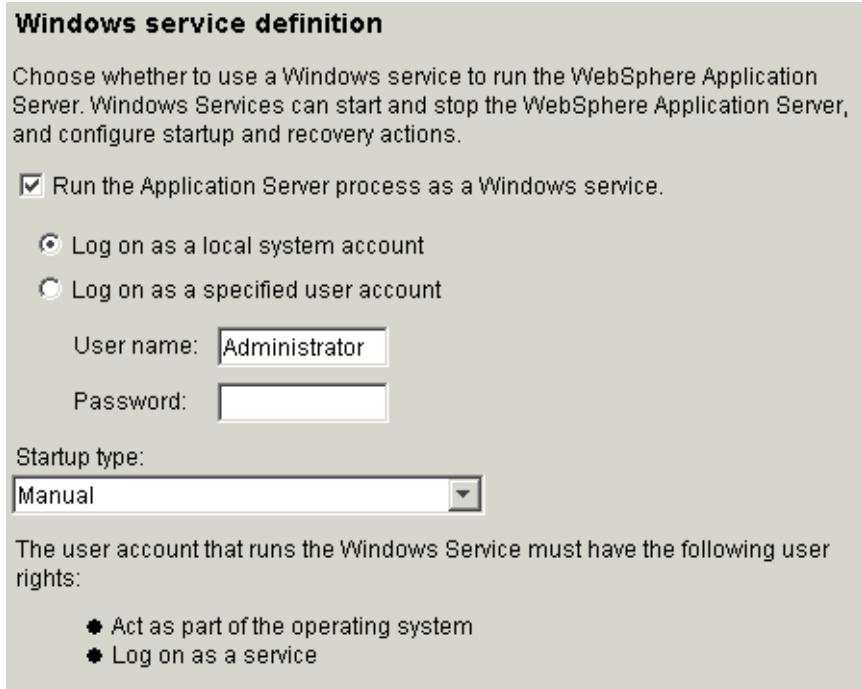


Figure 4-16 Creating an application server profile: Run as a service

Note that the panel 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**.

8. Review the options you have chosen. and click **Next** to create the profile. See Figure 4-17.

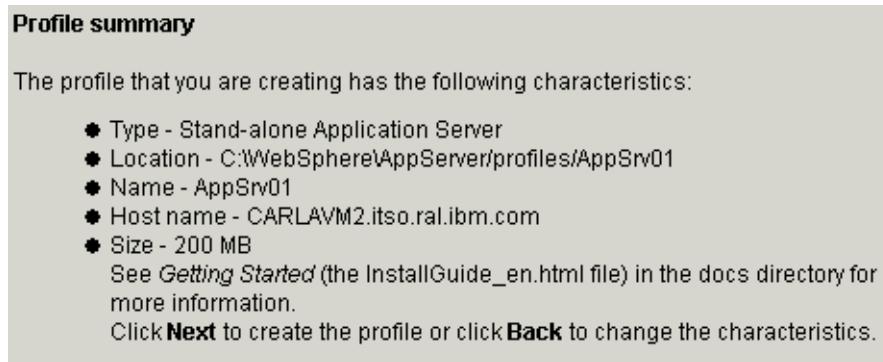


Figure 4-17 Creating a deployment manager profile: Finish

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

If you have errors during the Profile creation wizard, check the log at:

`<was_home>/logs/wasprofile/wasprofile_create_<profile_name>.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`

9. Click **Finish** to close the wizard and start the First Steps application. See Figure 4-18 on page 135.

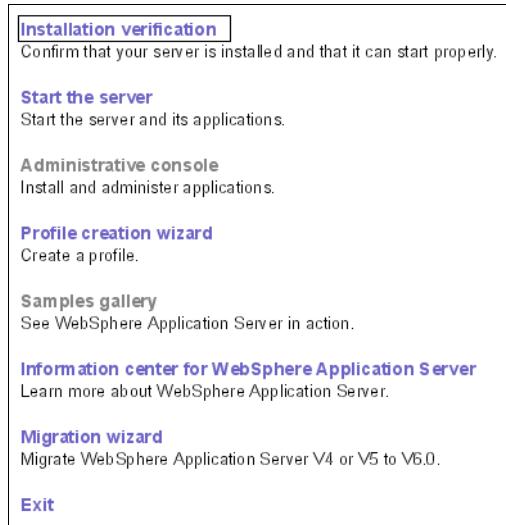


Figure 4-18 Application server First Steps menu

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. You will find it at `<was_home>/profiles/<profile_name>`. In this book, 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 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 creation wizard (see Figure 4-15 on page 132).

Click the **Log in** button. Because security is not active at this time, you do not have to enter a user name. If you choose to enter a name, it can be any name. If you enter a name, it will be used to track changes you made to the configuration.

5. Display the configuration from the console. See Figure 4-19. 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.

The screenshot shows the WebSphere Administrative Console interface. At the top, there is a navigation bar with links for Welcome, Logout, Support, and Help. Below the navigation bar is a left-hand sidebar with several categories: Servers (selected), Applications, Resources, Security, Environment, and System administration. Under the Servers category, there are links for Application servers and Web servers. The main content area is titled "Application servers". It has a header bar with "Application servers" and a help icon. Below the header, there is a brief description: "An application server is a server which provides services required to run enterprise applications." There is also a "Preferences" link. The main part of the screen is a table listing application servers. The table has columns for Select, Name, Node, and Version. One row is visible, showing "server1" as the name, "CARLAVM2Node01" as the node, and "6.0.0.0" as the version. A total count of "1" is displayed at the bottom of the table. The entire interface is presented in a light blue and white color scheme.

Figure 4-19 Application server defined by the application server profile

- b. Enterprise applications

Select **Applications** → **Enterprise Applications**. See Figure 4-20. You should see a list of applications. These are the WebSphere sample applications.

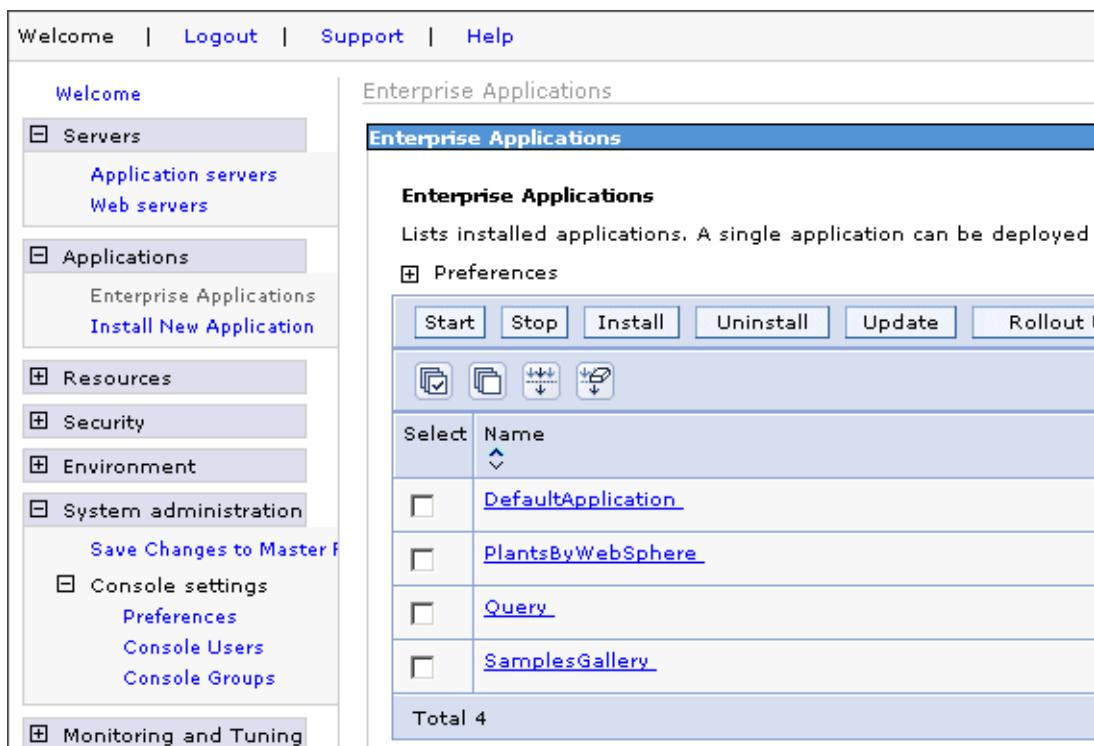


Figure 4-20 Applications installed on server1

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
```

4.3.3 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 working a deployment manager.

Note: With other profiles, you have the option of registering the processes as Windows services. This doesn't appear as an option when you create a custom profile. If you want to register the node agent as a Windows service later, see 4.5.3, "Enabling process restart on failure" on page 157.

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

1. Start the Profile creation wizard. Click **Next** on the Welcome page.
2. Select the **Create a custom profile** option. Click **Next**.
3. 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 4-8 on page 126) for the deployment manager. See Figure 4-21.

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

Federation

A custom profile contains an empty node that must be federated to a deployment manager to become a functional managed node. Identify a running deployment manager that will administer the node or choose to federate the node later using the `addNode` command.

The host name or the IP address for the deployment manager:

localhost

The SOAP port for the deployment manager (8879 is the default):

8879

Federate this node later using the `addNode` command

Figure 4-21 Creating a custom profile: Federate now or later

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 4-22.
Click the box if you want this to be the default profile for receiving commands. Click **Next**.

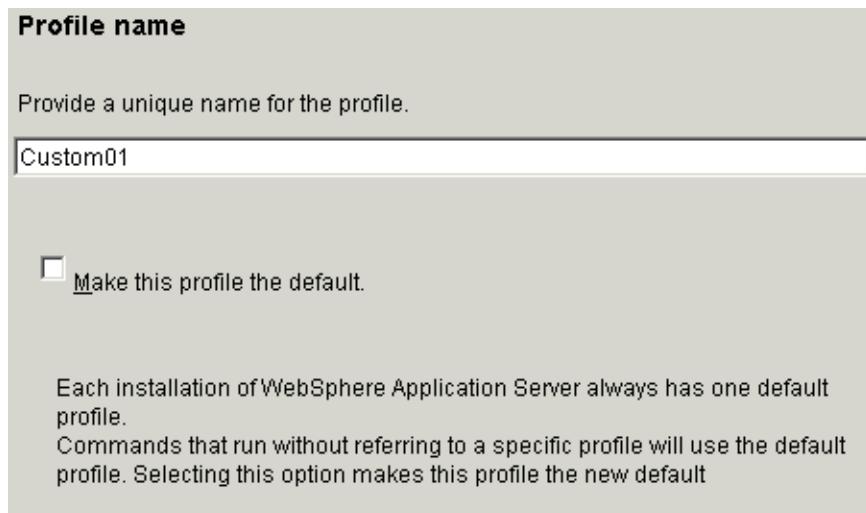


Figure 4-22 Creating a custom profile: Enter a name

5. In the next screen, Figure 4-23, enter a directory in which to store the profile or accept the default and click **Next**.

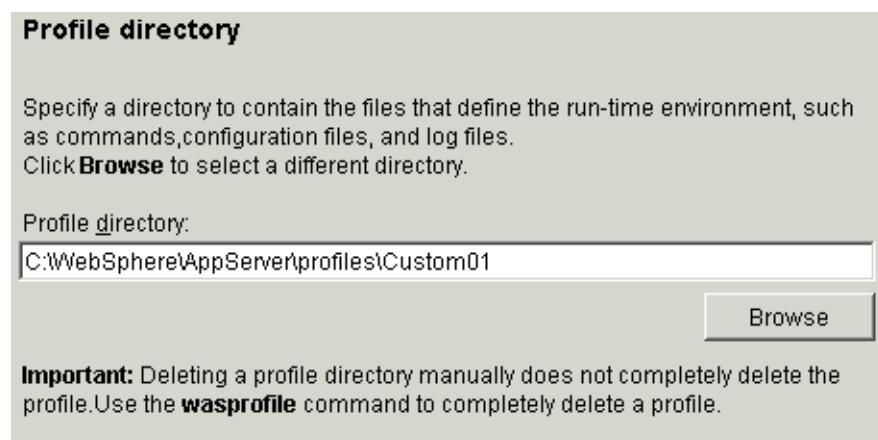


Figure 4-23 Creating a custom profile: Enter a profile directory

6. Enter the new node name and the system host name. See Figure 4-24. The node name defaults based on the hostname 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**.

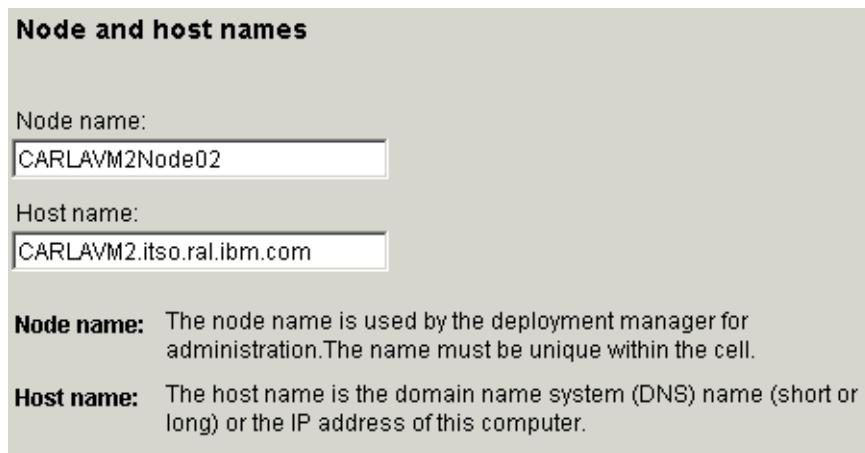


Figure 4-24 Creating a custom profile: Enter host, and node names

7. The wizard presents a list of TCP/IP ports for use by the node agent for this node. 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 are unique on the system. See Figure 4-25 on page 141.

Port value assignment

The values in the following fields define the ports for the node agent 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.

Bootstrap port (Default 2809):	2810
SOAP connector port (Default 8879):	8881
SAS SSL ServerAuth port (Default 9901):	9901
CSIV2 ServerAuth listener port (Default 9201):	9201
CSIV2 MultiAuth listener port (Default 9101):	9102
ORB listener port (Default 9100):	9103
Node discovery port (Default 7272):	7272
Node multicast discovery port (Default 5000):	5000
Node IPv6 multicast discovery port (Default 5001):	5001
High availability manager communication port (Default 9353):	9354

Figure 4-25 Creating a custom profile: Select ports

Note: If you did not choose to federate the new node as part of this wizard, you will not see this panel.

8. Review the options you have chosen. See Figure 4-26.

Profile summary

The profile that you are creating has the following characteristics:

- Type - Managed profile (Federate a managed profile to create a managed node.)
 - Location - C:\WebSphere\AppServer\profiles\Custom01
 - Name - Custom01
 - Host name - CARLAVM2.itso.ral.ibm.com
 - Size - 10 MB
See *Getting Started* (the InstallGuide_en.html file) in the docs directory for more information.
- Click **Next** to create the profile or click **Back** to change the characteristics.

Figure 4-26 Creating a custom profile: Summary

Click **Next** to create the profile.

9. After the wizard has finished, you will be presented with a screen containing messages indicating the success or failure of the process. See Figure 4-27.

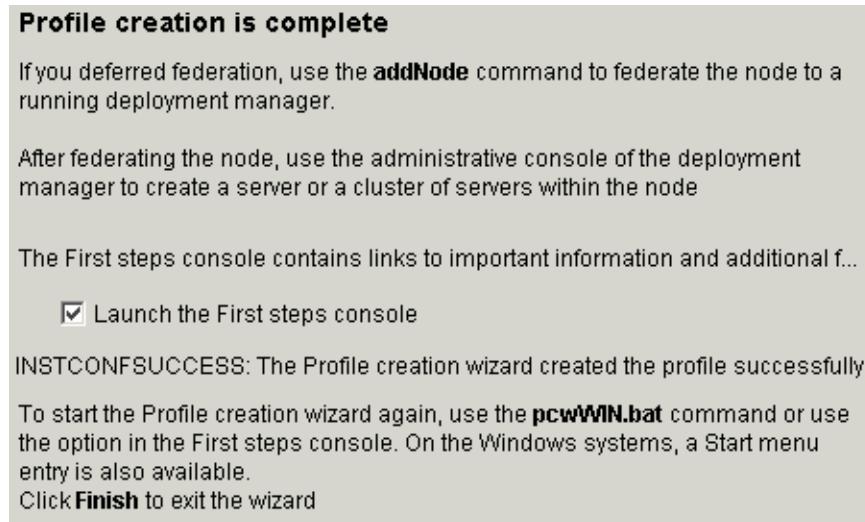


Figure 4-27 Creating a custom profile: Finish and launch First Steps

If you have errors during the Profile creation wizard, check the log at:

`<was_home>/logs/wasprofile/wasprofile_create_<profile_name>.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`

10. Click **Finish** to close the wizard and start the First Steps application. See Figure 4-28 on page 143.

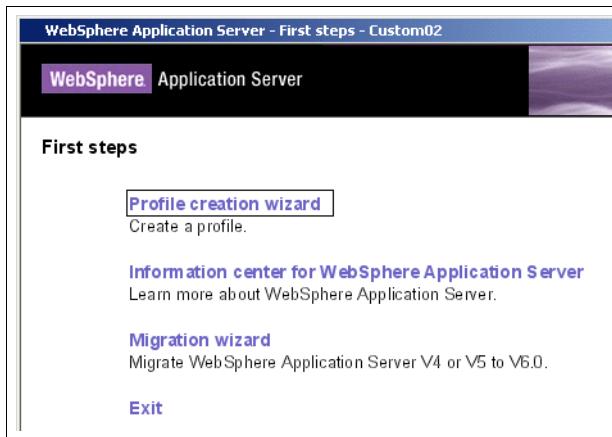


Figure 4-28 Custom profile First Steps window

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 directory structure and find the new profile. You will find it at `<was_home>/profiles/<profile_name>`. In this book we refer to it as `<profile_home>`. 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 4-29 on page 144.

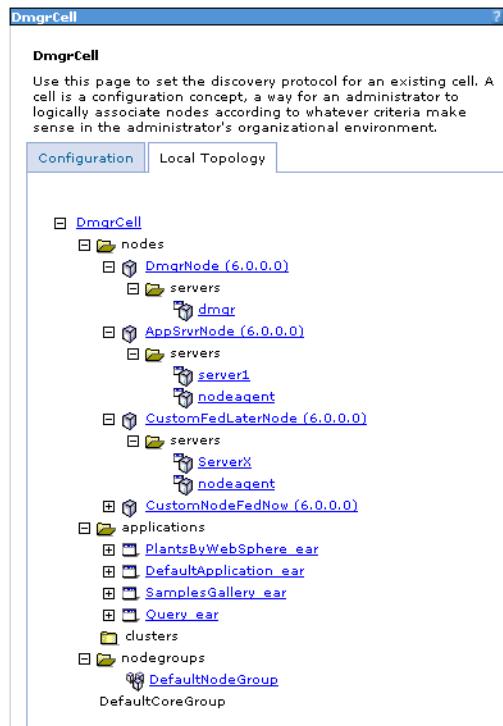


Figure 4-29 Topology view of the cell

3. The federation process creates a node agent for the new node, federate it to the cell, and start 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 the next section, 4.3.4, “Federating a custom node to a cell” on page 145. Otherwise, you can continue by defining an application server on the new node. To do this, see 4.3.5, “Creating a new application server on an existing node” on page 146.

4.3.4 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 unfederated custom profile defines a node that can be added to a cell. To federate a custom 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 by typing the following:
cd websphere\appserver\profiles\customprof1\bin
3. Run the addNode command. Here you need the host name of the deployment manager and the SOAP connector address (see Figure 4-7 on page 125 and Figure 4-8 on page 126).

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

See Example 4-1 for sample output.

Example 4-1 Federating a custom profile to a cell

```
C:\WebSphere\AppServer\profiles\CustomFedLater\bin>addnode carlavn2 8879
ADMU0116I: Tool information is being logged in file
          C:\WebSphere\AppServer\profiles\CustomFedLater\logs\addNode.log
ADMU0128I: Starting tool with the CustomFedLater profile
ADMU0001I: Begin federation of node CustomFedLaterNode with Deployment Manager
at carlavn2:8879.
ADMU0009I: Successfully connected to Deployment Manager Server: carlavn2:8879
ADMU0507I: No servers found in configuration under:
```

```
C:\WebSphere\AppServer/profiles/CustomFedLater\config/cells/CARLAVM2C
e11/nodes/CustomFedLaterNode/servers
ADMU2010I: Stopping all server processes for node CustomFedLaterNode
ADMU0507I: No servers found in configuration under:
```

```
C:\WebSphere\AppServer/profiles/CustomFedLater\config/cells/CARLAVM2C
e11/nodes/CustomFedLaterNode/servers
ADMU0024I: Deleting the old backup directory.
ADMU0015I: Backing up the original cell repository.
ADMU0012I: Creating Node Agent configuration for node: CustomFedLaterNode
ADMU0014I: Adding node CustomFedLaterNode configuration to cell: DmgrCell
ADMU0016I: Synchronizing configuration between node and cell.
ADMU0018I: Launching Node Agent process for node: CustomFedLaterNode
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:
1880
ADMU9990I: ADMU0300I: Congratulations! Your node CustomFedLaterNode has been
successfully incorporated into the DmgrCell cell.
ADMU9990I: ADMU0306I: Be aware:
ADMU0302I: Any cell-level documents from the standalone CARLAVM2Cell
configuration have not been migrated to the new cell.
ADMU0307I: You might want to:
ADMU0303I: Update the configuration on the DmgrCell 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 DmgrCell cell using wsadmin \$AdminApp
or the Administrative Console.
ADMU9990I:

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
```

4.3.5 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 5.4, “Working with application servers” on page 190.

If you plan to use clustering, you can create application servers when you create the cluster. For information about working with clusters, see 5.6, “Working with clusters” on page 235.

1. Ensure the custom profile node agent is started.
2. Open the deployment manager administrative console.
3. Select **Servers** → **Application Servers**
4. Click **New**. See Figure 4-30.
5. Select the custom profile node and enter a server name. Click **Next**.

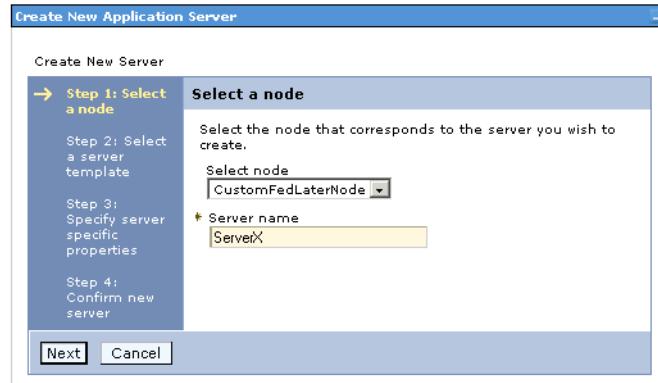


Figure 4-30 Creating a new server: Enter a node and name

6. Select a template to use as a basis for the new application server configuration. If you haven't previously set up a template based on an existing application server, select the default template. Click **Next**. See Figure 4-31.

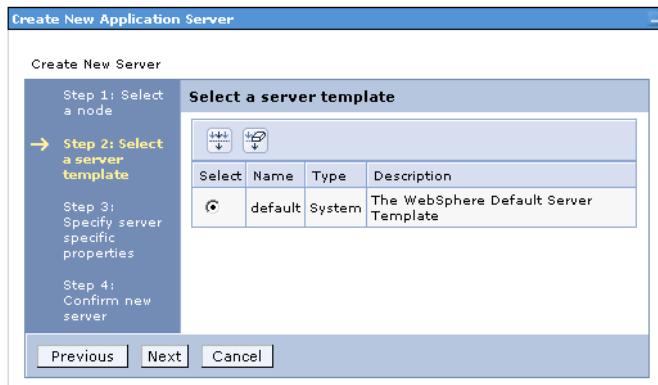


Figure 4-31 Creating a new server: Select a template

7. Each application server on a node must have unique ports assigned. The next screen gives you the option of having unique ports generated for this application server, as opposed to the default set. Click **Next**. See Figure 4-32 on page 148.

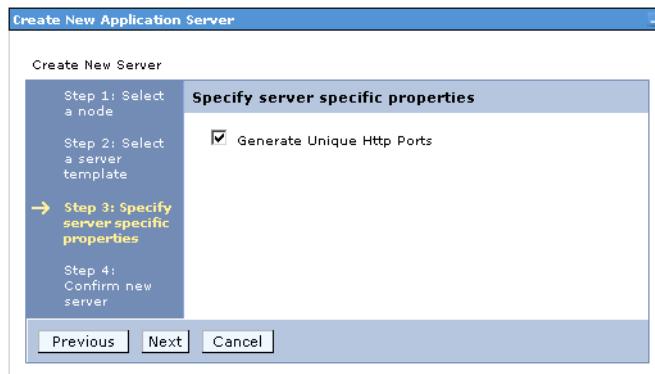


Figure 4-32 Creating a new server: Generate unique ports

8. The last screen summarizes your choices. See Figure 4-33. Click **Finish** to create the profile.

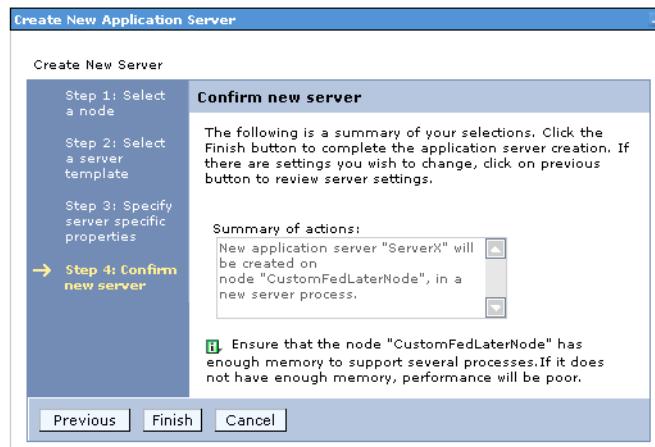


Figure 4-33 Creating a new server: Summary and finish

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 are installed automatically 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.

4.3.6 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**. See Figure 4-35 on page 150.
7. Enter the host name and SOAP connector port specified when you created the application server profile. See Figure 4-14 on page 131 and Figure 4-15 on page 132.

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.

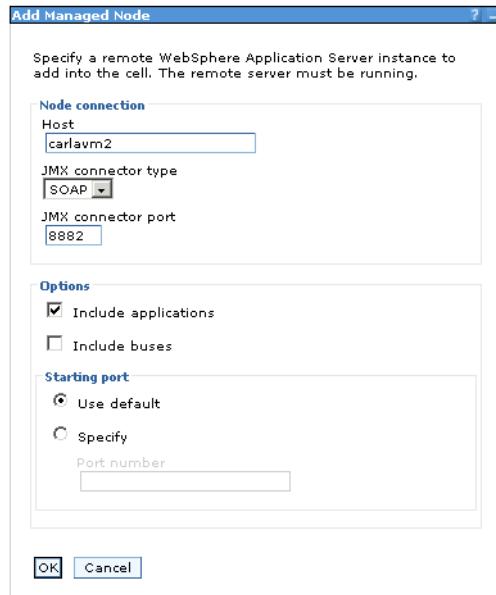


Figure 4-34 Adding a standalone application profile to a cell

Click **OK**.

8. If the node is a Windows node, in Figure 4-35 you have the opportunity to register the new node agent as a Windows service. Make your selection and click **OK**.

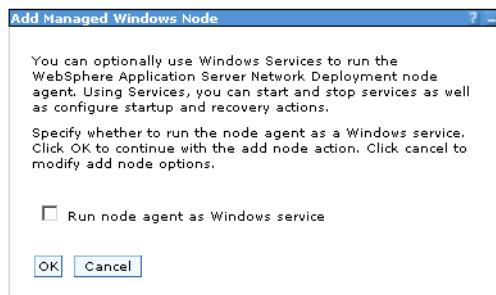


Figure 4-35 Run a node agent as a Windows service

The federation process stops the application server. It creates a new node agent for the node, 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.

4.4 Creating profiles manually

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 creation wizard. At the heart of this wizard is the wasprofile command, also known as the profile creation tool. This command provides you 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.

4.4.1 Using the wasprofile command

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

Syntax

Use the following syntax for the wasprofile command:

- ▶ For Windows use **wasprofile -mode -arguments**
- ▶ For Unix use **wasprofile.sh -mode -arguments**

The following modes in Table 4-1 are available:

Table 4-1 wasprofile modes

Mode	Use
-create:	Create a new profile
-augment	Augments the given profile using the given profile template.
-delete	Delete a profile
-unaugment:	Unaugments the profile
-deleteAll	Deletes all registered profiles
-listProfile	List the profiles in the profile registry.
-register	Registers the profile in the registry
-unregister	Removes the profile from the 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.
-help	Lists the valid modes for the wasprofile command.

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

- ▶ Enter `wasprofile -<mode> - help` for detailed help on each mode. See Example 4-2, for an example of the `wasprofile -create -help` command.

Example 4-2 Getting help for the wasprofile command

```
C:\WebSphere\AppServer\bin>wasprofile -create -help
The following command line arguments are required for this mode.
Command line arguments are case sensitive.
-create: Creates a new profile.
-profileName: The name of the profile.
-profilePath: The intended location of the profile in the file system.
-templatePath: The location of the profile template in the file system.
-nodeName: The node name of the profile. The name must be unique within its
cell.
```

- cellName: The cell name of the profile. The cell name must be unique for each profile.
- hostName: The host name of the profile.

C:\WebSphere\AppServer\bin>

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

```
C:\WebSphere\AppServer\bin>wasprofile -listProfiles  
[Dmgr01, AppSrv01, Custom01, Custom02, Dmgr02]
```

4.4.2 Creating a profile

You can use the wasprofile command to create profiles instead of using the Profile creation wizard.

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 wasprofile, you will need to specify one of the following templates:

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

For example, Example 4-3 shows the commands used to create an application server named *saserver1* on node *sanode1* in cell *sacell1* on host *carlavm2.itso.ibm.com*® from the command line.

Example 4-3 Creating a profile with the wasprofile command

```
cd websphere\appserver\bin
```

```
wasprofile -create -profileName saserver1 -profilePath  
c:\websphere\appserver\profiles\saserver1 -templatePath  
c:\websphere\appserver\profileTemplates\default -nodeName sanode1 -cellName  
sacell1 -hostName carlavm2.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_creation_tool> -options <response_file> -silent
```

In this example, `<profile_creation_tool>` is the command required to start the Profile creation wizard. The command to start the wizard is platform-specific and is located in `<was_home>/bin/ProfileCreator`. Choose your platform command from Table 4-2.

Table 4-2 Platform-specific creation wizard

Platform	Profile creation wizard command
AIX	pctAIX.bin
HP UX	pctHPUX.bin
HP UX 64-bit	pctHPUXIA64.bin
Linux	pctLinux.bin
Linux 64-bit	pct.bin
Linux Power	pctLinuxPPC.bin
S/390®	pctLinux390.bin
Solaris	pctSolaris.bin
Windows	pctWindows.exe
Windows 64-bit platforms:	pctWindowsIA64.exe

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

4.5 Managing the processes

In a standalone 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.

4.5.1 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 4-4.

Example 4-4 Starting the deployment manager from the command line

```
C:\WebSphere\AppServer\profiles\dmgr01\bin>startmanager
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 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 Figure 4-36. If there are any errors, check the log file for the node agent process by typing this command:

<profile_home>/logs/nodeagent/SystemOut.log

Figure 4-36 Starting and stopping the node agent from the command line

- 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 155, for each and every node associated with this deployment manager.

4.5.2 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 Figure 4-36 on page 155.
 - 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 155 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 Server dmgr stop completed, as shown in Figure 4-36 on page 155.

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.

4.5.3 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 section which discusses your operating system, Windows or “UNIX and Linux” on page 160.

Windows

The administrator can choose to register one or more of the WebSphere Application Server processes on a machine as a Windows service 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:

Example 4-5 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 panel as:

IBM WebSphere Application Server V6 - <service name>

The convention used by the Profile creation wizard 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 4-6 shows using the **WASService** command to add the deployment manager as a Windows service and sample successful output.

Example 4-6 Registering a deployment manager as a Windows service

```
C:\WebSphere\AppServer\bin>WASService -add "Deployment Mgr" -serverName dmgr  
-profilePath "C:\WebSphere\AppServer\profiles\DMGR01" -restart true  
  
Adding Service: Deployment Mgr  
Config Root: C:\WebSphere\AppServer\profiles\DMGR01 -restart true\config  
Server Name: dmgr  
Profile Path: C:\WebSphere\AppServer\profiles\DMGR01 -restart true  
Was Home: C:\WebSphere\AppServer\  
Start Args:  
Restart: 1  
IBM WebSphere Application Server V6 - Deployment Mgr service successfully  
added.
```

Note that the service name added in Figure 4-37 on page 159 will be IBM WebSphere Application Server V6 - concatenated with the name you specified for the service name.

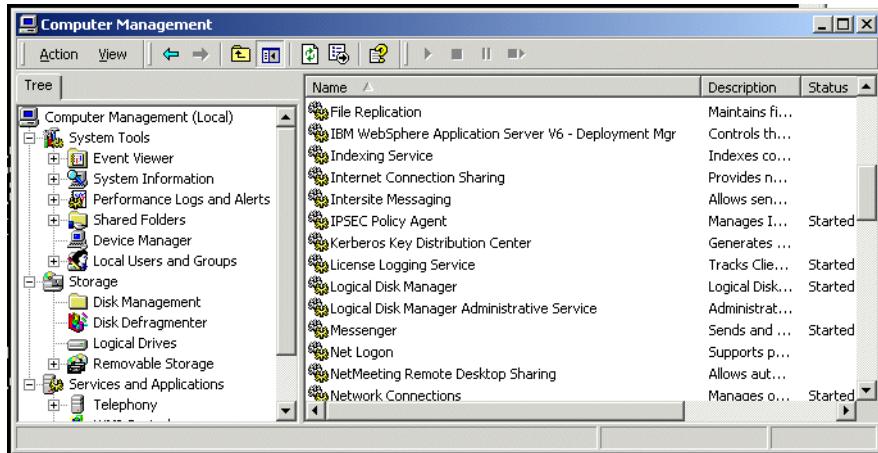


Figure 4-37 New service

If you remove the service using the **WASService -remove** command, specify only the latter portion of the name, as in Example 4-7.

Example 4-7 Removing a service

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

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

The commands shown in Example 4-8 are used on a WebSphere Application Server node to add the node agent and a server as Windows services.

Example 4-8 Registering WebSphere processes as Windows services

```
C:\WebSphere\AppServer\bin>WASService -add CustomNode -serverName nodeagent
-profilePath "C:\WebSphere\AppServer\profiles\CUSTOM01 -restart true
```

```
Adding Service: CustomNode
Config Root: C:\WebSphere\AppServer\profiles\CUSTOM01 -restart true\config
Server Name: nodeagent
Profile Path: C:\WebSphere\AppServer\profiles\CUSTOM01 -restart true
Was Home: C:\WebSphere\AppServer\
Start Args:
Restart: 1
IBM WebSphere Application Server V6 - CustomNode service successfully added.
```

```
C:\WebSphere\AppServer\bin>WASService -add "Cserver1" -serverName Cserver1
-profilePath "C:\WebSphere\AppServer\profiles\CUSTOM01 -restart true
```

```
dding Service: Cserver1
Config Root: C:\WebSphere\AppServer\profiles\CUSTOM01 -restart true\config
Server Name: Cserver1
Profile Path: C:\WebSphere\AppServer\profiles\CUSTOM01 -restart true
Was Home: C:\WebSphere\AppServer\
Start Args:
Restart: 1
BM WebSphere Application Server V6 - 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 processes on a machine, as shown in Example 4-9. Each such process will then be automatically restarted if it has failed.

Example 4-9 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.



Administration basics

In this chapter, we introduce the WebSphere administrative console, command line administration, and some basic administration tasks.

The topics we cover include:

- ▶ 5.1, “Introducing the WebSphere administrative console” on page 162
- ▶ 5.2, “Securing the administrative console” on page 182
- ▶ 5.3, “Working with the deployment manager” on page 183
- ▶ 5.4, “Working with application servers” on page 190
- ▶ 5.5, “Working with nodes” on page 217
- ▶ 5.6, “Working with clusters” on page 235
- ▶ 5.7, “Working with virtual hosts” on page 239
- ▶ 5.8, “Managing applications” on page 242
- ▶ 5.9, “Managing your configuration files” on page 258

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 publications:

- ▶ *WebSphere Application Server V6: High Availability Solutions*, REDP-3971
- ▶ *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392

5.1 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, applications, viewing product messages, and so on. The configuration data is stored as a set of XML documents arranged in a set of cascading directories under `<profile_home>/config` directory for each profile.

In a single server installation, 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 multi-server environment with Network Deployment, the administrative console is located in the deployment manager server, dmgr. In this case, the administrative console provides centralized administration of multiple nodes, and nodes on multiple machines. Configuration changes are made to the master repository XML configuration files and pushed to the local XML 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 in the nodes where the WebSphere Application Server instances are installed.

In WebSphere Application Server V6, the administrative console groups administrative tasks into the following categories:

- ▶ Servers
- ▶ Applications
- ▶ Resources
- ▶ Security
- ▶ Environment
- ▶ System administration
- ▶ Monitoring and tuning
- ▶ Troubleshooting
- ▶ Service integration
- ▶ UDDI

5.1.1 Starting the administrative console

The way you access the administrative console is the same whether you have a standalone server environment or a distributed server environment. However, the location and how you start the necessary processes will vary.

Standalone server environment

In a single application server installation, the console is hosted on the application server so you must start it 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: `<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: `<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/admin`
 - `https://<hostname>:9043/admin`

`<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: `<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: `<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.

5.1.2 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 global security is enabled.

- ▶ WebSphere global security is not enabled.

If global 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 global security is enabled.

If global security is enabled, you must enter a valid user ID and password.

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. If the user closed a Web browser during a session and did not logout first or if the session timed out, for example.

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. 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.

5.1.3 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. Edit the `<was_home>/systemApps/adminconsole.ear/deployment.xml` file in a text editor.
 - a. Locate the xml statement, as shown in Example 5-1:

Example 5-1 InvalidationTimeout statement

```
<tuningParams xmi:id="TuningParams_1088453565469"
maxInMemorySessionCount="1000" allowOverflow="true"
writeFrequency="TIME_BASED_WRITE" writeInterval="10"
writeContents="ONLY_UPDATED_ATTRIBUTES" invalidationTimeout="30">
```

- b. Change the invalidationTimeout value to the desired session timeout and save the file
2. Restart the console.

5.1.4 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 5-1.

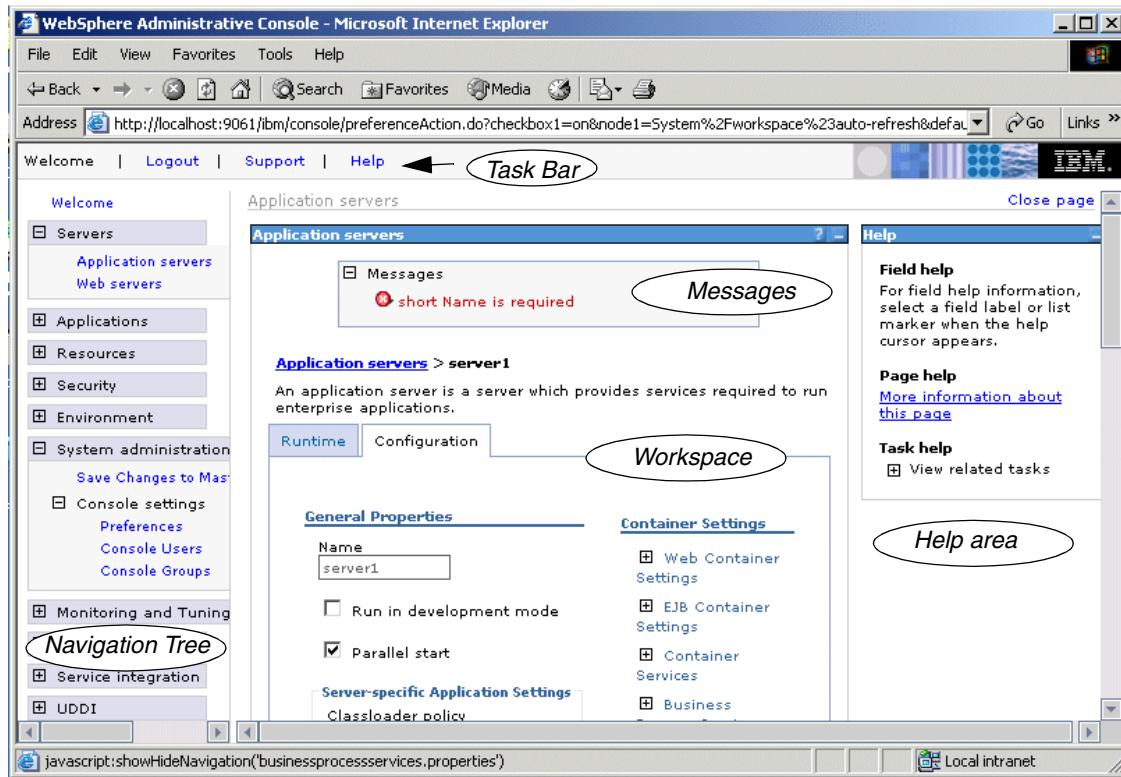


Figure 5-1 The administrative console graphical interface

Taskbar

The taskbar is the horizontal bar near the top of the console. The task bar provides the following actions:

- ▶ **Welcome** displays the administrative console home page. It contains links to information sources.
- ▶ **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.

- ▶ **Support** takes you to a page with links to support sites. It also contains a link to download the IBM Support Assistant.
- ▶ **Help** opens a new Web browser with detailed online help for the administrative console. This is not part of the Information Center.

The task bar display is controlled with the **Show banner** setting in the console preferences. See “Setting console preferences” on page 168.

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 following folders are provided for selection:

Table 5-1 Navigation tree options

Navigation tree option	Description	Standalone	Deployment Manager
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
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

Navigation tree option	Description	Standalone	Deployment Manager
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 runtime 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 5-1 on page 166, 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 runtime 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 5-2 on page 178.

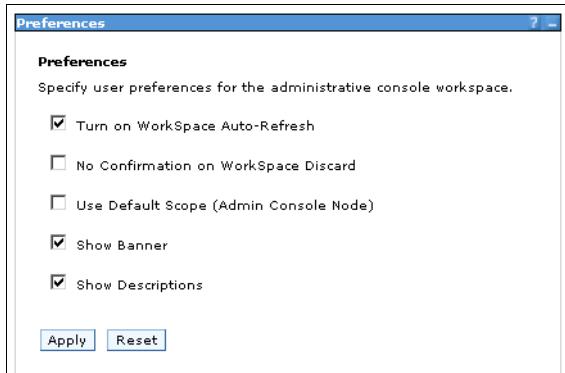


Figure 5-2 Administrative console preferences

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 logout 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 (Admin console node)** sets the default scope to the node of the administration console.
- ▶ **Show banner** displays the task bar at the top of the console.
- ▶ **Show descriptions** displays help information in the right-hand portion of the workspace.

Click the boxes to select which preferences you want to enable and click **Apply**.

5.1.5 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. Set the scope to define which processes have access to the resource.

- 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 a WebSphere administrative cell. To create a JDBC provider, for example, expand **Resources** and then select the **JDBC Providers** action. See Figure 5-3.

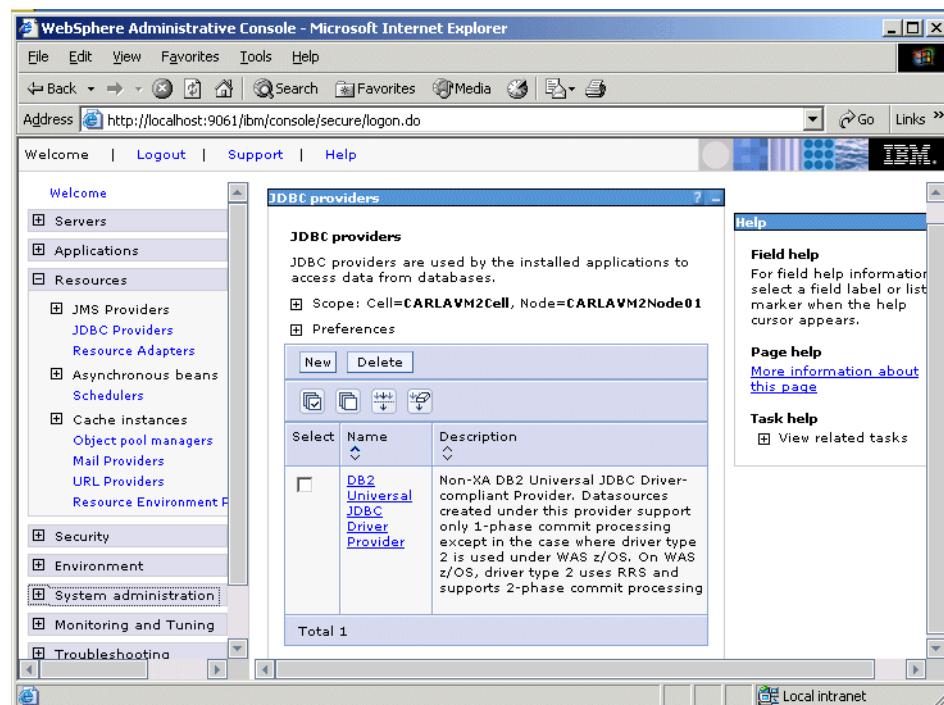


Figure 5-3 Working with the administrative console

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 5-4.

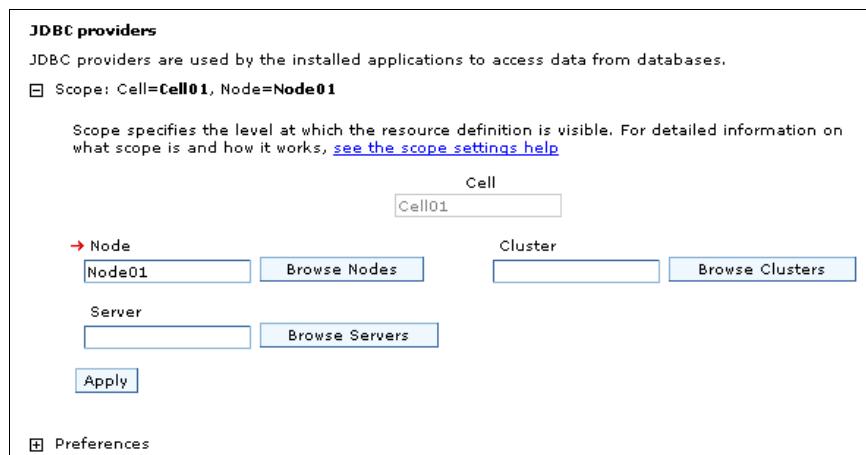


Figure 5-4 Setting scope

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. The application scope is not an option on the administration console. Resources and variables are scoped at the application level by defining them in an enhanced EAR.
2. Resources scoped at the server level apply only to that server. If a server 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. If a node is specified but the server field is empty, the scope is set to that 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 don't 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. If the node and server fields are blank, the scope is set to the cell level.

Click **Apply** to set the scope.

The scope setting is available for all resource types, WebSphere variables, shared libraries, and name space bindings.

Standalone 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. You will not have the option to fill in a name for the cell, server, or node. And you will not see the cluster scope as an option.

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 5-5 on page 173.

The screenshot shows the 'Application servers' configuration page. At the top, there's a section for 'Preferences' with options like 'Maximum rows' set to 20, and checkboxes for 'Retain filter criteria', 'Show confirmation for stop command', 'Show confirmation for immediate stop command', and 'Show confirmation for terminate command'. Below this are 'Apply' and 'Reset' buttons.

Annotations on the left side of the screenshot include:

- A label 'clear the filter' with an arrow pointing to the 'Delete' button in the toolbar above the table.
- A label 'set a filter' with an arrow pointing to the 'Filter' dropdown menu in the toolbar above the table.

The main area contains a table with columns: Select, Name, Node, Version, and Status. The table shows three rows of server data:

Select	Name	Node	Version	Status
<input type="checkbox"/>	Name	Node01	6.0.0.0	
<input type="checkbox"/>	Cserver2	Node01	6.0.0.0	
<input type="checkbox"/>	ServerN11	Node01	6.0.0.0	

Figure 5-5 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 5-5, 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, when we activate the filter, we will only see servers with names which start with **Cs**.
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 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.

5.1.6 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 5-6 on page 175.

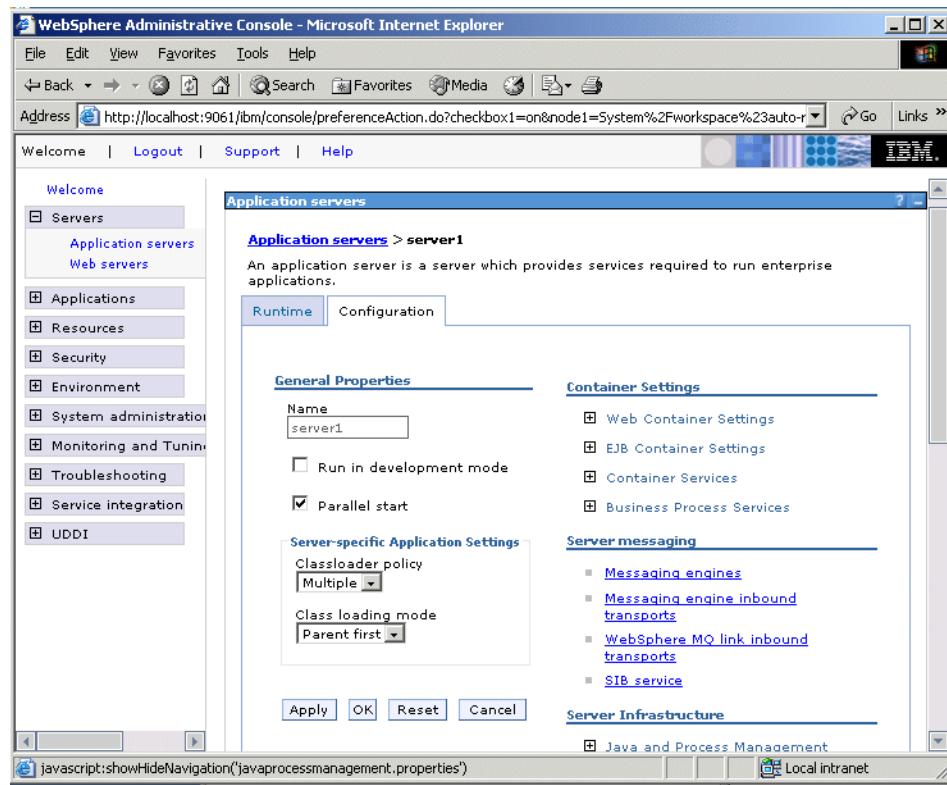


Figure 5-6 Editing application server properties

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 5-7.

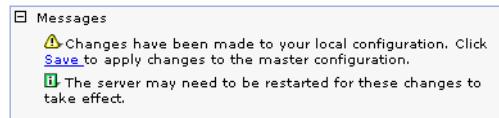


Figure 5-7 Save changes to the master repository

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.

5.1.7 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 5-8.
2. Select **Scope** and click **Apply** to set it, if applicable.
3. Click the **New** button above the collection table in the workspace.

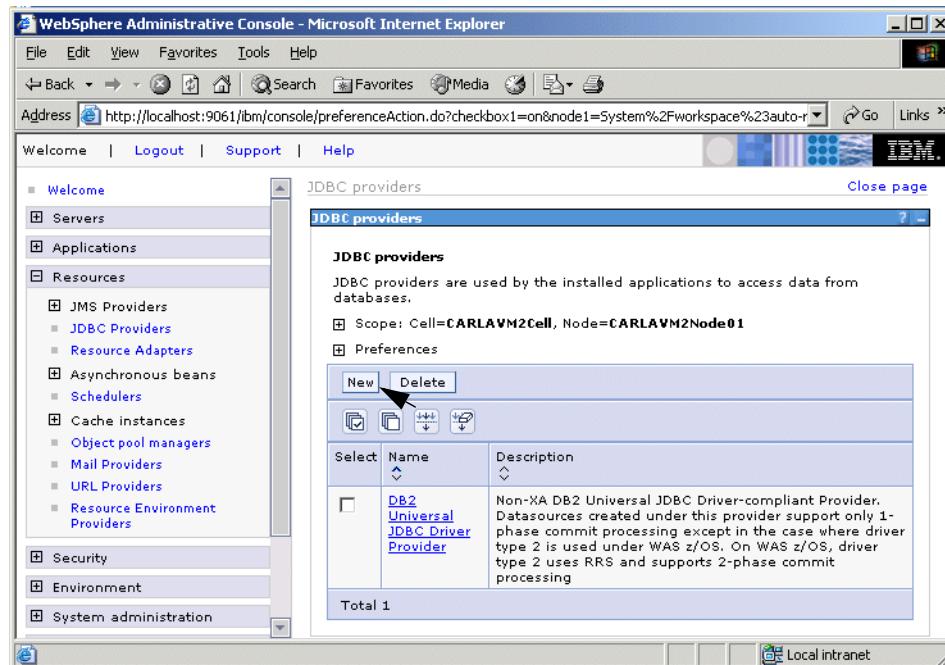


Figure 5-8 Create a new item

You might be presented with one or more configuration pages in which you have to specify the item properties. If so, fill in the information and click **Apply**. This can in turn, generate additional links for advanced configuration properties. Proceed until all the required properties are entered.

Alternatively, a wizard might start, prompting you to enter information and taking you through the process.

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.

5.1.8 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 task bar 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**.

5.1.9 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 5-9 on page 178.

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**.

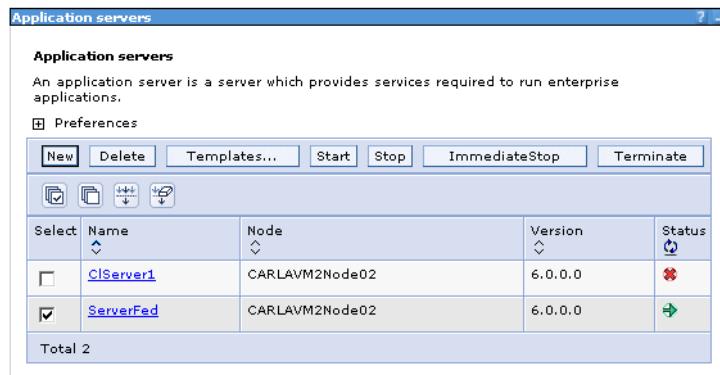


Figure 5-9 Starting and stopping items

Table 5-2 shows how to start and stop various items from the administrative console.

Table 5-2 How to stop and start items

Items	Admin console	Menu selections
Applications	Standalone & distributed server environment	Applications →Enterprise Applications
Application servers	Distributed server environment	Servers →Application Servers
Clusters ¹	Distributed server environment	Servers →Clusters
Web servers	Standalone & distributed server environment	Servers →Web servers
Generic servers	Distributed server environment	Servers →Generic servers
Nodes ³	Distributed server environment	System administration →Nodes

Items	Admin console	Menu selections
Node agents ⁴	Distributed server environment	System administration →Node agents
Deployment manager ^{2,3}	Distributed server environment	System administration →Deployment Manager

¹ Starting or stopping a cluster starts or stops the application servers in the cluster.

² Stopping the deployment manager also stops your administrative console session. It does not stop any of the node agents or the application servers running under those node agents.

³ This item can only be stopped from the administrative console, not started.

⁴ This item can be stopped and recycled, but cannot be started if it is stopped from the administrative console.

5.1.10 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 5-10.

The screenshot shows the 'WebSphere Variables' interface. At the top, there's a toolbar with buttons for 'New' and 'Delete'. Below the toolbar is a table with columns for 'Select', 'Name', 'Value', and 'Scope'. Three variables are listed:

Select	Name	Value	Scope
<input type="checkbox"/>	APP_INSTALL_ROOT	<code>\$(USER_INSTALL_ROOT)/installedApps</code>	cells:CARLAVM2Cell01:nodes
<input type="checkbox"/>	CLOUDSCAPE_JDBC_DRIVER_PATH	<code>\$(WAS_INSTALL_ROOT)/cloudscape/lib</code>	cells:CARLAVM2Cell01:nodes
<input type="checkbox"/>	CONNECTJDBC_JDBC_DRIVER_PATH		cells:CARLAVM2Cell01:nodes

Figure 5-10 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 5-11.

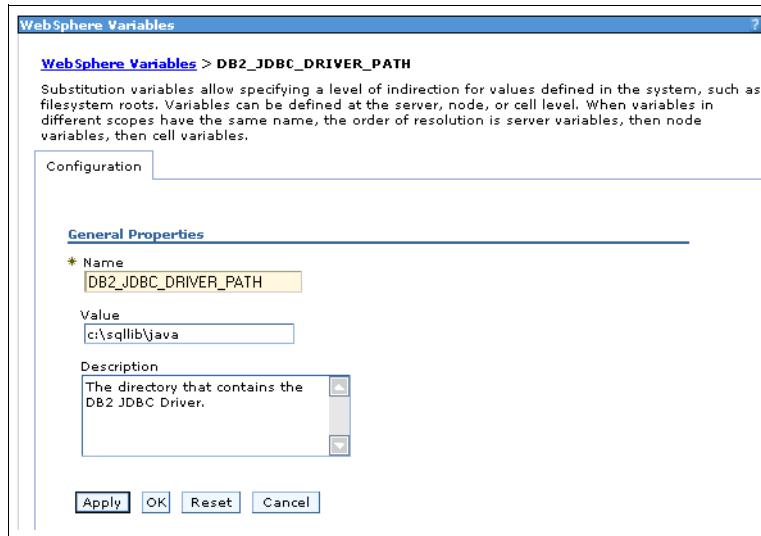


Figure 5-11 New WebSphere variable

5.1.11 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 changes by expanding **View items with changes** 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.

5.1.12 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 5-12 on page 182.

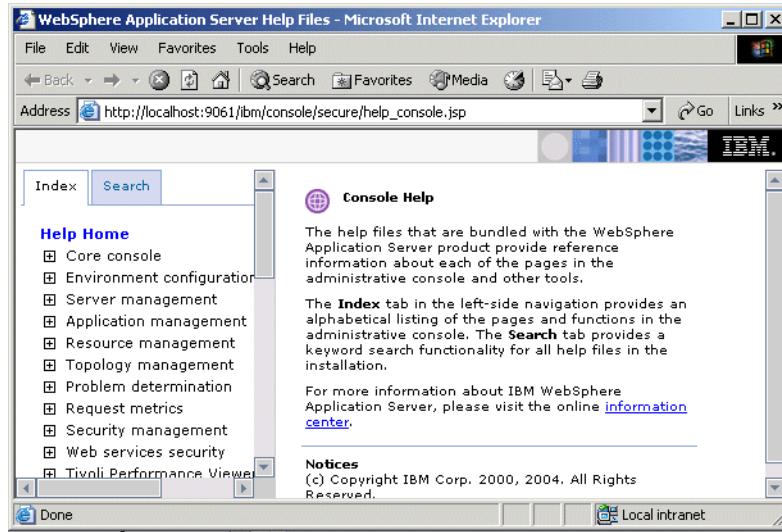


Figure 5-12 Online help

2. Enable the **Show Descriptions** option in the console preferences. If this option is enabled, you can minimize the Help window in the workspace, if you like.
3. The Information Center can be viewed online or downloaded from:

<http://www.ibm.com/software/webservers/appserv/infocenter.html>

5.2 Securing the administrative console

WebSphere Application Server provides the ability to secure the administrative console so only authenticated users can use it. In order to take advantage of this feature, you need to first activate WebSphere global security. Enabling security is an important step in ensuring a safe WebSphere environment. However, the considerations and decisions involved in achieving a secure environment are outside the scope of this book. To understand your security options and for help designing a secure system, refer to the *WebSphere Application Security V6 Security Handbook*, SG24-6316.

This section assumes that you have enabled WebSphere global security and therefore concentrates on the steps needed to secure the console.

Console 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 roles determine the administrative actions the user can perform.

Users and groups are added and roles assigned to them by selecting **System Administration** → **Console Users** or **System Administration** → **Console Groups**.

You can choose the following roles for each user. The roles are listed from most restrictive to most privileges:

- ▶ **Monitor** allows a user to view the WebSphere configuration and current state.
- ▶ **Configurator** has Monitor privilege plus the ability to change the WebSphere configuration.
- ▶ **Operator** incorporates Monitor privilege plus the ability to change the runtime state, such as starting and stopping services.
- ▶ **Administrator** incorporates Operator plus Configurator properties.

Be sure to save your work. See 5.1.11, “Saving work” on page 180. After saving the configuration, you must restart the application server in a standalone 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.

5.3 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.

5.3.1 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 5-13 on page 184 shows the Configuration page.

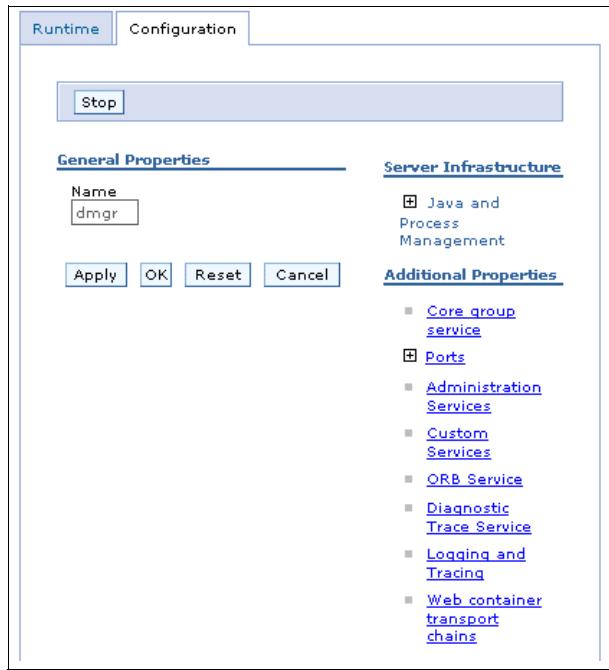


Figure 5-13 Deployment manager configuration

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 V6: High Availability Solutions*, REDP-3971.

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 are used to enable auditing.
- ▶ Existing JMX connectors include RMI and SOAP. This allows you to update, add HTTP and JMS connectors, or remove connectors.
- ▶ Mbean extensions you can add in order to manage new types of resources.
- ▶ Custom properties consist of name/value pairs.
- ▶ Web server plug-in is automated.

Custom Services

Custom services settings provide an extension point for configuration data for plug-in services. This allows you to add in custom code which will be executed during process initialization.

ORB Service

These settings allow you to specify settings for the Object Request Broker service.

Diagnostic Trace Service

These settings allow you to configure the diagnostic trace service. You can change these from two places :

- ▶ On the configuration panel, requiring a server restart to activate

- ▶ On the runtime panel to activate them immediately

The settings include whether to put the trace information in memory to file, and the trace output format. For more information about the diagnostic trace service settings, see 9.4.1, “Diagnostic trace service” on page 431.

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 “Web container transport chains” on page 213.

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 5-14 on page 187 shows the Runtime tab.

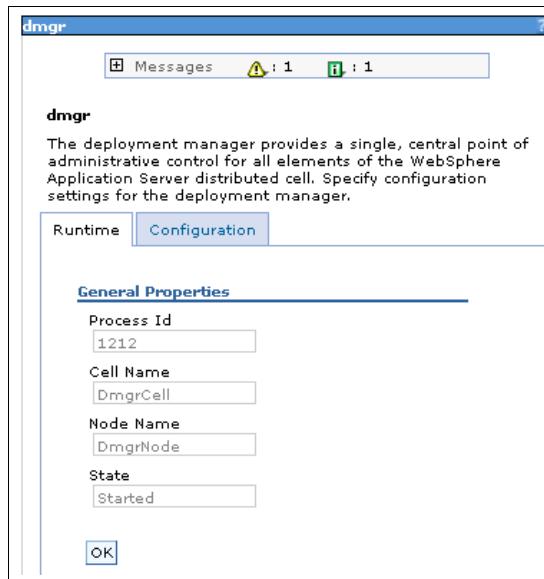


Figure 5-14 Deployment manager runtime page

Perhaps the most useful item on this page is the process ID. If you have multiple deployment managers running, you could use this page to determine the administrative console to which you are connected. The fact that the State is Started does not mean much, because you would not be able to access the administrative console otherwise.

5.3.2 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 4.5.3, “Enabling process restart on failure” on page 157).

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 → Application Server Network Deployment V6 → Profiles → <profile_name> → Start the deployment manager**

- ▶ Start → Programs → IBM WebSphere → Application Server Network Deployment V6 → 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, shown in Example 5-2.

Example 5-2 startManager command

```
c:\>cd websphere\appserver\profiles\dmgr01\bin
C:\WebSphere\AppServer\profiles\dmgr01\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 5-3.

Table 5-3 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</i> >/logs/dmgr/startServer.log

Option	Description
-profileName <profile>	Specify a 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.
-trace	Generates trace information into a file for debugging purposes. The output goes to startServer.log.
-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 start_dmgr.bat(sh). The script is saved to the <dmgr_profile_home>/bin directory. The -background 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: -D<name>=<value>
-help or -?	Prints the command syntax to the console.

Stopping the deployment manager

The deployment manager is stopped with the stopManager command, as shown in Example 5-3.

Example 5-3 stopManager command

```
c:\>cd websphere\appserver\profiles\dmgr01\bin
C:\WebSphere\AppServer\profiles\dmgr01\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 5-4.

Table 5-4 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.

5.4 Working with application servers

This section discusses the following topics:

- ▶ 5.4.1, “Creating an application server” on page 191
- ▶ 5.4.2, “Viewing the status of an application server” on page 194
- ▶ 5.4.3, “Starting an application server” on page 197

- ▶ 5.4.4, “Stopping an application server” on page 200
- ▶ 5.4.5, “Viewing runtime attributes of an application server” on page 203
- ▶ 5.4.6, “Customizing application servers” on page 206

Server types: This section uses the following terms.

- ▶ A *standalone 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 standalone 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.

5.4.1 Creating an application server

The process to create an application server depends on your WebSphere Application Server package.

Standalone application servers

Standalone application servers are created by creating an application server profile. This results in a profile that defines one standalone 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 4.3.2, “Creating an application server profile” on page 130.

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 5.6, “Working with clusters” on page 235.

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 5-15.
4. Select the node for the new server and enter a name for the new server.

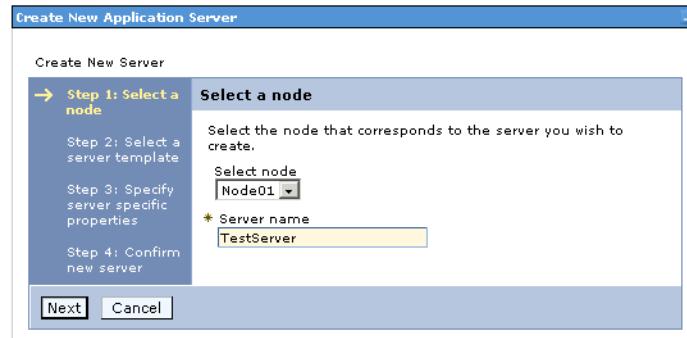


Figure 5-15 Create an application server: Step 1

Click **Next**.

5. Select a template to use by clicking the appropriate radio button. See Figure 5-16. You automatically have a default template to select. Later, you can create templates based on existing application servers. (see “Creating a template” on page 194).

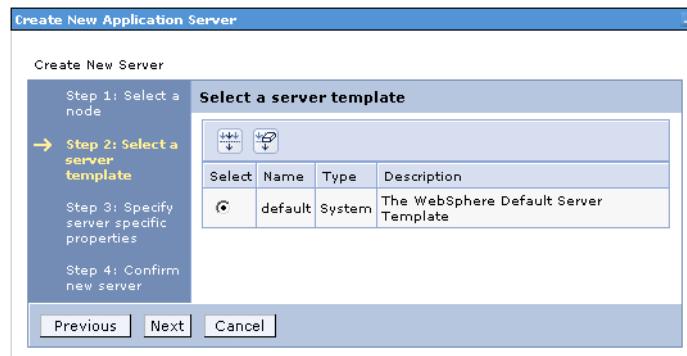


Figure 5-16 Create an application server: Step 2

Click **Next**.

6. See Figure 5-17. Select the core group from the list and check the **Generate Unique Http Ports** box to have unique ports generated for this server.

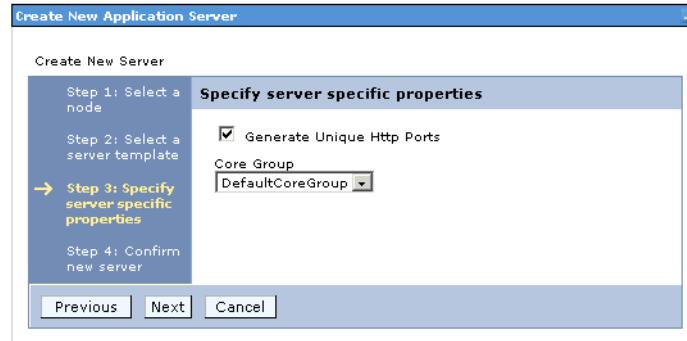


Figure 5-17 Create an application server: Step 3

Click **Next**.

7. See Figure 5-18. A summary panel is presented with the options you chose.

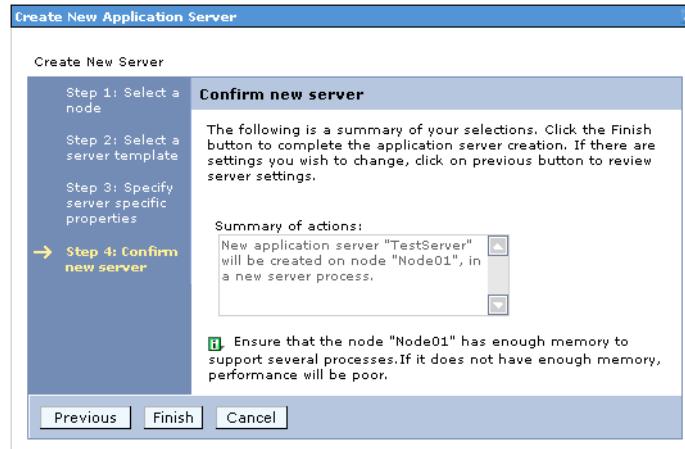


Figure 5-18 Create an application server: Step 4

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 4.5.3, “Enabling process restart on failure” on page 157).

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.

5.4.2 Viewing the status of an application server

Table 5-5 shows a summary of ways to view the status of an application server.

Table 5-5 Methods to view the status of an application server

Method	Server types	Summary
Windows service	Managed and standalone	If an application server is registered as a Windows service, then check the Windows services panel for its status.
Command line	Managed and standalone	<p>To view the status of a standalone application server, type:</p> <pre>cd <profile_home>/bin serverStatus(.sh) server1</pre> <p>To view the status of a managed application server, type:</p> <pre>cd <profile_home>/bin serverStatus(.sh) <server_name></pre> <p>To check the status of all servers on the node, type:</p> <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**.
2. The servers are listed. The last column to the right contains an icon to indicate the status of each server. Figure 5-19 shows the icons and the corresponding status.



Figure 5-19 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 5-6 on page 196 for a list of available options.

Table 5-6 Options for serverStatus

Option	Description
-logfile <log file path>	Specify alternative location for 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 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 5-4 shows an example of using the serverStatus command.

Example 5-4 serverStatus example

```
C:\WebSphere\AppServer\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.

5.4.3 Starting an application server

How you start an application server depends largely on personal preference and on whether the application server is standalone 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 5-7 shows the various methods you can use to start an application server.

Table 5-7 Methods to start an application server

Method	Server types:	Summary
Windows service	Managed and standalone	Application servers can be registered as a Windows service. You can start the server by starting the service.
First steps menu	Managed and standalone	The First Steps menu is located at <profile_home>/firststeps/firststeps.bat (.sh)
Windows Start menu	Managed and standalone	Start → Programs → IBM WebSphere → Application Server V6 → Profiles → <profile_name> → Start the Server
Command line	Managed and standalone	cd <profile_home>/bin startServer(.sh) server1
Administrative console	Managed	Servers → Application Servers To start a managed server from the administrative console, the node agent must be started.
Administrative console	Clusters	Servers → Clusters Starting a cluster starts each application server in the cluster.

Using the administrative console to start a managed server

Note: Before managing a server in a Network Deployment 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 from the command line of the node using the following command:

```
<profile_home>/bin/startNode (.sh)
```

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 5.8.7, “Preventing an enterprise application from starting on a server” on page 247.

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 5-8.

Table 5-8 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.

Option	Description
-logfile <log file path>	Specify 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>]	<p>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>.</p> <p>The script needs to be saved to the bin directory of the node installation.</p>
-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.
-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 5-5 on page 200 shows an example of using the startServer command.

Example 5-5 startServer example

```
C:\WebSphere\AppServer\profiles\AppSrv02\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
```

5.4.4 Stopping an application server

How you stop an application server depends largely on personal preference and on whether the application server is standalone or managed. Keep in mind that the application server created by a application server profile is always called server1.

Table 5-9 Methods to stop an application server

Method	Server types:	Summary
Windows service	Managed and standalone	Application servers can be registered as a Windows service. You can stop the server by stopping the service.
First steps menu	Managed and standalone	The First Steps menu is located at <profile_home>/firststeps/firststeps.bat (.sh)
Windows Start menu	Managed and standalone	For a standalone application server, do the following: Start → Programs → IBM WebSphere → Application Server V6 → Profiles → <profile_name> → Stop the Server For a standalone or managed application server on a Network Deployment system, do the following: Start → Programs → IBM WebSphere → Application Server Network Deployment V6 → Profiles → <profile_name> → Stop the Server

Method	Server types:	Summary
Command line	Managed and standalone	<p>For a standalone application server:</p> <pre>cd <profile_home>/bin stopServer(.sh) server1</pre> <p>For a managed application server :</p> <pre>cd <profile_home>/bin stopServer(.sh) <server_name></pre>
Administrative console	Managed	<p>Servers → Application Servers.</p> <p>To stop a managed server from the administrative console, the node agent must be started.</p>
Administrative console	Managed	<p>System Administration → Node Agents → Restart all Servers on the Node.</p> <p>This restarts all the servers on the node.</p>

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, 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, 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 5-10.

Table 5-10 stopServer command options

Option	Description
-nowait	Tells the command not to wait for 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. Output is to stopServer.log.
-logfile <log file path>	Specify alternative location for 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 output to the existing file.

Option	Description
-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 to use 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 5-6 shows an example of the stopServer command

Example 5-6 stopServer command example

```
C:\WebSphere\AppServer\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.
```

5.4.5 Viewing runtime attributes of an application server

To view runtime 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 5-20 on page 204 shows the Runtime tab and the information it provides.

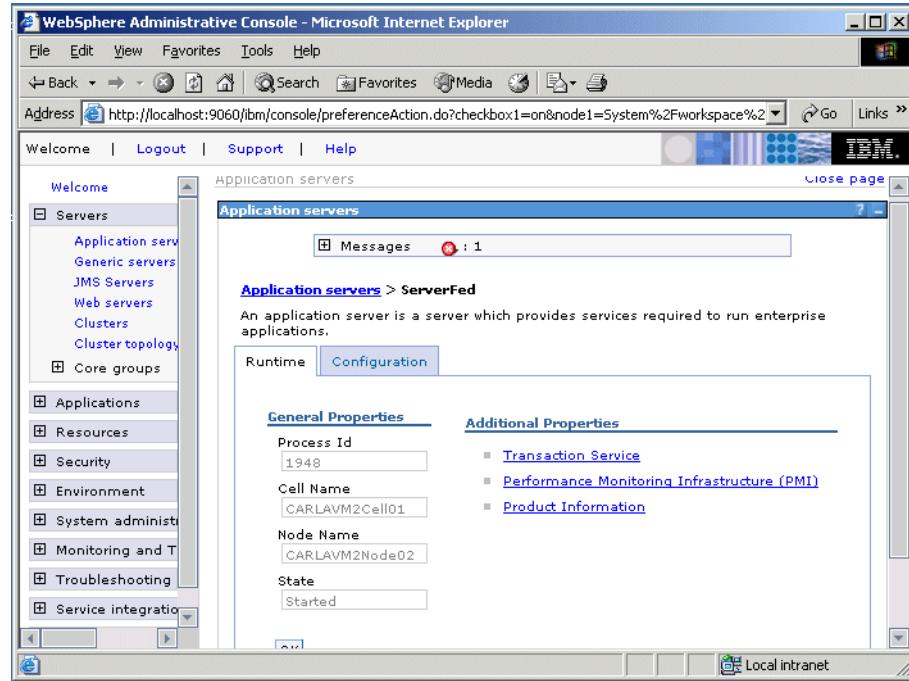


Figure 5-20 Application server Runtime tab

4. From the Runtime tab, you have access to the following:

- **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.

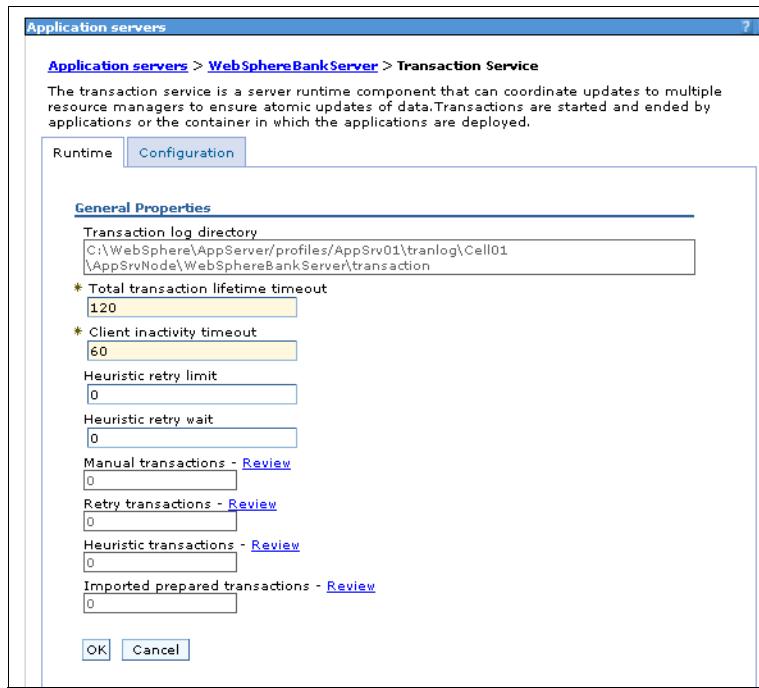


Figure 5-21 Transaction service options and settings

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/retry, 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.

5.4.6 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 5-22 on page 207.

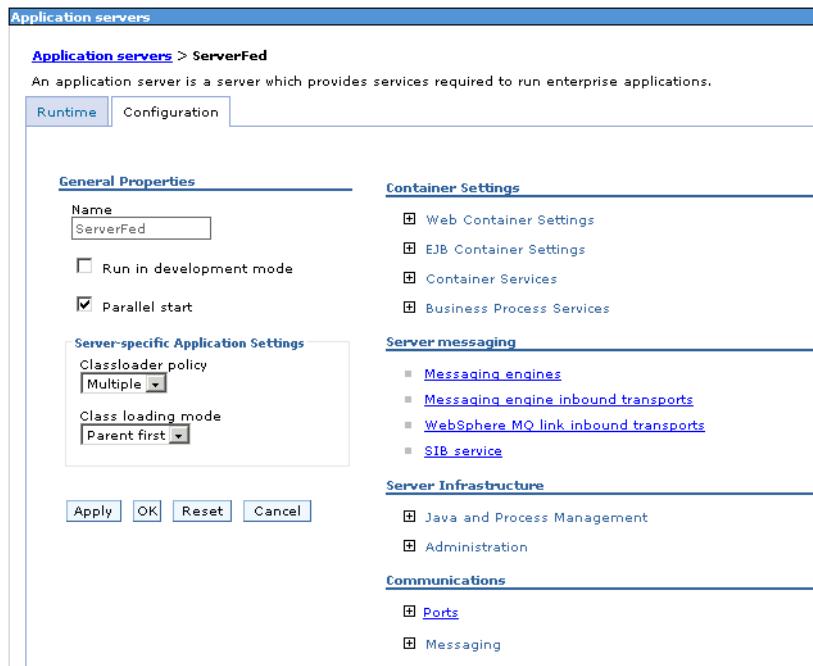


Figure 5-22 Application server configuration

General properties

The general properties consist of a few items which 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.

- ▶ **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 14, “Understanding class loaders” on page 821.

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 5-23.

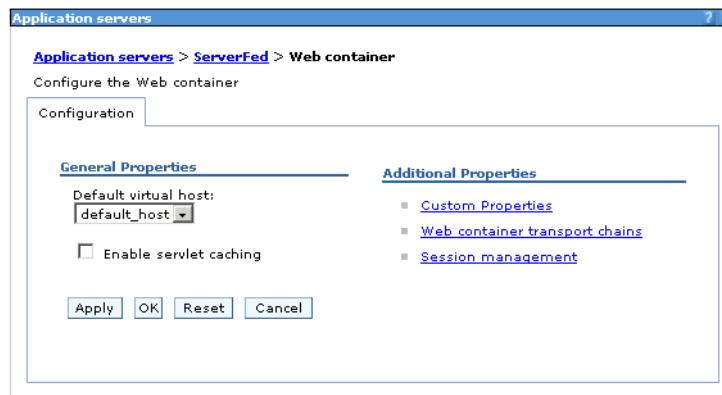


Figure 5-23 Web container settings

- ▶ **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.
- ▶ **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 12, “Session management” on page 697.
- ▶ **Web container transport chains:** You can add to or configure the communication channels used for accessing applications in the Web container. By default, you have four transport chains predefined. These are for secure and nonsecure administration console access, and for default access to the Web container.

The transport chains are related to port definitions seen communications section. Port numbers must be unique for each application server instance on a given machine. For more information about transport chains, see “Web container transport chains” on page 213.
- ▶ **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.

EJB container properties

These properties allow you configure the services provided by the EJB container.

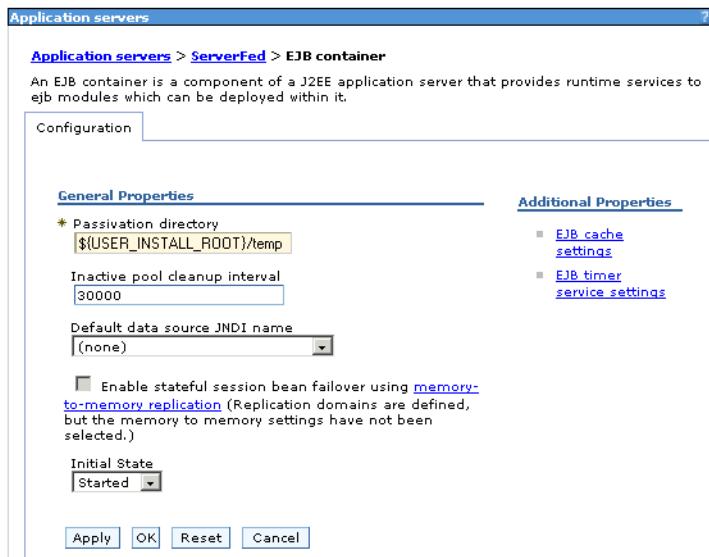


Figure 5-24 EJB container settings

- ▶ **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.

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 runtime.
- ▶ **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 10, “Asynchronous messaging” on page 463 and Chapter 11, “Default messaging provider” on page 593.

Server infrastructure

The server infrastructure settings include settings for Java and process management and administration services.

- ▶ **Class loader:** You can define new class loaders. Class loaders are discussed in Chapter 14, “Understanding class loaders” on page 821.
- ▶ **Process definition:** You can enhance the operation of an application server, 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 **Java virtual machine** definitions, such as the initial and maximum heap sizes, debug options, the process classpath, or different runtime options such as profiler support and heap size.

- ▶ **Environment entries:** These entries are a set of name/value pairs for use in the application server. For example, to set DB2 required environment variables.
- ▶ **Process execution settings:** 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 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.

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 “Web container transport chains” on page 213), the service integration bus ports (see 11.2.2, “Service integration bus transport chains” on page 614), 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 Security Handbook*, SG24-6316.

Troubleshooting

These settings include those for logging and tracing. For information troubleshooting and using these settings, see Chapter 9, “Problem determination” on page 417.

Additional properties

The following settings are defined under the additional properties section:

- ▶ Core group service and core group bridge service
 - These settings are related to high availability.
- ▶ 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 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 5-11.

Table 5-11 Web container transports

Port	Transport chain	Transport channels
WC_adminhost	WCInboundAdmin <ul style="list-style-type: none"> ▶ Enabled ▶ Host = * ▶ Port = 9061 ▶ SSL disabled 	TCP Inbound Channel (TCP 1) <ul style="list-style-type: none"> ▶ Host = * ▶ Port = 9061 ▶ Thread pool=Web container ▶ Max open connections = 100 ▶ Inactivity timeout = 60 sec HTTP Inbound Channel (HTTP 1) <ul style="list-style-type: none"> ▶ Keepalive enabled ▶ Max persistent requests = 100 ▶ Read timeout = 60 sec ▶ Write timeout = 60 sec ▶ Persistent timeout = 30 sec Web Container Inbound Channel (WCC 1) <ul style="list-style-type: none"> ▶ Discrimination weight = 1 ▶ Write buffer size = 32768

Port	Transport chain	Transport channels
WC_adminhost_secure	WCInboundAdminSecure <ul style="list-style-type: none"> ▶ Enabled ▶ Host = * ▶ Port = 9044 ▶ SSL enabled 	TCP Inbound Channel (TCP 1) <ul style="list-style-type: none"> ▶ Host = * ▶ Port = 9044 ▶ Thread pool=Web container ▶ Max open connections = 100 ▶ Inactivity timeout = 60 sec SSL Inbound Channel (SSL 1) <ul style="list-style-type: none"> ▶ SSL repertoire DMGRNode/DefaultSSLSettings HTTP Inbound Channel (HTTP 3) <ul style="list-style-type: none"> ▶ Keepalive enabled ▶ Max persistent requests = 100 ▶ Read timeout = 60 sec ▶ Write timeout = 60 sec ▶ Persistent timeout = 30 sec Web Container Inbound Channel (WCC 1) <ul style="list-style-type: none"> ▶ Discrimination weight = 1 ▶ Write buffer size = 32768
WC_defaulthost	WCInboundAdminSecure <ul style="list-style-type: none"> ▶ Enabled ▶ Host = * ▶ Port = 9080 ▶ SSL disabled 	TCP Inbound Channel (TCP 2) <ul style="list-style-type: none"> ▶ Host = * ▶ Port = 9080 ▶ Thread pool=Web container ▶ Max open connections = 20000 ▶ Inactivity timeout = 60 sec HTTP Inbound Channel (HTTP 2) <ul style="list-style-type: none"> ▶ Keepalive enabled ▶ Max persistent requests = 100 ▶ Read timeout = 60 sec ▶ Write timeout = 60 sec ▶ Persistent timeout = 30 sec
		Web Container Inbound Channel (WCC 2) <ul style="list-style-type: none"> ▶ 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 server 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.

5.5 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.

5.5.1 Adding 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 that you have not federated yet. 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. If you are adding a custom profile to the cell, you will need to use the `addNode` command.

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 detail 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 8, “Managing Web servers” on page 377. See Figure 5-25 on page 218.
3. Specify the host name of the node to be added to the cell.
4. Fill in the following fields, as applicable:

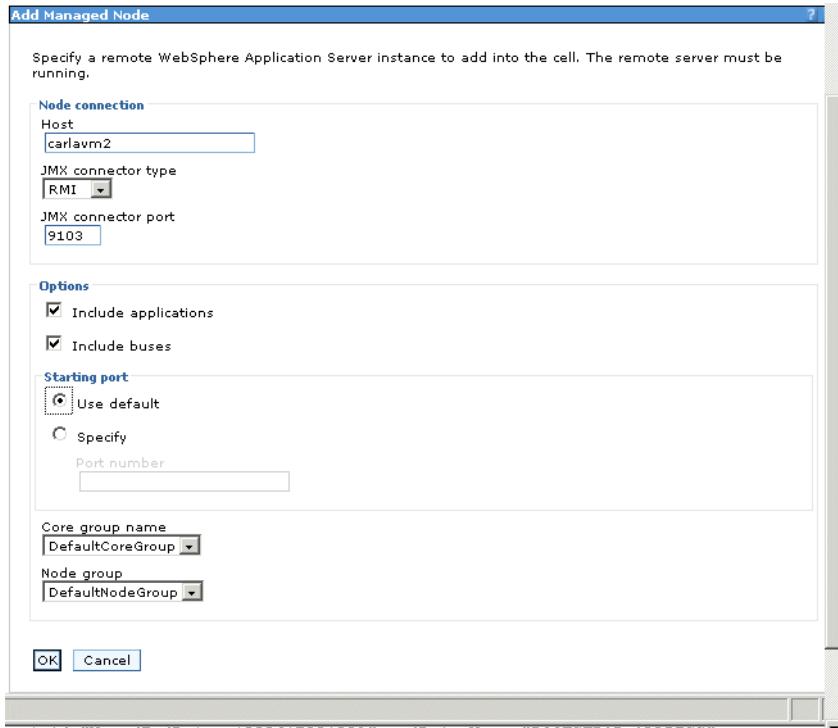


Figure 5-25 Working with nodes

– 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.

– 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, etc.

- CoreGroupName

If you have defined core groups other than the default core group, you can specify the group of which this node will be a member.

- NodeGroupName

If you have defined node groups other than the default node group, you can specify the group of which this node will be a member.

Click **OK**. The messages will be displayed on the administrative console. See Example 5-7.

Example 5-7 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 will can find this port number by looking in the Additional Properties section in the detail 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 standalone application server.
2. Change the directory to the `<profile_home>/bin` directory of the standalone application server installation.
3. Run the **addNode** command.

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 5-12.

Table 5-12 Options for addNode

Option	Description
<code>-nowait</code>	Tell the command not to wait for successful completion of the node addition.
<code>-quiet</code>	Suppress progress information printed to console in normal mode. This option does not affect information written to file.
<code>-trace</code>	Generate trace information into a file for debugging purposes. The output goes to <code>addNode.log</code> .
<code>-logfile <log file path></code>	Specify an alternative location for command's log output, instead of <code>addNode.log</code> . The path can be specified in the following forms: absolute, relative, or file name. The default is <code><profile_home>/logs/addNode.log</code> .
<code>-replacelog</code>	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.
<code>-conntype <type></code>	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 <code><dmgr_port></code> argument.
<code>-profileName <profile></code>	Specify the profile to run the command against. If the command is run from <code><was_home>/bin</code> and <code>-profileName</code> is not specified, the default profile is used. If it is run from <code><profile_home>/bin</code> , that profile is used.
<code>-username <username></code>	Specify a user name for authentication if WebSphere security is enabled. The user name is ignored if WebSphere security is disabled.
<code>-password <password></code>	Specify a password for authentication if WebSphere security is enabled. The password is ignored if WebSphere security is disabled.
<code>-includeapps</code>	Attempt to include the applications in the incorporation of the base node into a cell. Default is not to include the applications.

Option	Description
-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 shortname to use for the new node agent.
-nodegroupname <name>	Specify 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 5-8 shows an example of using the addNode command to add a custom node to a cell.

Example 5-8 addNode usage examples

```
C:\WebSphere\AppServer\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:\WebSphere\AppServer/profiles/Node02/config/cells/CARLAVM2Node02Cell
1/nodes/Node02/servers
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:\WebSphere\AppServer\profiles\Node02\bin>

5.5.2 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, saved 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

`removeNode` detaches a node from a cell and returns it to a standalone configuration.

To use the command, do the following:

1. Change directory to the `<profile_home>/bin` directory of the application server installation for that node.
2. Run **removeNode**. All parameters are optional for this command.

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 5-13 shows the removeNode parameters.

Table 5-13 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.
-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	Print command syntax information.

Example

Table 5-9 shows an example of using the removeNode command.

Example 5-9 removeNode example

```
C:\WebSphere\AppServer\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.
```

5.5.3 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 5-14.

Table 5-14 Options for syncNode

Option	Description
-nowait	Tell the command not to wait for successful synchronization of the node.
-quiet	Suppress progress information printed to 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.

Option	Description
-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 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.
-help or -?	Print a usage statement.

Example

Example 5-10 shows and example of using the syncNode command. This example was run on a Windows system.

Example 5-10 syncNode usage examples

C:\WebSphere\AppServer\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:\WebSphere\AppServer\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
```

5.5.4 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 node system.

startNode command

The command syntax is as follows:

```
startNode [options]
```

The parameters are shown in Table 5-15.

Table 5-15 startNode parameters

Parameter	Description
<code>-nowait</code>	Do not wait for successful initialization of the node agent process.
<code>-quiet</code>	Suppress the printing of progress information.
<code>-logfile <fileName></code>	Specify the location of the log file to which information gets written. The default is <code><profile_home>/logs/nodeagent/startServer.log</code> .
<code>-profileName <profile></code>	Specify the profile to run the command against. If the command is run from <code><was_home>/bin</code> and <code>-profileName</code> is not specified, the default profile is used. If it is run from <code><profile_home>/bin</code> , that profile is used.
<code>-replacelog</code>	Replace the log file instead of appending to the current log.
<code>-trace</code>	Generate trace information into the log file for debugging purposes.
<code>-timeout <seconds></code>	Specify the wait time before node agent initialization times out and returns an error.
<code>-statusport <portNumber></code>	Set the port number for node agent status callback.

Parameter	Description
-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 5-11, for a sample of the startNode command.

Example 5-11 startNode command

```
C:\WebSphere\AppServer\profiles\Custom01\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
```

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: <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 5-16 on page 232.

Table 5-16 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 5-12, for an example and sample output of the stopNode command.

Example 5-12 stopNode command

```
C:\WebSphere\AppServer\profiles\Custom01\bin>stopNode  
ADMU0116I: Tool information is being logged in file
```

```
C:\WebSphere\AppServer\profiles\Custom01\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 (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**.

5.5.5 Node groups

With WebSphere Application Server V6, you can now have nodes in cells with different capabilities. Currently, this means having a cell with both nodes on distributed platforms and WebSphere for 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**.

- ▶ 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 5-26, we have displayed the **DefaultNodeGroup**.
- ▶ 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.

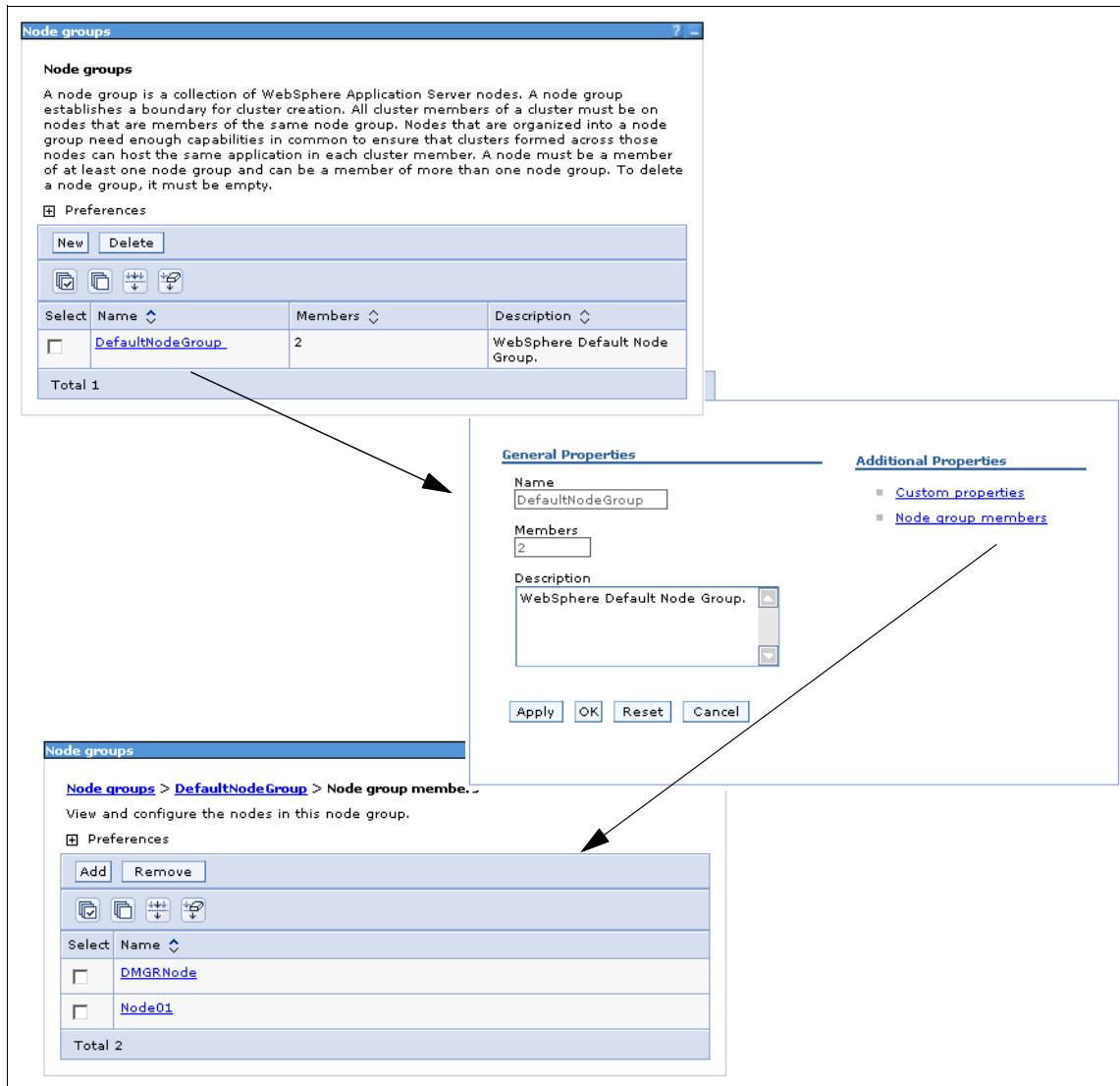


Figure 5-26 Displaying node groups

5.6 Working with clusters

This section discusses creating, configuring, and managing clusters using the administrative console. Clustering is an option in a Network Deployment installation only.

5.6.1 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 5-27 on page 236.

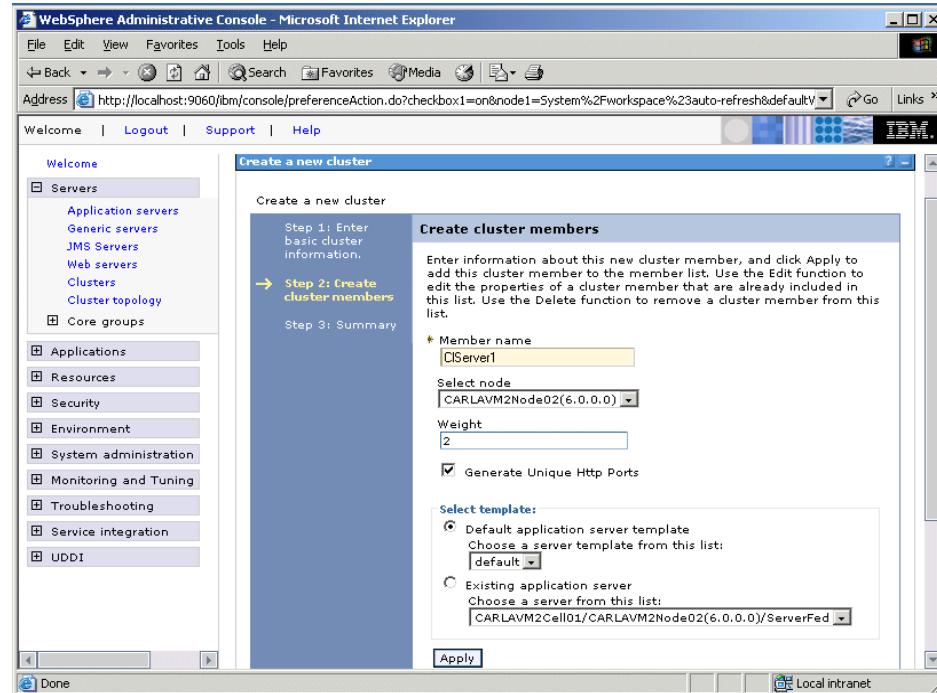


Figure 5-27 Creating a new cluster

3. Enter the information for the new cluster:
 - **Cluster name:** Enter a cluster name of your choice.

- **Prefer local:** This setting indicates that a request to an EJB should be routed to an EJB on the local node if available.
- **Internal replication domain:** Use this setting to indicate you want to use memory-to-memory replication for persistent session management and that a replication domain should be created. Replication domains are discussed in Chapter 12, “Session management” on page 697.
- **Existing server:** One application server must be added to the cluster. You can choose an existing application server from the list and create new application servers for the cluster.

If you want to use an existing server, select **Select an existing server to add to this cluster** and click **Next**. The server you select is added to the cluster and you are given the opportunity to add new servers also.

If you are not using an existing server, select **Do not include an existing server** in this cluster and click **Next**.

4. The next window, Figure 5-29, allows you to add servers to the cluster. For each server, enter a name, select the appropriate parameters, and click **Apply**. The servers in the cluster will be listed at the bottom of the window.

The screenshot shows the 'Create a new cluster' wizard, specifically Step 2: Create cluster members. The left sidebar lists steps: Step 1: Enter basic cluster information, Step 2: Create cluster members (which is active and highlighted in yellow), and Step 3: Summary. The main panel has a title 'Create cluster members' and instructions: 'Enter information about this new cluster member, and click Apply to add this cluster member to the member list. Use the Edit function to edit the properties of a cluster member that are already included in this list. Use the Delete function to remove a cluster member from this list.' It contains fields for 'Member name' (text input field containing 'CIServer1'), 'Select node' (dropdown menu showing 'CARLAVM2Node02(6.0.0.0)'), 'Weight' (text input field containing '2'), and a checked checkbox for 'Generate Unique Http Ports'. A large 'Apply' button is below these fields. At the bottom of the main panel is a table titled 'Cluster members' with columns 'Select', 'Application servers', 'Nodes', 'Version', and 'Weight'. The table shows one row for 'CIServer1' with 'CARLAVM2Node02' in the 'Nodes' column, '6.0.0.0' in 'Version', and '2' in 'Weight'. Below the table are 'Edit' and 'Delete' buttons. At the very bottom of the screen are 'Previous', 'Next', and 'Cancel' buttons.

Select	Application servers	Nodes	Version	Weight
<input type="checkbox"/>	CIServer1	CARLAVM2Node02	6.0.0.0	2

Figure 5-28 Add servers to the new cluster

- **Member Name:** Type a new name for the new server.
- **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:

```
% routed to Server1 = weight1 / (weight1+weight2+...+weight n)
```

In the formula, n represents the number of cluster members in the cluster.

- **Replication entry:** For information about replication domains and replication entries, see Chapter 12, “Session management” on page 697.

5. When all the servers have been entered, click **Next**.
6. A summary page shows you what will be created.
7. Click **Finish** to create the cluster and new servers.
8. Save the configuration.

5.6.2 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 5-29.



Figure 5-29 Cluster topology view

3. Selecting a server displays the attributes of the server as related to the cluster. See Figure 5-30.

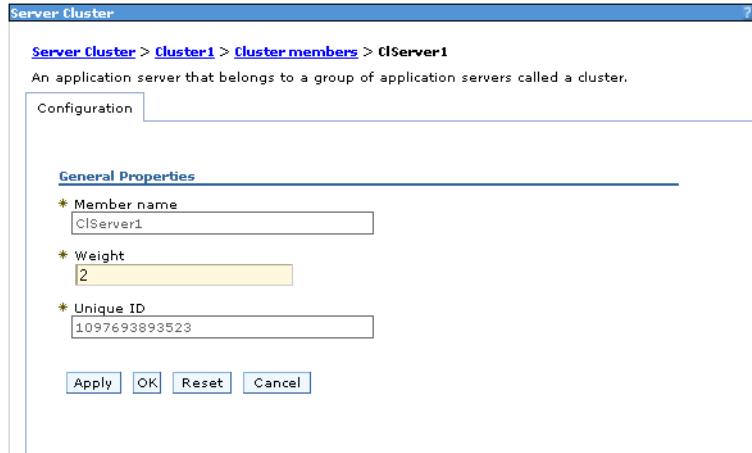


Figure 5-30 Viewing the cluster characteristics of a server

5.6.3 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 stops 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.

5.7 Working with virtual hosts

Note: For an example of defining and using a new virtual host, see 16.1.4, “Defining the WebSphere Bank virtual host” on page 914.

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.

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.

5.7.1 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 in 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 8.4.1, “Regenerating the plug-in configuration file” on page 406.

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.

5.8 Managing applications

Applications can be managed using the following methods:

- ▶ Using wsadmin script

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 on using wsadmin scripts is found in Chapter 6, “Administration with scripting” on page 267.

- ▶ 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, EJB JAR files), or application artifacts (Java source files, Java class files, images, JSPs etc.) 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.

This is covered in Chapter 17, “WebSphere Rapid Deployment” on page 957

- ▶ 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 16, “Deploying applications” on page 907.

5.8.1 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 5-31 on page 243. The following list describes the actions you can choose on this screen.

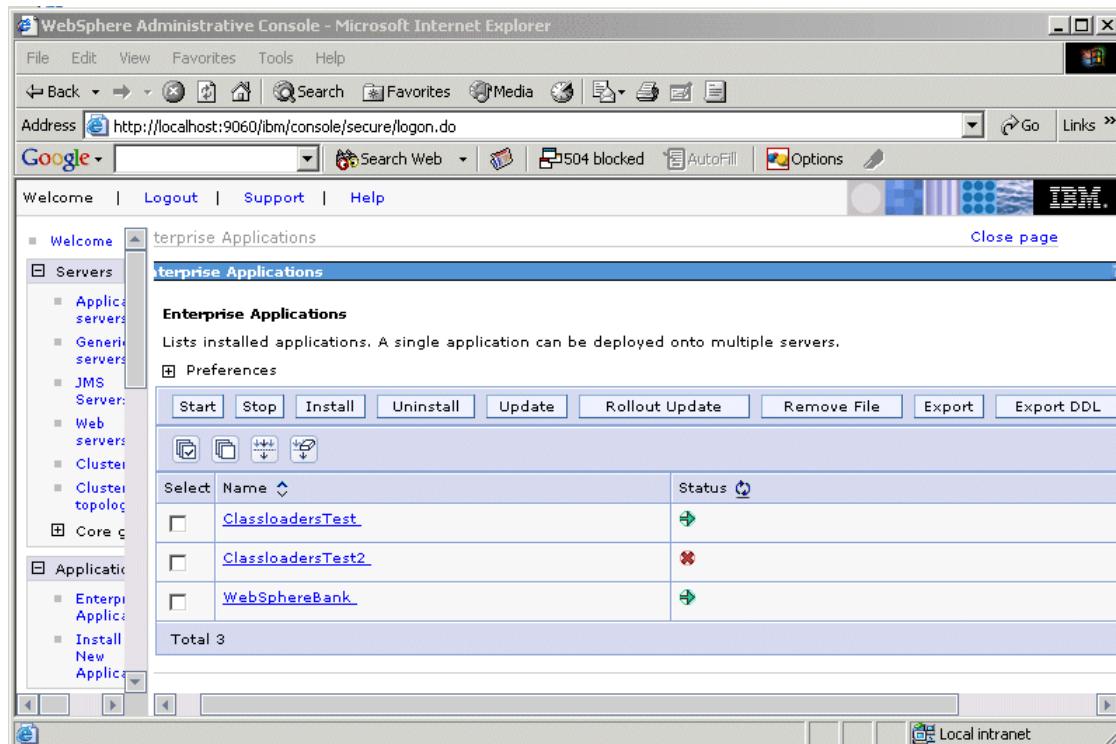


Figure 5-31 Working with enterprise applications

► 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.

5.8.2 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 5-32 on page 245.

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.

Click **Next**.

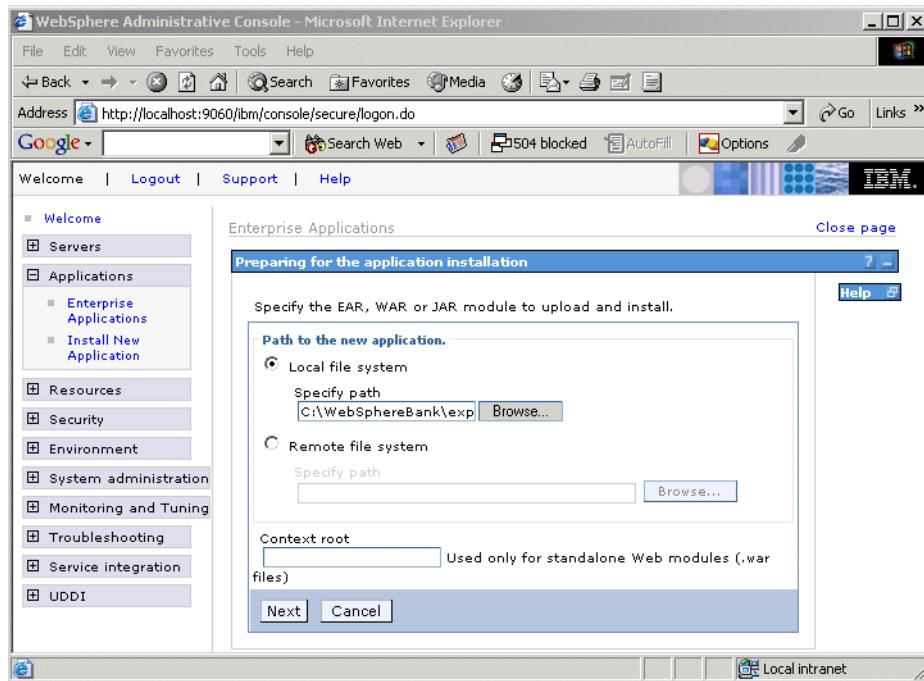


Figure 5-32 Installing an enterprise application

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. For example, you can elect to override existing bindings, specify values to use as a default for EJB 1.1 CMP bindings and connection factory bindings, and the default virtual host to use.
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:
 - 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 application servers and Web servers.
 - c. Select the backend ID to use for database access.
 - d. Provide listener bindings for message-driven beans.
 - e. Provide JNDI names for beans.
 - f. Bind message destination references to administered objects.

- g. Map JCA resource references to resources.
 - h. Provide default data source mapping for modules containing EJB 2.x entity beans.
 - i. Map data sources for all CMP 2.x data beans.
 - j. Map EJB references to beans.
 - k. Map resource references to resources.
 - l. Map virtual hosts for Web modules.
 - m. Map security roles to users/groups.
 - n. Ensure all unprotected 2.x methods have the correct level of protection.
 - o. Summary.
5. Click **Finish** to install the application.
 6. Save the configuration.

For information about where the application files are stored, see 3.4.3, “Application data files” on page 107.

5.8.3 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**.

5.8.4 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**.

5.8.5 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 on 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.

5.8.6 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**.

5.8.7 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 Mappings** in the Additional Properties table.

- d. Select the server for which you want to disable the application.
- e. Deselect the Enable option and click **OK**.
- f. Save the configuration.

5.8.8 Viewing installed applications

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 see the WAR files and JAR files associated with an enterprise 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 Additional Properties.

Figure 5-33 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

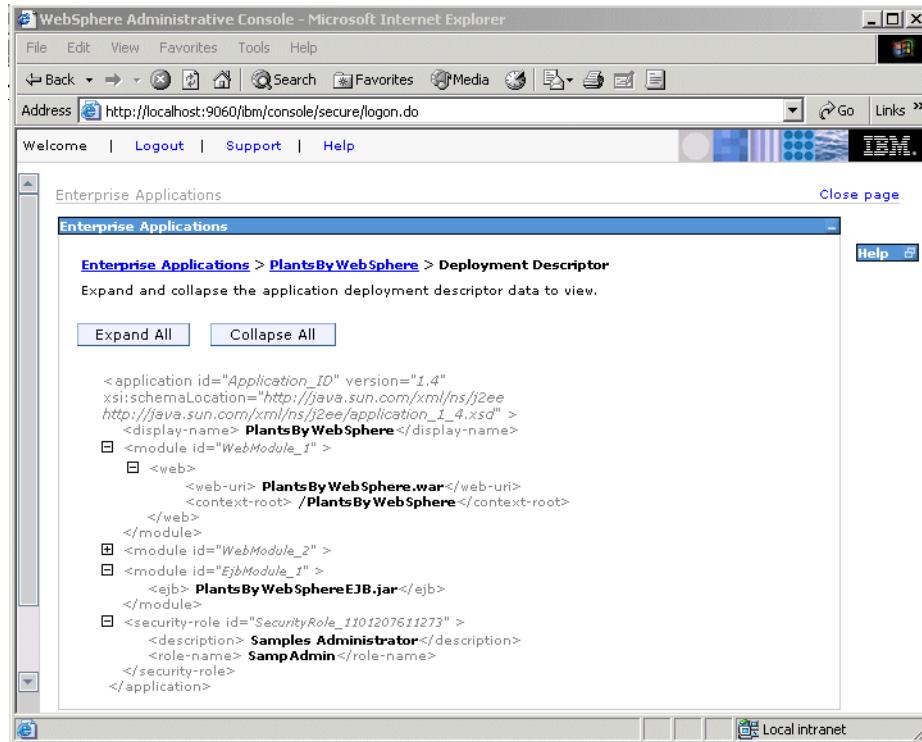


Figure 5-33 Enterprise application deployment descriptor

4. Click the application name in the navigation bar at the top to return to the enterprise application page. In addition to the Configuration tab, you have access to the Local Topology tab. This provides a view of the elements defined by the deployment descriptor. Each is a link to the configuration properties page for that element.

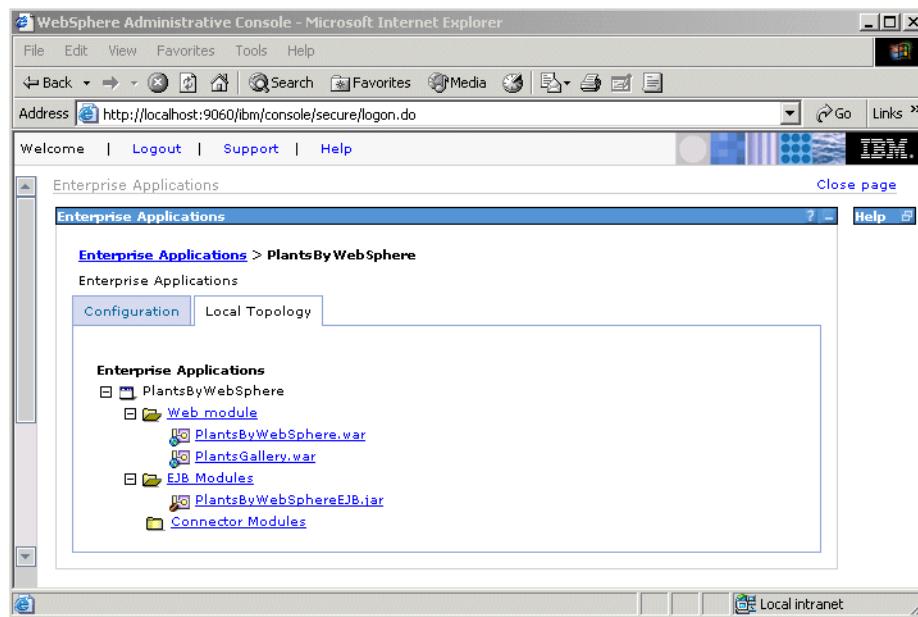


Figure 5-34 Topology view of the application

5.8.9 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 **EJB Modules** under Related Items.
4. Click the EJB module you want to view.

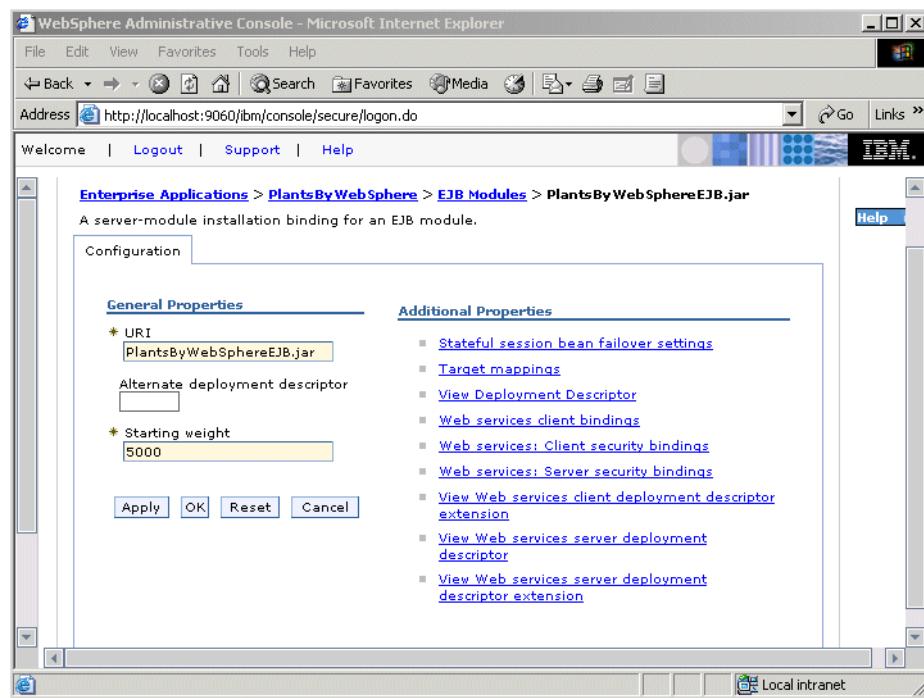


Figure 5-35 Viewing an EJB module configuration

5. Click **View Deployment Descriptor** under Additional Properties to see the EJB deployment descriptor.

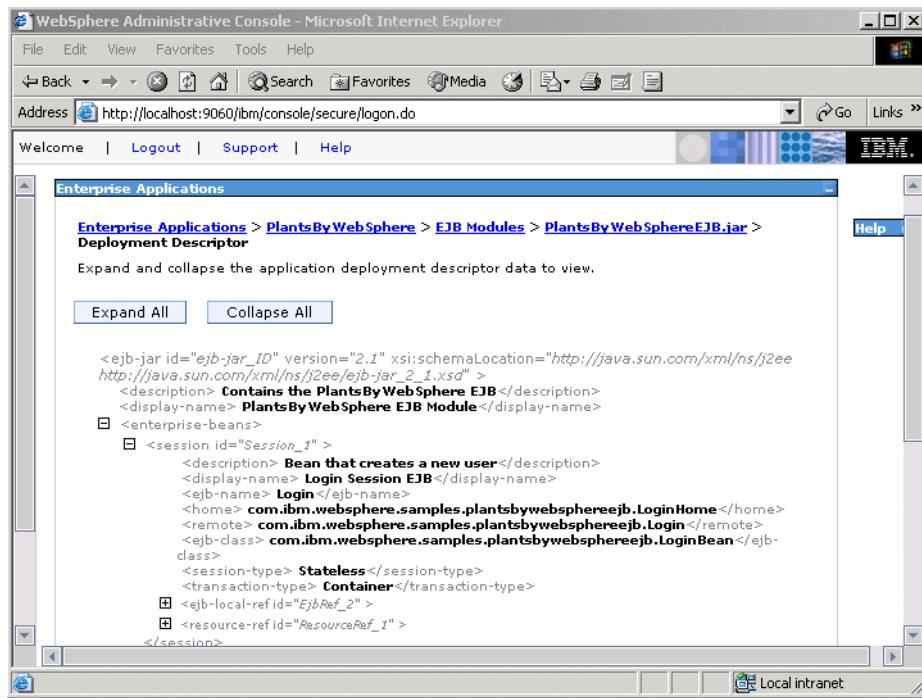


Figure 5-36 EJB module deployment descriptor

5.8.10 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 **Web Modules** under Related Items.
4. Click the Web module you want to view.
5. Click **View Deployment Descriptor**.

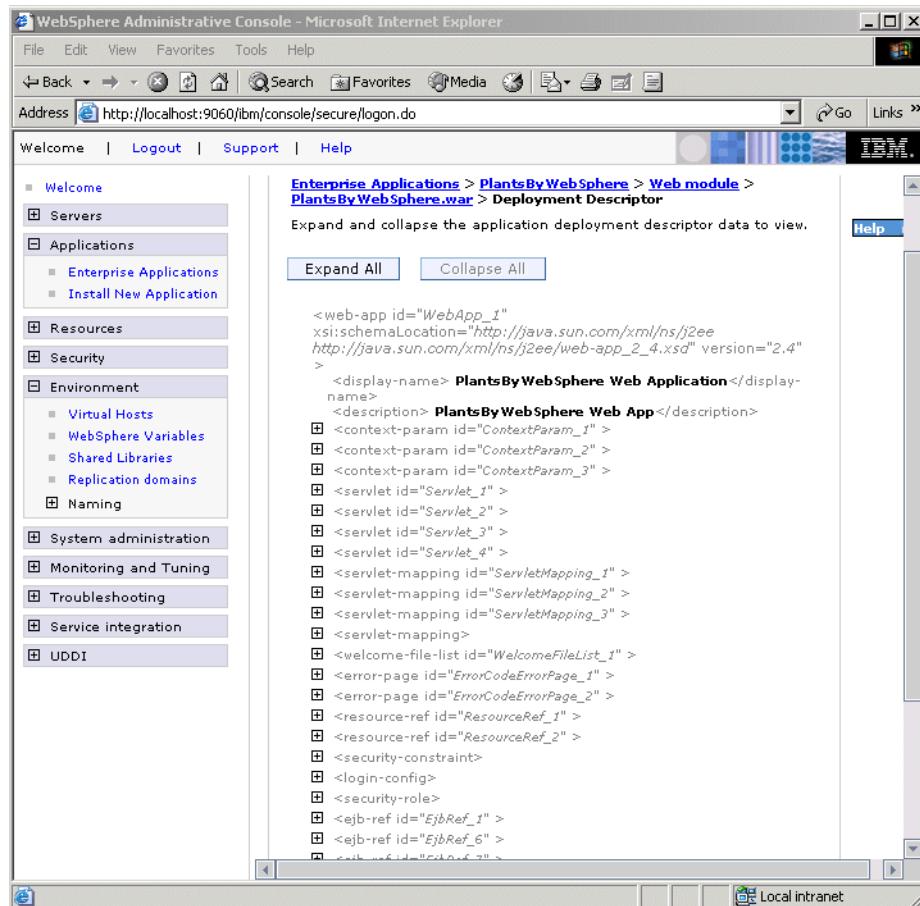


Figure 5-37 Web module deployment descriptor

5.8.11 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 **View Deployment Descriptor** under Additional Properties.

Figure 5-38 shows the deployment descriptor window for the DefaultApplication enterprise application. 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.

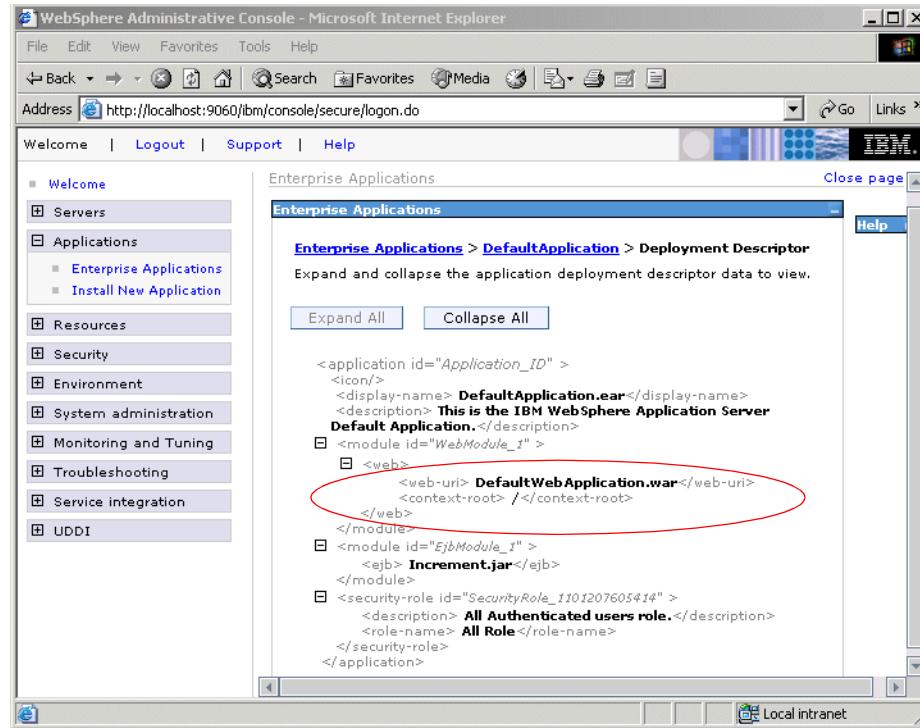


Figure 5-38 Deployment descriptor for DefaultApplication

- d. Click **Back** to return to the DefaultApplication configuration.
2. Find the URL for the snoop servlet:
 - a. In the DefaultApplication configuration page, select **Web Modules** under Related Items.
 - b. Click the DefaultWebApplication Web module to see the general properties.
 - c. Click **View Deployment Descriptor** under Additional Properties.

This displays the Web module properties window, as shown in Figure 5-39. Note that the URL pattern for the snoop servlet starting from the Web module context root is “/snoop/*”. The Web module context root was “/”.

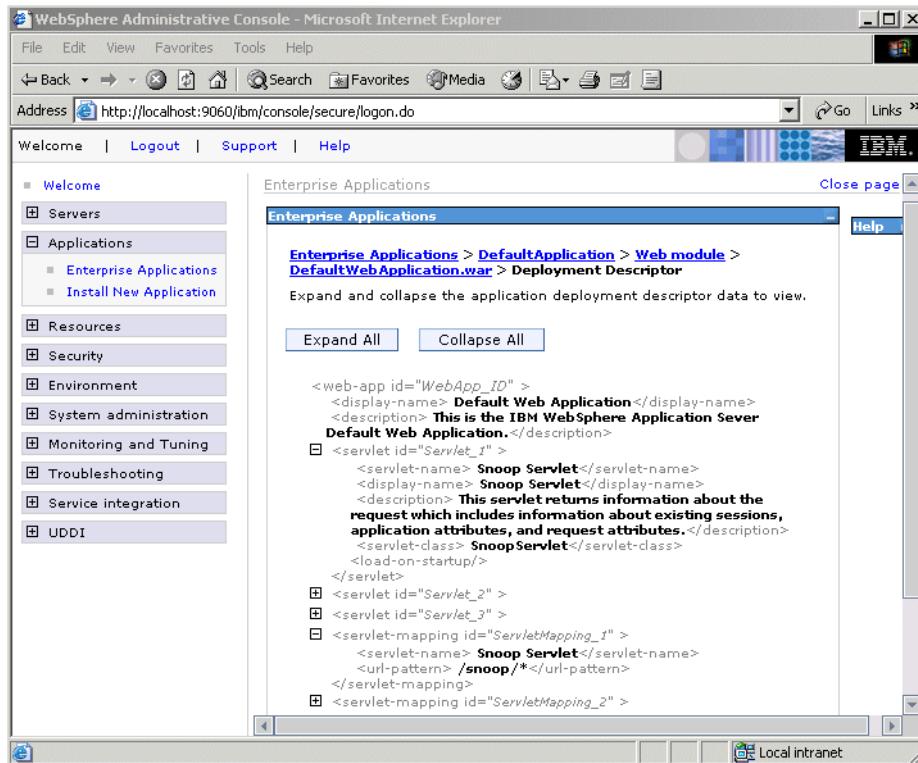


Figure 5-39 DefaultWebApplication Web module deployment descriptor

- 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 **Map virtual hosts for web modules** under Additional Properties.

This will display all of the Web modules contained in the enterprise application, and the virtual hosts in which they have been installed. Note that the DefaultWebApplication Web module has been installed on the default_host virtual host.

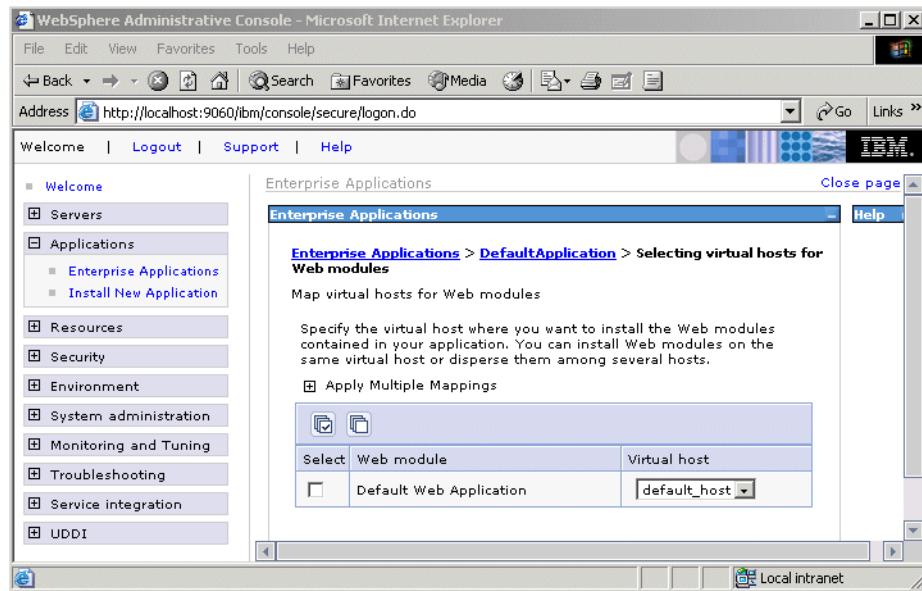


Figure 5-40 List of virtual hosts

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.

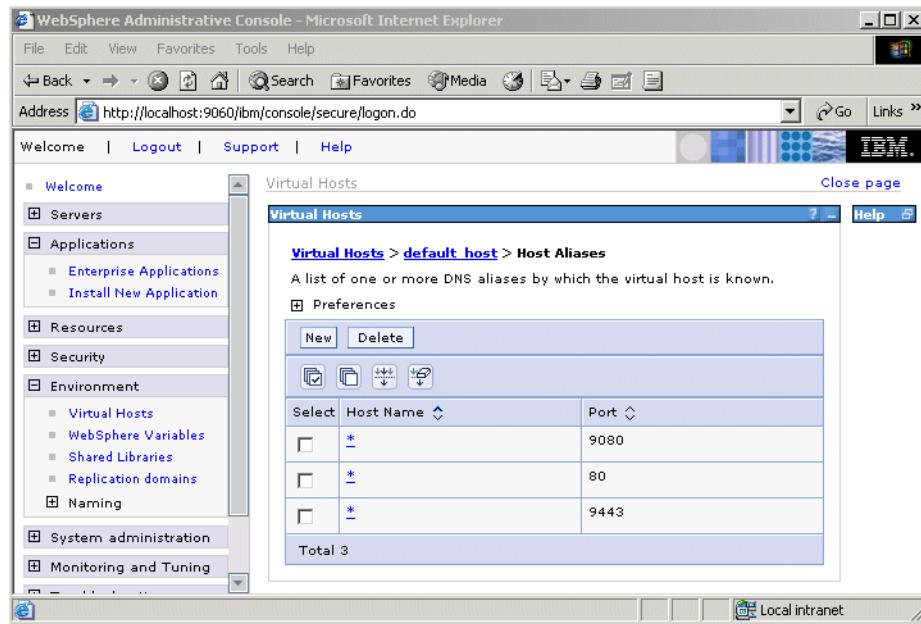


Figure 5-41 Default_host virtual host aliases

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
```

5.9 Managing your configuration files

This section summarizes some of the most common system management tasks:

- ▶ Backing up a node configuration
- ▶ Restoring a node configuration
- ▶ Backing up the cell configuration
- ▶ Restoring the cell configuration

5.9.1 Backing up a profile configuration

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 backup the default directory.
- ▶ Executing **backupConfig** from the `<profile_home>/bin` directory without the `-profileName` parameter will backup 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 standalone 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 5-17.

Table 5-17 *backupConfig* parameters

Parameter	Description
<code>-nostop</code>	Servers are not to be stopped before backing up the configuration.
<code>-quiet</code>	Suppresses the printing of progress information.
<code>-logfile <fileName></code>	Name of the log file to which information gets written. Default is <code><profile_home>/logs/backupConfig.log</code>
<code>-profileName <profile></code>	Profile to run the command against. If the command is run from <code><was_home>/bin</code> and <code>-profileName</code> is not specified, the default profile is used. If run from <code><profile_home>/bin</code> , that profile is used.
<code>-replacelog</code>	Replaces the log file instead of appending to the current log.
<code>-trace</code>	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	Prints command syntax information

Example

Example 5-13 shows an example of backing up a deployment manager.

Example 5-13 backupConfig example

```
C:\WebSphere\AppServer\bin>backupConfig
d:\WASbackups\DeploymentManagerNov022004 -profileName Dmgr01 -logfile
d:\WASbackups\logs\DeploymentManagerNov022004

DMU0116I: Tool information is being logged in file
          d:\WASbackups\logs\DeploymentManagerNov022004
ADMU0128I: Starting tool with the Dmgr01 profile
ADMU5001I: Backing up config directory
          C:\WebSphere\AppServer/profiles/Dmgr01/config to file
          D:\WASbackups\DeploymentManagerNov022004
ADMU0505I: Servers found in configuration:
ADMU0506I: Server name: dmgr
ADMU2010I: Stopping all server processes for node DMNode
ADMU0512I: Server dmgr cannot be reached. It appears to be stopped.
ADMU5002I: 152 files successfully backed up
```

5.9.2 Restoring a node configuration

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 5-18.

Table 5-18 restoreConfig parameters

Parameter	Description
-nowait	Don't wait for 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. 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	Prints command syntax information

Example

Example 5-14 shows an example of restoring an application server profile.

Example 5-14 restoreConfig example

```
C:\WebSphere\AppServer\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
```

5.9.3 Exporting and importing profiles

WebSphere Application Server V6 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.

A tutorial on creating and using archives can be found in the Information Center.
See

ftp://ftp.software.ibm.com/software/eod/WAS_6-0/SystemManagement/Presentations/WASv6_SM_Configuration_Archives/playershell.swf

Server archives

The following command 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, breaks the relationship between the server that you specify and the core group of the server, cluster, or SIBus 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

The following command can be used to create an archive of a profile:

```
$AdminTask exportWasprofile {-archive <archive_location>}
```

You can only create an archive of an unfederated profile (standalone application server).

```
$AdminTask importWasprofile {-archive <archive_location>}
```

5.9.4 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.
 - Stop the node agent.
 - Remove any nodes federated to the cell using the administrative console or the **removeNode** command. Removing a node doesn't delete it, but restores it to its pre-federated configuration that was saved as part of the federation process.
 - Delete the profile using **wasprofile -delete**.
 - Delete the <profile_home> directory.

- Verify that the registry entry for the profile is gone.
- ▶ If you are removing an application server profile that has not been federated to a cell:
 - Stop the application server.
 - Delete the profile using **wasprofile -delete**.
 - Delete the *<profile_home>* directory.
 - Verify that the registry entry for the profile is gone.
- ▶ 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 doesn't 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 **wasadmin -delete**.
 - Delete the *<profile_home>* directory.
 - Verify that the registry entry for the profile is gone.

Deleting a profile with wasprofile

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

```
wasprofile -delete
  -profileName <profile> | -profilePath <profile_path>
  [-debug]
```

At the completion of the command, the profile will be removed from the profile registry, and the runtime 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/wasprofile/wasprofile_delete_<profile_name>.log*

For example, in Example 5-15, you can see the use of the wasprofile command to delete the profile named Custom02.

Example 5-15 Deleting a profile using wasprofile

```
C:\WebSphere\AppServer\bin>wasprofile -delete -profileName Custom02
```

```
detectCurrentOSFamily:
setOSFileSeparator:
resolveNodeUninstScriptForTheCurrentPlatform:
runNodeUninstScript:
```

```
nodeUninstConfig:  
BUILD SUCCESSFUL  
Total time: 10 seconds  
  
detectCurrentOSFamily:  
setOSFileSeparator:  
defineOSSpecificConfigFlag:  
processJScriptForDeletion:  
    [copy] Copying 1 file to C:\WebSphere\AppServer\temp  
replaceAllInstancesOfGivenTokenWithValueForTheGivenFile:  
deleteStartMenuShortCut:  
    [exec] Microsoft (R) Windows Script Host Version 5.6  
    [exec] Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.  
deleteProfileShortCutFromStartMenu:  
BUILD SUCCESSFUL  
Total time: 1 second  
  
deleteProfile:  
    [delete] Deleting 183 files from C:\WebSphere\AppServer\profiles\Custom02  
    [delete] Deleted 53 directories from  
C:\WebSphere\AppServer\profiles\Custom02  
  
BUILD SUCCESSFUL  
Total time: 1 second  
INSTCONFSUCCESS: Success: The profile no longer exists.
```

As you can see in Example 5-15, 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 5-16.

Example 5-16 Verifying the delete profile results

```
C:\WebSphere\AppServer\bin>wasprofile -listProfiles  
[Dmgr01, AppSrv01, Custom01, Dmgr02]
```

```
C:\WebSphere\AppServer\bin>wasprofile -validateRegistry  
[]
```

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.



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:

- ▶ Overview of scripting concept
- ▶ Overview of wsadmin basics
- ▶ Common operational administration tasks using wsadmin
- ▶ Common configurational administration tasks using wsadmin
- ▶ Differences to WebSphere Application Server V5 scripts
- ▶ Overview of alternatives to scripting

In WebSphere Application Server V6.0 both the Jacl and Jython scripting languages are supported by the wsadmin tool. This chapter shows scripting using Jacl. Jython scripting looks similar; there are many examples in the Information Center that you can use to get started with Jython.

6.1 Overview of WebSphere scripting

WebSphere Application Server V6 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

Figure 6-1 shows the major components involved in the *wsadmin* scripting solution.

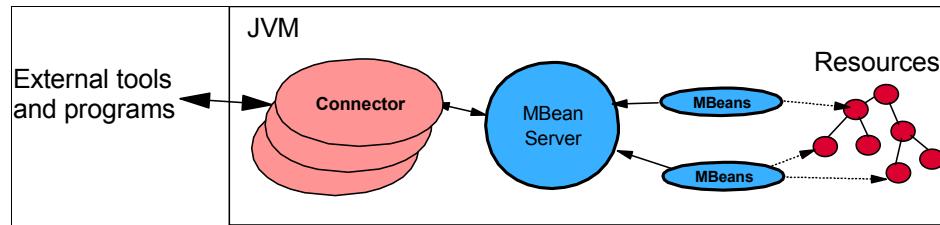


Figure 6-1 *wsadmin* scripting

Because *wsadmin* uses BSF, it can make various Java objects available through language-specific interfaces to scripts.

6.2 Using *wsadmin*

In this section, we describe how to start *wsadmin*, configurations you need to perform to launch *wsadmin* and how to get online information.

6.2.1 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 the **wsadmin** from a command prompt with the command:

```
<was_home>\profiles\<profile_name>\bin\wsadmin.bat (.sh)
```

Note that the wsadmin command file 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 6-1 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 6-1 wsadmin command-line options

```
C:\WebSphere\AppServer\profiles\HerkulesBase\bin>wsadmin -?
```

WASX7001I: wsadmin is the executable for WebSphere scripting.
Syntax:

```
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] |
    JMS <jms parms> |
    NONE
  ]
  [ script parameters ]
```

6.2.2 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 6-1.

Table 6-1 wsadmin properties

property	value
com.ibm.ws.scripting.connectionType	SOAP, RMI or NONE
com.ibm.scripting.host	host name of target system
com.ibm.scripting.port	TCP port of target system
com.ibm.ws.scripting.defaultLang	jacl or Jython
com.ibm.ws.scripting.traceFile	File for trace information
com.ibm.ws.scripting.traceString	=com.ibm.*=all=enabled

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 a standalone server environment this is not a concern.

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 in 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.

6.2.3 Commands and scripts 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 6-2. In the example, we use the AdminControl object to query the node name of the WebSphere server process.

Example 6-2 Running a single command in wsadmin

```
c:\was\app\profiles\HerkulesBase\bin>wsadmin -c "$AdminControl getNode"
WASX7209I: Connected to process "server1" on node HerkulesNode using SOAP
connector;
The type of process is: UnManagedProcess
HerkulesNode
```

Invoking commands interactively

The wsadmin command execution environment can be run in interactive mode, for invoking multiple commands without having the overhead of starting and stopping the wsadmin environment for every single command. Run **wsadmin** without the command (-c) and script file (-f) options to start the interactive command execution environment, as shown in Example 6-3.

Example 6-3 Starting the wsadmin interactive command execution environment

```
c:\was\app\profiles\HerkulesBase\bin>wsadmin
WASX7209I: Connected to process "server1" on node HerkulesNode using SOAP
connector;
The type of process is: UnManagedProcess
wsadmin>
```

From the wsadmin> prompt, the WebSphere administrative objects and built-in language objects can be invoked, as shown in Example 6-4. Type the commands as shown in bold..

Example 6-4 Interactive command invocation

```
wsadmin>puts "Running in interactive mode"
Running in interactive mode

wsadmin>$AdminControl getNode
HerkulesNode
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 6-5 shows a two-line Jacl script named myScript.jacl. The script has a .jacl extension, to reflect the Jacl language syntax of the script. The extension plays no significance to 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 6-5 Jacl script

```
puts "This is an example JACL script"
puts "[\$AdminControl getNode]"
```

Example 6-6 shows how to execute the script.

Example 6-6 Running a Jacl script in wsadmin

```
c:\was\app\profiles\HerkulesBase\bin>wsadmin -f myScript.jacl
WASX7209I: Connected to process "server1" on node HerkulesNode using SOAP
connector; The type of process is: UnManagedProcess
This is an example JACL script
HerkulesNode
```

As WebSphere also supports Jython, Example 6-7 shows the equivalent script in Jython. Note that the Jython script has a .py extension, to indicate the Python syntax of this script. Again, **wsadmin** does not use the extension, so the **-lang** option has to be specified in order to identify the correct language. Alternatively, you can specify the default language by setting the **com.ibm.ws.scripting.defaultLang** property in the **wsadmin.properties** file.

Example 6-7 Jython script

```
print "This is an example Jython script"  
print AdminControl.getNode()
```

Example 6-8 shows the execution of the Jython script.

Example 6-8 Running a Jython script in wsadmin

```
c:\was\app\profiles\HerkulesBase\bin>wsadmin -lang jython -f myScript.py  
WASX7209I: Connected to process "server1" on node HerkulesNode using SOAP  
connector; The type of process is: UnManagedProcess  
This is an example Jython script  
HerkulesNode
```

Note the difference in syntax between Jacl and Jython, but the equivalence of the use of the **AdminControl** object.

Choosing between Jacl and Jython is simply a question of programming style and preference. One is not better suited over the other. In fact they both offer the same interfaces to managing the WebSphere environment.

The WebSphere Application Server V6 Information Center has detailed descriptions of syntax and semantics for both the Jacl and Jython languages. Also generic programming guides exist for both (see “Online resources” on page 990). As the Jython language is simply an alternative implementation of Python V. 2.1 written entirely in Java, a general Python language guide is well suited for learning the Jython language. Jacl is a pure Java implementation of TCL and a standards TCL programming guide can be used for getting started with Jacl.

The purpose of this chapter is to provide details on **wsadmin** and examples of performing administrative tasks with scripting commands. The Jacl scripting language is the basis for this task in this book. However, you could use the Jython language as well.

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 6-9 shows an example of invoking `wsadmin` to execute a script file using a specific properties file.

Example 6-9 Specifying properties file on the command line

```
c:\was\app\profiles\HerkulesBase\bin>wsadmin -f c:\scriptexamples\myScript.jacl
-p c:\temp\custom.properties
```

WWASX7209I: Connected to process "server1" on node HerkulesNode using SOAP connector; The type of process is: UnManagedProcess

```
This is an example JACL script
MyServ
```

6.2.4 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

inquires 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. This is not of concern in standalone server environments.

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 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. They provide easier to use and task-oriented 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>$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 6-10.

Example 6-10 Getting method-specific help

```
wsadmin>$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  
is returned.
```

Similarly, you can get a brief description, as well as 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 279.

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*.

6.2.5 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. For performing common types of administrative and configurative tasks

without in-depth knowledge of the JMX framework and the WebSphere XML configuration structure, the AdminTask has been introduced with WebSphere Application Server V6. The following sections explain these wsadmin objects in more detail.

Administration using AdminControl

In order to invoke administrative methods on running JMX MBeans (as is the case for most AdminControl commands), 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 runtime 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 runtime 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 Jacl 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 of all the node agents and managed application servers that are manageable from this level in the WebSphere 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 6-11 on page 278 shows a Jacl script that collects information about running MBeans into a file named mbean.txt. The list returned by the **queryNames** command is a single Jacl 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 Jacl 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 6-11 Finding information for running MBeans

```
set file "mbean.txt"
set logFile [open $file a+]
set mbStr [$AdminControl queryNames "*:*"]
set mbList [split $mbStr $env(line.separator)]
foreach item $mbList {
    if {$item != ""} { puts $logFile "ObjectName: $item" }
}
close $logFile
```

An object name item returned by the queryNames command, could look like Example 6-12:

Example 6-12 Returned object name item

```
WebSphere:name=dmgr,process=dmgr,platform=common,node=PlatoCellManager,j2eeType=J2EEServer,version=6.0.0.0,type=Server,mbeanIdentifier=cells/PlatoCell/nodes/PlatoCellManager/servers/dmgr/server.xml#Server_1,cell=PlatoCell,processType=DeploymentManager
```

This represents a deployment manager (dmgr) running in cell PlatoCell on node PlatoCellManager. 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 platoCellManager, as follows:

```
$AdminControl queryNames WebSphere:type=Server,node=PlatoCellManager,*
```

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 instance, the deployment manager, the `completeObjectName` command could be a better choice. For example, this command would obtain the deployment manager object name:

```
$AdminControl completeObjectName  
    type=DeploymentManager,node=PlatoCellManager,*
```

Javadoc: All MBean types are documented in 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:\Program Files\WebSphere\AppServer\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 `AdminControl completeObjectName` command as explained previously.

Example 6-13 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 `SocratesNode`, 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 6-13 Finding attributes for a running MBean

```
wsadmin>set serv [$AdminControl completeObjectName  
    type=Server,node=SocratesNode,*]  
WASX7026W: String "type=Server,node=SocratesNode,*" corresponds to 2 different  
MBeans; returning first one.  
WebSphere:name=nodeagent,process=nodeagent,platform=common,node=SocratesNode,j2  
eeType=J2EEServer,version=6.0.0.0,type=Server,mbeanIdentifier=cells/PlatoCell/n  
odes/SocratesNode/servers/nodeagent/server.xml#Server_1097068263653,cell=PlatoC  
ell,processType=NodeAgent
```

```
wsadmin>$Help attributes $serv  
Attribute          Type          Access  
name              java.lang.String  R0  
shortName         java.lang.String  R0  
threadMonitorInterval int          RW  
threadMonitorThreshold int          RW  
threadMonitorAdjustmentThreshold int          RW  
pid               java.lang.String  R0  
cellName          java.lang.String  R0  
cellShortName     java.lang.String  R0
```

deployedObjects	[Ljava.lang.String;	RO
javaVMs	[Ljava.lang.String;	RO
nodeName	java.lang.String	RO
nodeShortName	java.lang.String	RO
processType	java.lang.String	RO
resources	[Ljava.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
objectName	java.lang.String	RO
stateManageable	boolean	RO
statisticsProvider	boolean	RO
eventProvider	boolean	RO
eventTypes	[Ljava.lang.String;	RO

Attribute values for any specific MBean can be read with the **getAttribute** command of the AdminControl object. Depending on 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:

```
$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 6-14 shows the usage of the **operation** command and its output.

Example 6-14 Finding operations for a running MBean (partial list of operations)

```
wsadmin>$Help operations $serv
Operation
java.lang.String getProductVersion(java.lang.String)
java.lang.String getComponentVersion(java.lang.String)
java.lang.String getEFixVersion(java.lang.String)
java.lang.String getPTFVersion(java.lang.String)
java.lang.String getExtensionVersion(java.lang.String)
[Ljava.lang.String; getVersionsForAllProducts()
[Ljava.lang.String; getVersionsForAllComponents()
[Ljava.lang.String; getVersionsForAllEFixes()
[Ljava.lang.String; getVersionsForAllPTFs()
[Ljava.lang.String; getVersionsForAllExtensions()
void stop()
void stopImmediate()
void stop(java.lang.Boolean, java.lang.Integer)
void restart()
```

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:

```
$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 `wsadmin` invocation syntax displayed for most operations. For more information, search for `MBeanInspector` on AlphaWorks.

Note: Even though MBI was not available for WebSphere version 6 at the time of this writing, 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.

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.

types

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 6-15 shows the partial output of the **types** command.

Example 6-15 Partial output of types command

```
wsadmin> $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 WebSphere 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 the section “Input and output of configuration object attributes” on page 285.

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 **parent**

command. Locate the parent configuration objects for the ApplicationServer object by issuing:

```
$AdminConfig parent ApplicationServer
```

getid

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 6-16 shows how to obtain the configuration name for SocratesServer1. 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 6-16 Finding configuration name of an object

```
sadmin>$AdminConfig getid "/Node:SocratesNode/Server:SocratesServer1/"
```

```
SocratesServer1(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|  
server.xml#Server_1097069277091)
```

Example 6-16 illustrates how the parent to child relationship of configuration objects comes into play. As the configuration object name for the SocratesServer1 residing on the SocratesNode 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.

list

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 6-17 list all objects of DataSource object type in the test environment. The **list** command returns five objects of DataSource type, one defined for the PlatoCell cell, two for the SocratesServer1 server, one for the SocratesNode node and one for the socServer2 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 6-17 Finding objects of the same object type

```
wsadmin>$AdminConfig list DataSource
"Cell Datasource(cells/PlatoCell|resources.xml#DataSource_1097095184323)"
BANKDS(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
       resources.xml#DataSource_1097097147246)
DefaultEJBTimerDataSource(cells/PlatoCell/nodes/SocratesNode/servers/
                           SocratesServer1|resources.xml#DataSource_1000001)
PLANTSDB(cells/PlatoCell/nodes/SocratesNode|
          resources.xml#DataSource_1097095254434)
SocratesNode.SocratesServer1-SocratesNodeSamplesBus(cells/PlatoCell/nodes/
                                                    SocratesNode/servers/socServer2|resources.xml#DataSource_1097097148798)
```

defaults

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 6-18 shows the usage of the **defaults** command to list the attributes and default values for those attributes for object type DynamicCache.

Example 6-18 Finding attributes and default values for an object type

```
wsadmin>$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              0
enableDiskOffload   boolean          false
diskOffloadLocation String          null
flushToDiskOnStop   boolean          false
```

enableTagLevelCaching	boolean	false
context	ServiceContext	
properties	Property	
cacheGroups	ExternalCacheGroup	
cacheReplication	DRSSettings	

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 6-19 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 6-19 Output of attribute command of AdminConfig object

```
wsadmin>$AdminConfig attributes DynamicCache
"cacheGroups ExternalCacheGroup*"
"cacheReplication DRSSettings"
"cacheSize int"
"context ServiceContext@"
"defaultPriority int"
"diskOffloadLocation String"
"enable boolean"
"enableCacheReplication boolean"
"enableDiskOffload boolean"
"enableTagLevelCaching boolean"
"flushToDiskOnStop boolean"
"hashSize int"
"properties Property(TypedProperty)*"
"pushFrequency int"
"replicationType ENUM(PULL, PUSH, PUSH_PULL, NONE)"
```



```
wsadmin>$AdminConfig attributes ExternalCacheGroup
```

```

"members ExternalCacheGroupMember*"
"name String"
"type ENUM(SHARED, NOT_SHARED)"

wsadmin>$AdminConfig attributes TypedProperty
"description String"
"name String"
"required boolean"
"type String"
"validationExpression String"
"value String"

```

In Example 6-19, 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 attribute is 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 6-20, the top-level attributes for the SocratesServer1 object is shown by use of the **show** command.

Example 6-20 Finding top-level attributes for a given object (formatted for readability)

```

wsadmin>$AdminConfig show [$AdminConfig getid
                           /Node:SocratesNode/Server:SocratesServer1]
{components
  {(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#NameServer_1097069277361)
   (cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#ApplicationServer_1097069277361)}`}
{customServices {}}
{developmentMode false}
{errorStreamRedirect
  {(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#StreamRedirect_1097069277361)`}
{name SocratesServer1}
{outputStreamRedirect
  {(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#StreamRedirect_1097069277362)`}
{parallelStartEnabled true}
{processDefinitions
  {(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#JavaProcessDef_1097069277371)`}}
{serverType APPLICATION_SERVER}
{services
  {(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#PMIService_1097069277091)`}}

```

```

(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#AdminService_1097069277091)
(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#TraceService_1097069277091)
(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#RASLoggingService_1097069277091)
(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#CoreGroupBridgeService_1097069277091)
(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#TPVService_1097069277091)
(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#ObjectRequestBroker_1097069277091)
(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#TransportChannelService_1097069277091)
(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#ThreadPoolManager_1097069277361)
(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#HTTPAccessLoggingService_1097069277361)})}
{stateManagement (cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#StateManageable_1097069277091)}
{statisticsProvider
    (cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
        server.xml#StatisticsProvider_1097069277091)}

```

The value for a particular attribute can be retrieved with the **showAttribute** command. In Example 6-21, the values for the name attribute and the services attribute of SocratesServer1 server object are listed.

Example 6-21 Retrieving attribute values for a given object (formatted for readability)

```

wsadmin>set serv [$AdminConfig getid
                    /Node:SocratesNode/Server:SocratesServer1]

wsadmin>$AdminConfig showAttribute $serv name
SocratesServer1

wsadmin>$AdminConfig showAttribute $serv services
{ (cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#PMIService_1097069277091)
(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#AdminService_1097069277091)
(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#TraceService_1097069277091)
(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#RASLoggingService_1097069277091)
(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#CoreGroupBridgeService_1097069277091)
(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#TPVService_1097069277091)

```

```
(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#ObjectRequestBroker_1097069277091)
(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#TransportChannelService_1097069277091)
(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#ThreadPoolManager_1097069277361)
(cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1|
    server.xml#HTTPAccessLoggingService_1097069277361)}
```

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 V6 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.

Configuring using AdminTask

Use of the classic `wsadmin` administrative objects 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.

AdminTask is introduced with support of a new command framework API in the WebSphere product. 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 server environment and WebSphere Application Server package.

Overview of AdminTask commands

The AdminTask object provides a large number of commands that performs simple and complex administrative tasks. In order to find a command for a specific task, commands have been logically grouped into command groups. 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 6-22.

Example 6-22 AdminTask Groups and SIBAdmin commands (formatted for readability)

```
wsadmin>$AdminTask help -commandGroups
WASX8005I: Available admin command groups:

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.
CoreGroupBridgeManagement - A group of administrative commands that help in
    configuring core groups.
CoreGroupManagement - A set of commands for modifying core groups
JCAManagement - A group of administrative commands that helps to configure
    Java 2 Connector Architecture (J2C)-related resources.
LocalModeGroup - No description available
ManagedObjectMetadata - Managed Object Metatadata Helper Commands
NodeGroupCommands - a group of admin commands for the Node Group Administration
SIBAdminBusSecurityCommands - SIB_ADMIN_SECURITY_COMMANDS_GROUP_DESCRIPTION
SIBAdminCommands - A group of commands that help configure SIB queues and
    messaging engines.
SIBJMSAdminCommands - A group of commands that help configure SIB JMS
    connection factories, queues and topics.
SIBWebServices - A group of commands to configure service integration
    bus Web services.
ServerManagement - A group of command that configure servers
TAMConfig - This group contains commands for configuring embedded Tivoli
    Access Manager.
UnmanagedNodeCommands - Commands to configure unmanaged nodes.
WSGateway - A group of commands to configure Web services gateway.
```

```
wsadmin>$AdminTask help SIBAdminCommands
WASX8007I: Detailed help for command group: SIBAdminCommands
```

Description: A group of commands that help configure SIB queues and messaging engines.

Commands:

```
createSIBus - Create a bus.
deleteSIBus - Delete a named bus, including everything on it.
listSIBuses - List all buses in the cell.
modifySIBus - Modify a bus.
showSIBus - Show the attributes of a bus.
addSIBusMember - Add a member to a bus.
```

```
removeSIBusMember - Remove a member from a bus.  
listSIBusMembers - List the members on a bus.  
showSIBusMember - Show a member from a bus.  
modifySIBusMember - Modify a bus member.  
createSIBEngine - Create a messaging engine.  
deleteSIBEngine - Delete the default engine or named engine from the  
    target bus.  
listSIBEngines - List messaging engines.  
showSIBEngine - Show a messaging engine's attributes.  
listSIBDestinations - List destinations belonging to a bus.  
createSIBDestination - Create bus destination.  
deleteSIBDestination - Delete bus destination.  
modifySIBDestination - Modify bus destination.  
showSIBDestination - Show a bus destination's attributes.  
createSIBMediation - Create a mediation.  
deleteSIBMediation - Delete a mediation.  
listSIBMediations - List the mediations on a bus.  
modifySIBMediation - Modify a mediation.  
showSIBMediation - Show the attributes of a mediation.  
mediateSIBDestination - Mediate a destination.  
unmediateSIBDestination - Mediate a destination.
```

All the available AdminTask commands can be retrieved in one list with use of the help command:

```
$AdminTask help -commands
```

In order to invoke the **createSIBus** command, a number of options are needed. To list the options for a command, invoke help on the command:

```
$AdminTask help createSIBus
```

An example of creating a service integration bus interactively is shown in Example 6-23 on page 291. 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:

```
$AdminConfig reset
```

Example 6-23 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 (secure): [true] 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]

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/PlatoCell/buses/WSBus|sib-bus.xml#SIBus_1097761138869)
```

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, that provide 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 6-24.

Example 6-24 createCluster help text

```
wsadmin>$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.

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.

6.3 Common operational administrative 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.

6.3.1 General approach for operational tasks

In order to invoke an operation on a running MBean, you first need to know the object 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, two commands can be combined into one command.

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 275 for information about how to do this.

6.3.2 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. In our test environment, we installed both WebSphere Application Server Express and Network Deployment on the same machine. To show the command syntax, we used the WebSphere sample applications.

The elements of our Network Deployment environment include:

- ▶ Server node: SocratesNode
- ▶ Deployment manager node: PlatoCellManager
- ▶ Node agent server: nodeagent
- ▶ Servers: SocratesServer1 and socServer2

6.3.3 Managing the deployment manager

This section describes how to start and stop tasks on the deployment manager using the WebSphere scripting interface 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 (see 5.3.2, “Starting and stopping the deployment manager” on page 187).

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 6-25 on page 294 shows an example of stopping the deployment manager.

Example 6-25 Stopping deployment manager using a single line command

```
wsadmin>$AdminControl stopServer dmgr PlatoCellManager  
WASX7337I: Invoked stop for server "dmgr" Waiting for stop completion.  
WASX7264I: Stop completed for server "dmgr" on node "PlatoCellManager"
```

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 6-26 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 6-26 Getting MBean information and stopping the deployment manager

```
wsadmin>set dmgr [$AdminControl completeObjectName  
    "type=Server,processType=DeploymentManager,*"]  
WebSphere:name=dmgr,process=dmgr,platform=common,node=PlatoCellManager,j2eeType  
=J2EEServer,version=6.0.0.0,type=Server,mbeanIdentifier=cells/PlatoCell/nodes/P  
latoCellManager/servers/dmgr/server.xml#Server_1,cell=PlatoCell,processType=Dep  
loymentManager  
wsadmin>$AdminControl invoke $dmgr stop
```

6.3.4 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 5.5.4, “Starting and stopping nodes” on page 230.

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. In a WebSphere cell, 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 6-27 shows the command to stop a node agent.

Example 6-27 Single line command to stop a node agent

```
wsadmin>$AdminControl stopServer nodeagent SocratesNode
WASX7337I: Invoked stop for server "nodeagent" Waiting for stop completion.
WASX7264I: Stop completed for server "nodeagent" on node "SocratesNode"
```

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 6-28 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 6-28 Getting MBean information for a node agent Server object

```
wsadmin>set naServer [$AdminControl completeObjectName
                      "type=Server,node=SocratesNode,name=nodeagent,*"]
WebSphere:name=nodeagent,process=nodeagent,platform=common,node=SocratesNode,j2
eeType=J2EEServer,version=6.0.0.0,type=Server,mbeanIdentifier=cells/PlatoCell/n
odes/SocratesNode/servers/nodeagent/server.xml#Server_1097068263653,cell=PlatoC
ell,processType=NodeAgent

wsadmin>$AdminControl invoke $naServer stop
```

6.3.5 Managing application servers

This section describes how to perform common administration tasks on application servers using wsadmin.

Starting an application server

In a Network Deployment environment the node agent can start an application server. Example 6-29 shows the command for starting the socServer2 application server by use of the **startServer** command.

Example 6-29 Start an application server

```
wsadmin>$AdminControl startServer socServer2 SocratesNode
WASX7262I: Start completed for server "socServer2" on node "SocratesNode"
```

You can also use the **launchProcess** operation on the NodeAgent to start the socServer2 application server. Example 6-30 on page 296 shows the command syntax to query the MBean information for the NodeAgent object and to invoke the **launchProcess** operation from the identified MBean.

Example 6-30 Getting MBean information for a node agent NodeAgent object

```
wsadmin>set naMain [$AdminControl completeObjectName
   "name=NodeAgent,node=SocratesNode,type=NodeAgent,*"]  
WebSphere:platform=common,cell=PlatoCell,version=6.0.0.0,name=NodeAgent,mbeanId  
entifier=NodeAgent,type=NodeAgent,node=SocratesNode,process=nodeagent  
wsadmin>$AdminControl invoke $naMain launchProcess {{socServer2}}  
true
```

Stopping an application server

Example 6-31 shows the command for stopping the socServer2 application server.

Example 6-31 Stop an application server

```
wsadmin>$AdminControl stopServer socServer2 SocratesNode  
WASX7337I: Invoked stop for server "socServer2" Waiting for stop completion.  
WASX7264I: Stop completed for server "socServer2" on node "SocratesNode"
```

You can also use the AdminControl object to invoke the stop method on the application server. To do this you need to identify the MBean representing the application server. Example 6-32 shows the command to query the MBean information of the application server and stop the server.

Example 6-32 MBean information for Application Server server1

```
wsadmin>set socSrv2 [$AdminControl completeObjectName
   "name=socServer2,node=SocratesNode,type=Server,*"]  
WebSphere:name=socServer2,process=socServer2,platform=common,node=SocratesNode,  
j2eeType=J2EEServer,version=6.0.0.0,type=Server,mbeanIdentifier=cells/PlatoCell  
/nodes/SocratesNode/servers/socServer2/server.xml#Server_1097183733881,cell=Pla  
toCell,processType=ManagedProcess  
wsadmin>$AdminControl invoke $socSrv2 stop
```

If there are multiple application servers running on a node, you can stop all the servers from a single script. Example 6-33 on page 297 shows a script that stops all application servers on the SocratesNode node. In this example, the node name is hard-coded, but it is also possible to write Jacl code that accepts the node name from the command line or a menu.

To invoke the script from a command, type the following:

```
cd \WebSphere\Appserver\profiles\<profile_name>\bin  
wsadmin -f <script_filename>
```

Example 6-33 Stopping all application servers on a node

```
set servername [$AdminControl queryNames  
node=SocratesNode,type=Server,processType=ManagedProcess,*]  
foreach item $servername {  
    set shortname [$AdminControl getAttribute $item name]  
    set completename [$AdminControl completeObjectName  
type=Server,node=SocratesNode,name=$shortname,*]  
    puts "Stopping server : $shortname"  
    $AdminControl invoke $completename stop {}  
}
```

6.3.6 Managing enterprise applications

This section describes how to perform common administration tasks on enterprise applications using the scripting interface, wsadmin.

Viewing installed applications

Use the **AdminApp** object to view the applications installed on an application server. Example 6-34 shows the use of the list command and the resulting output.

Example 6-34 Listing installed applications

```
wsadmin>$AdminApp list  
DefaultApplication  
PlantsByWebSphere  
Query  
SamplesGallery  
WebSphereBank
```

You can also do this by querying the MBeans for running applications on a node. Example 6-35 shows you how to perform this task.

Example 6-35 Listing applications by MBeans query

```
wsadmin>$AdminControl queryNames type=Application,node=PericlesNode,*  
WebSphere:name=DefaultApplication,process=server1,platform=dynamicproxy,node=Pe  
riclesNode,J2EEName=DefaultApplication,Server=server1,version=6.0.0.0,type=Appl  
ication,mbeanIdentifier=cells/PericlesCell/applications/DefaultApplication.ear/  
deployments/DefaultApplication/deployment.xml#ApplicationDeployment_10970951643  
25,cell=PericlesCell  
...
```

Running objects are represented by MBeans. If an object is not running the MBean for that object does not exist. Based on this, we can write a simple JACL script that will display running applications.

Example 6-36 shows a script using the AdminApp object which lists the installed applications. The data obtained is configurational data and cannot be interrogated to determine runtime 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 6-36 Script to display the status of Applications

```
set application [$AdminApp list]
foreach app $application {
    set objName [$AdminControl queryNames type=Application,name=$app,*]
    if {[llength $objName] ==0} {
        puts "The Application $app is not running"
    } else {
        puts "The Application $app is running"
    }
}
```

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 6-37 shows the sequence of commands used to query the MBean and stop the application.

Example 6-37 Stopping a running application

```
wsadmin>set appservername [$AdminControl queryNames
type=ApplicationManager,node=PericlesNode,process=server1,*]
WebSphere:platform=dynamicproxy,cell=PericlesCell,version=6.0.0.0,name=ApplicationManager,mbeanIdentifier=ApplicationManager,type=ApplicationManager,node=PericlesNode,process=server1
wsadmin>$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 6-38 on page 299 shows the sequence of commands used to start the DefaultApplication application.

Example 6-38 Starting a stopped application

```
wsadmin>set appservername [$AdminControl queryNames  
type=ApplicationManager,node=PericlesNode,process=server1,*]  
WebSphere:platform=dynamicproxy,cell=PericlesCell,version=6.0.0.0,name=Applicat  
ionManager,mbeanIdentifier=ApplicationManager,type=ApplicationManager,node=Peri  
clesNode,process=server1  
wsadmin>$AdminControl invoke $appservername startApplication DefaultApplication
```

6.3.7 Managing clusters

This section describes how to perform common administration tasks on clusters using wsadmin.

Starting a cluster

Example 6-39 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 tstC1st with the cluster object name. The final command invokes the start operation on the cluster object.

Example 6-39 Start a cluster

```
wsadmin>$AdminControl queryNames type=Cluster,*  
WebSphere:platform=common,cell=PlatoCell,version=6.0.0.0,name=testCluster,mbean  
Identifier=testCluster,type=Cluster,node=PlatoCellManager,process=dmgr  
  
wsadmin>set tstC1st [$AdminControl completeObjectName  
type=Cluster,name=testCluster,*]  
WebSphere:platform=common,cell=PlatoCell,version=6.0.0.0,name=testCluster,mbean  
Identifier=testCluster,type=Cluster,node=PlatoCellManager,process=dmgr  
  
wsadmin>$AdminControl invoke $tstC1st start
```

Stopping a cluster

Example 6-40 on page 299 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 tstC1st with the cluster object name. The final command invokes the stop operation on the cluster object.

Example 6-40 Stopping a cluster

```
wsadmin>$AdminControl queryNames type=Cluster,*  
WebSphere:platform=common,cell=PlatoCell,version=6.0.0.0,name=testCluster,mbean  
Identifier=testCluster,type=Cluster,node=PlatoCellManager,process=dmgr
```

```
wsadmin>set tstC1st [$AdminControl completeObjectName  
type=Cluster,name=testCluster,*]  
WebSphere:platform=common,cell=PlatoCell,version=6.0.0.0,name=testCluster,mbean  
Identifier=testCluster,type=Cluster,node=PlatoCellManager,process=dmgr  
  
wsadmin>$AdminControl invoke $tstC1st stop
```

6.3.8 Generating the Web server plug-in configuration

Example 6-41 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 on a node. The second command generates the Web server plug-in.

Example 6-41 Generating the Web server plug-in

```
wsadmin>set pluginGen [$AdminControl completeObjectName  
type=PluginCfgGenerator,*]  
WebSphere:platform=common,cell=PlatoCell,version=6.0.0.0,name=PluginCfgGenerato  
r,mbeanIdentifier=PluginCfgGenerator,type=PluginCfgGenerator,node=PlatoCellMana  
ger,process=dmgr  
  
wsadmin>$AdminControl invoke $pluginGen generate "C:/WebSphere/Appserver  
C:/WebSphere/AppServer/profiles/PlatoDMGR/config PlatoCell PlatoCellManager  
dmgr plugin-cfg.xml"
```

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.

6.3.9 Enabling tracing for WebSphere components

Tracing used for problem determination is discussed in 9.4, “Traces” on page 430. 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 6-42 shows how to use `queryNames` to list the TraceService MBeans.

Example 6-42 List of TraceService MBeans

```
wsadmin>$AdminControl queryNames type=TraceService,*  
WebSphere:platform=common,cell=PlatoCell,version=6.0.0.0,name=TraceService,mbeanIdentifier=cells/PlatoCell/nodes/PlatoCellManager/servers/dmgr/server.xml#TraceService_1,type=TraceService,node=PlatoCellManager,process=dmgr  
  
WebSphere:platform=common,cell=PlatoCell,version=6.0.0.0,name=TraceService,mbeanIdentifier=cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1/server.xml#TraceService_1097069277091,type=TraceService,node=SocratesNode,process=SocratesServer1  
  
WebSphere:platform=common,cell=PlatoCell,version=6.0.0.0,name=TraceService,mbeanIdentifier=cells/PlatoCell/nodes/SocratesNode/servers/nodeagent/server.xml#TraceService_1097068263653,type=TraceService,node=SocratesNode,process=nodeagent
```

To start tracing for a server, you need to locate the TraceService MBean for the server process using the `completeObject` command. Example 6-43 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 6-43 Enable tracing using TraceService mbean

```
wsadmin>set ts [$AdminControl completeObjectName  
type=TraceService,process=SocratesServer1,*]  
WebSphere:platform=common,cell=PlatoCell,version=6.0.0.0,name=TraceService,mbeanIdentifier=cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1/server.xml#TraceService_1097069277091,type=TraceService,node=SocratesNode,process=SocratesServer1  
  
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.

6.4 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:

- ▶ 6.4.1, “General approach for configuration tasks” on page 302
- ▶ 6.4.2, “Specific examples of WebSphere configuration tasks” on page 302

6.4.1 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 Jacl lists. A Jacl 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.

6.4.2 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 6-44. 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 6-44 Creating an application server using AdminTask

```
wsadmin>$AdminTask createApplicationServer -interactive
Create Server

Command that creates a server

*Node Name: SocratesNode
*Server Name (name): socServer3
Template Name (templateName):
Generate Unique Ports (genUniquePorts): [true]
template location (templateLocation):
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
SocratesNode {-name socServer3 }
```

```
socServer3(cells/PlatoCell/nodes/SocratesNode/servers/socServer3|server.xml#Server_1098190565885)
```

```
wsadmin>$AdminConfig save
```

The alternative approach to using AdminTask for creating an application server is using the AdminConfig object. Example 6-45 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 6-45 Creating an application server using AdminConfig

```
wsadmin>set node [$AdminConfig getid /Node:SocratesNode/]
SocratesNode(cells/PlatoCell/nodes/SocratesNode|node.xml#Node_1)

wsadmin>$AdminConfig create Server $node {{name socServer4}}
socServer4(cells/PlatoCell/nodes/SocratesNode/servers/socServer4|server.xml#Server_1098190899605)

wsadmin>$AdminConfig save
```

Removing an application server

As with creating application servers, an application server can be removed by either use of the AdminTask object or the AdminConfig object. Example 6-46 illustrates removing an application server using AdminTask.

Example 6-46 Remove an application server using AdminTask

```
wsadmin>$AdminTask deleteServer -interactive
Delete Server

Delete a server configuration

*ADMG0106I (serverName): socServer4
*ADMG0104I (nodeName): SocratesNode
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  
socServer4 -nodeName SocratesNode }
```

```
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:

```
$AdminApp installInteractive  
C:/WebSphere/AppServer/samples/lib/BeenThere/BeenThere.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:

```
$AdminApp options  
C:/WebSphere/AppServer/samples/lib/BeenThere/BeenThere.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 Jacl, 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 6-47 shows an example of installing a new application named BeenThere on a server named socServer2.

Example 6-47 Installing an application

```
wsadmin>$AdminApp install  
C:/WebSphere/AppServer/samples/lib/BeenThere/BeenThere.ear {-node SocratesNode  
-server socServer2 -appname BeenThere}  
....  
wsadmin>$AdminConfig save
```

Uninstalling an enterprise application

The general syntax for uninstalling enterprise application is:

```
$AdminApp uninstall <application name>
```

Example 6-48 shows an example of uninstalling an application.

Example 6-48 Uninstalling an enterprise application

```
wsadmin>$AdminApp uninstall BeenThere  
ADMA5017I: Uninstallation of BeenThere started.  
ADMA5104I: The server index entry for  
WebSphere:cell=PlatoCell,node=SocratesNode is updated successfully.  
ADMA5102I: The configuration data for BeenThere from the configuration  
repository is deleted successfully.  
ADMA5011I: The cleanup of the temp directory for application BeenThere is  
complete.  
ADMA5106I: Application BeenThere 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 6-49, 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 6-49 Edit an enterprise application

```
wsadmin>$AdminApp edit BeenThere {-MapModulesToServers {"BeenThere WAR"
    BeenThere.war,WEB-INF/web.xml
    WebSphere:cell=PlatoCell,node=SocratesNode,server=socServer3}}}

wsadmin>$AdminConfig save
```

Preventing startup of an application

To prevent 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 6-50 the steps to locate, modify and save the target property are outlined.

Note the use of **lindex** in this example. All commands that return lists, are nested within a **lindex** method call in order to retrieve the target list item. Also, be aware that the **modify** command takes a list option containing list items of property name and value to modify.

Example 6-50 Disable of enterprise application on target server

```
wsadmin>$AdminConfig list ApplicationDeployment
(cells/PericlesCell/applications/WebSphereBank.ear/deployments/WebSphereBank|
 deployment.xml#ApplicationDeployment_1097097167195)

wsadmin>set eaBk [lindex [$AdminConfig list ApplicationDeployment] 0]
(cells/PericlesCell/applications/WebSphereBank.ear/deployments/WebSphereBank|
```

```

deployment.xml#ApplicationDeployment_1097097167195)

wsadmin>$AdminConfig show $wsBank
....
{targetMappings {(cells/PericlesCell/applications/WebSphereBank.ear/
    deployments/WebSphereBank|
    deployment.xml#DeploymentTargetMapping_1097097167195)}}
....
wsadmin>set eaBTgt [lindex [$AdminConfig showAttribute $eaBk targetMappings] 0]
(cells/PericlesCell/applications/WebSphereBank.ear/deployments/WebSphereBank|
    deployment.xml#DeploymentTargetMapping_1097097167195)

wsadmin>$AdminConfig modify $eaBTgt {{enable false}}

wsadmin>$AdminConfig queryChanges
WASX7146I: The following configuration files contain unsaved changes:

cells/PericlesCell/applications/WebSphereBank.ear/deployments/WebSphereBank/dep-
loyment.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. Virtual host is a WebSphere resource defined in a WebSphere cell. Therefore, by creating a virtual host we are modifying the configuration of the WebSphere cell object. Example 6-51 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 6-51 Find an object using AdminConfig command

```

wsadmin>set cell [$AdminConfig getid /Cell:PlatoCell/]
PlatoCell(cells/PlatoCell|cell.xml#Cell_1)

wsadmin>$AdminConfig create VirtualHost $cell {{name WSBank}}
WSBank(cells/PlatoCell|virtualhosts.xml#VirtualHost_1098199040140)

wsadmin>$AdminConfig save

```

Modifying a virtual host

Modify the virtual host configuration with the **modify** command in the AdminConfig object. Example 6-52 shows an example of modifying a virtual host. The example gets the ID of the WSBank virtual host, then uses that ID in the **modify** command to redefine the list of aliases.

Example 6-52 Modifying a virtual host

```
wsadmin>set wsbVHost [$AdminConfig getid /VirtualHost:WSBank/]
WSBank(cells/PlatoCell|virtualhosts.xml#VirtualHost_1098199040140)

wsadmin>$AdminConfig modify $wsbVHost {{aliases {{hostname *}{port
9082}}}{hostname *}{port 80}}}}}

wsadmin>$AdminConfig save
```

Modifying an application server

Modify an application server configuration using the AdminConfig object. The **modify** command is used for changing 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 runtime 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 runtime.

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 traverse through the following:

Application Server →Process Definition →Monitoring Policy →Ping Interval

Example 6-53 on page 310 shows an example of changing the ping interval for a server named socServer3.

Example 6-53 Modifying an application server

```
wsadmin>$AdminControl stopServer socServer3 SocratesNode  
  
WASX7337I: Invoked stop for server "socServer3" Waiting for stop completion.  
WASX7264I: Stop completed for server "socServer3" on node "SocratesNode"  
  
wsadmin>set srv [$AdminConfig getid /Node:SocratesNode/Server:socServer3]  
socServer3(cells/PlatoCell/nodes/SocratesNode/servers/socServer3|server.xml#Ser  
ver_1098190565885)  
  
wsadmin>set prcDef [$AdminConfig list ProcessDef $srv]  
(cells/PlatoCell/nodes/SocratesNode/servers/socServer3|server.xml#JavaProcessDe  
f_1098190565895)  
  
wsadmin>set monPol [$AdminConfig list MonitoringPolicy $prcDef]  
(cells/PlatoCell/nodes/SocratesNode/servers/socServer3|server.xml#MonitoringPol  
icy_1098190565895)  
  
wsadmin>$AdminConfig modify $monPol {{pingInterval 120}}  
  
wsadmin>$AdminConfig save  
  
wsadmin>$AdminControl startServer socServer3 SocratesNode  
WASX7262I: Start completed for server "socServer3" on node "SocratesNode"
```

Creating a cluster

To create a new cluster use either the AdminTask or AdminConfig object. In Example 6-54, the AdminTask object is used for creating a cluster named T4SCluster adding an existing server named socServer3 as a cluster member.

Example 6-54 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)
```

```
S (Select)
N (Next)
C (Cancel)
H (Help)
```

```
Select [S, N, C, H]: [S] S
```

```
Cluster Configuration (clusterConfig)
```

```
*Cluster Name (clusterName):
Prefer Local (preferLocal):
```

```
Select [C (Cancel), E (Edit)]: [E] E
*Cluster Name (clusterName): T4SCluster
Prefer Local (preferLocal): [true]
Create Server Cluster
```

```
Creates a new application server cluster.
```

1. Cluster Configuration (clusterConfig)
- > 2. Replication Domain (replicationDomain)
3. Convert Server (convertServer)

```
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)

```
S (Select)
P (Previous)
F (Finish)
C (Cancel)
H (Help)
```

```

Select [S, P, F, C, H]: [F] S

Convert Server (convertServer)

Converted Server Node Name (serverNode):
Converted Server Name (serverName):
Member Weight (memberWeight):
Node Group (nodeGroup):
Create Replicator Entry (replicatorEntry):

Select [C (Cancel), E (Edit)]: [E] E
Converted Server Node Name (serverNode): SocratesNode
Converted Server Name (serverName): socServer3
Member Weight (memberWeight):
Node Group (nodeGroup):
Create Replicator Entry (replicatorEntry):
Create Server Cluster

Creates a new application server cluster.

1. Cluster Configuration (clusterConfig)
2. Replication Domain (replicationDomain)
3. Convert Server (convertServer)
-> End of task

P (Previous)
F (Finish)
C (Cancel)
H (Help)

```

```

Select [P, F, C, H]: [F] F
WASX7278I: Generated command line: $AdminTask createCluster {-clusterConfig
{{T4SCluster true}} -convertServer {{SocratesNode socServer3 "" "" ""}}}
T4SCluster(cells/PlatoCell/clusters/T4SCluster|cluster.xml#ServerCluster_109821
6719021)

```

The AdminConfig object provides 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 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 6-55 shows how to create a new server, socServer4, and make it a member of a cluster, T4SCluster, by use of the batch invocation of the **createClusterMember** command from the AdminTask.

Example 6-55 Create a new cluster member

```
wsadmin>$AdminTask createClusterMember {-clusterName T4SCluster -memberConfig
{{SocratesNode socServer4 "" "" true false}}}
socServer4(cells/PlatoCell/clusters/T4SCluster|cluster.xml#ClusterMember_109822
1876367)
```

```
wsadmin>$AdminConfig save
```

Deleting a member from a cluster

To delete a member from a cluster, use the **AdminTask deleteClusterMember** command. Example 6-56 shows how to delete a cluster member.

Example 6-56 Delete a cluster member

```
wsadmin>$AdminTask deleteClusterMember {-clusterName T4SCluster -memberNode
SocratesNode -memberName socServer4}
ADMG9239I: Cluster member socServer4 on node SocratesNode deleted from cluster
T4SCluster.
```

```
wsadmin>$AdminConfig save
```

Configuring JDBC providers

Example 6-57 on page 314 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 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 6-57, 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 6-57 Configuring a JDBC driver

```
wsadmin>set cluster [$AdminConfig getid /ServerCluster:testCluster/]
testCluster(cells/PlatoCell/clusters/testCluster|cluster.xml#ServerCluster_1098
125859682)

wsadmin>set JDBCTempl [lindex [$AdminConfig listTemplates JDBCProvider
"CLOUDSCAPE JDBC Provider (XA)"] 1]
Cloudscape JDBC Provider
(XA)(templates/servertypes/APPLICATION_SERVER/servers/default|resources.xml#bu
1tin_jdbcprovider)

wsadmin>$AdminConfig createUsingTemplate JDBCProvider $cluster {{name
testDriver}} $JDBCTempl
testDriver(cells/PlatoCell/clusters/testCluster|resources.xml#JDBCProvider_1098
203064998)

wsadmin>$AdminConfig save
```

6.5 Differences from WebSphere V5

WebSphere Application Server V6 implements Java Management Extensions (JMX) Version 1.2, while WebSphere Application Server V5 implements JMX Version 1.0.

Due to the evolution of the JMX specification, the serialization format for JMX objects, such as the `javax.management.ObjectName` object, differs between the V5 implementation and the V6 implementation. The V6 JMX run time is enhanced to be aware of the version of the client with which it is communicating. The V6 run time makes appropriate transformations on these incompatible serialized formats to support communication between the different version run times. A V5 `wsadmin` script or a V5 administrative client can call a V6 deployment manager, node, or server. A V6 `wsadmin` script or a V6 administrative client can call a V5 node or server.

When a V5 `wsadmin` script or a V5 administrative client calls a V6 MBean, the instances of classes that are new in V6 cannot be passed back to V5 because these classes are not present in the V5 environment. The problem occurs infrequently. However, it usually occurs when an exception embeds a nested exception that is new in V6. The symptom is usually a serialization exception or a `NoClassDefFoundException` exception.

Due to changes in the JMX implementation from V5 to V6, different exceptions are created when a method on an MBean is invoked for V5 than when a method on an MBean is invoked for V6. For example, when a method gets or sets an unknown attribute for V5, the `MBeanRuntimeException` exception is created. When a method gets or sets an unknown attribute for V6, the `MBeanException` exception that wraps a `ServiceNotFoundException` exception is created.

An instance of a user-defined class that implements the `Serializable` interface that is passed as a parameter or return value during MBean invocation, or sent as part of a notification, cannot contain a non-transient instance variable that is in the `javax.management.package` package. If the instance does, it cannot be properly deserialized when passed between V5 and V6 run times.

Due to changes in the supported format for the `ObjectName` class from V5 to V6, the configuration ID in V6 contains a vertical bar (|), but in V5, the ID contains a colon (:). This change is reflected in the output for `wsadmin` clients. For example, for a V5 client, the output is:

```
wsadmin>$AdminConfig list Cell  
DefaultCellNetwork(cells/DefaultCellNetwork:cell.xml#Cell_1)
```

For a V6 client, the output is:

```
wsadmin>$AdminConfig list Cell
DefaultCellNetwork(cells/DefaultCellNetwork|cell.xml#Cell_1)
```

The change to the configuration ID is not usually a problem because configuration IDs are generated dynamically. When a V5 client passes a configuration ID that contains a colon, the JMX run time, for upward compatibility, automatically transforms the configuration ID with a colon into a configuration ID with a vertical bar. Similarly, a reverse transformation is performed for backward compatibility.

Do not save the configuration ID and then try to use it later. Only query the ID and use it.

6.6 End-to-end examples

Examples of installing and configuring applications and resources, can be found in the Samples gallery, installed as part of an application server profile. The gallery manages installation and configuration by use of Jacl scripts. See the Information Center topic *Accessing the Samples (Samples Gallery)* for information about the Gallery samples.

6.7 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/infocenter.html>
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-106.ibm.com/developerworks/websphere/library/samples/SampleScripts.html>
- ▶ Tcl Developer Xchange
<http://www.tcl.tk/>
- ▶ IBM WebSphere Developer Technical Journal
<http://www-106.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:

- ▶ 7.2, “JDBC resources” on page 321
- ▶ 7.3, “JCA resources” on page 341
- ▶ 7.4, “JavaMail resources” on page 354
- ▶ 7.5, “URL providers” on page 364
- ▶ 7.6, “Resource environment providers” on page 369
- ▶ 7.7, “Resource authentication” on page 374

7.1 WebSphere resources

WebSphere Application Server provides a number of resources you can define for applications to use. The resource types can be seen in the administrative console under the Resources category, as in Figure 7-1.

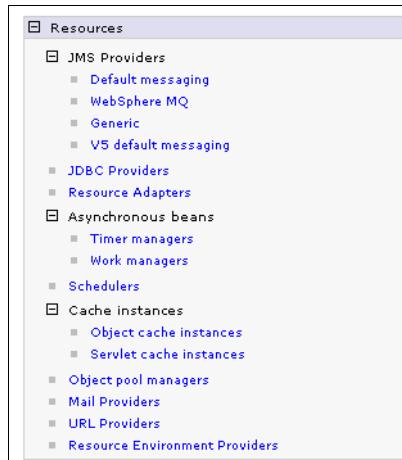


Figure 7-1 WebSphere Application Server resource types

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 10, “Asynchronous messaging” on page 463.

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 book. Information on 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

7.2 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
- ▶ Third-party vendor JDBC data source using SQL99 standards

7.2.1 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 hard code 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.

Note: 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.

7.2.2 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 7-2.

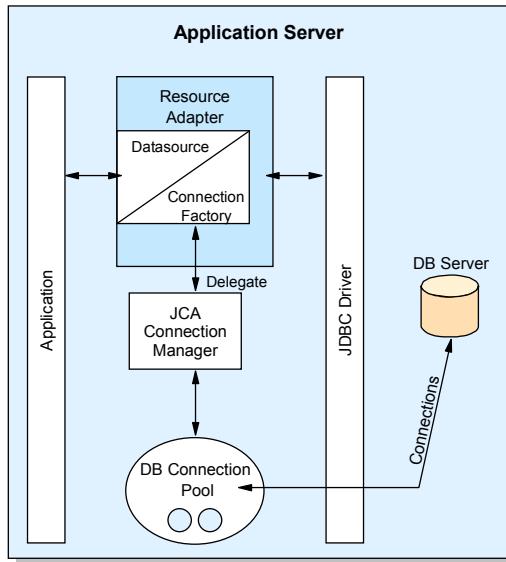


Figure 7-2 Resource adapter in J2EE connector architecture

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 7-3 on page 324 shows the relational resource adapter model.

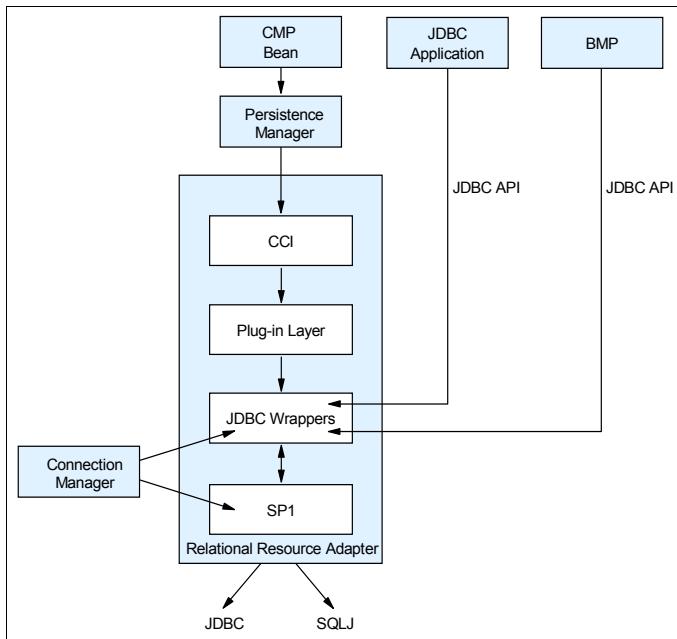


Figure 7-3 Persistence resource adapter model

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.

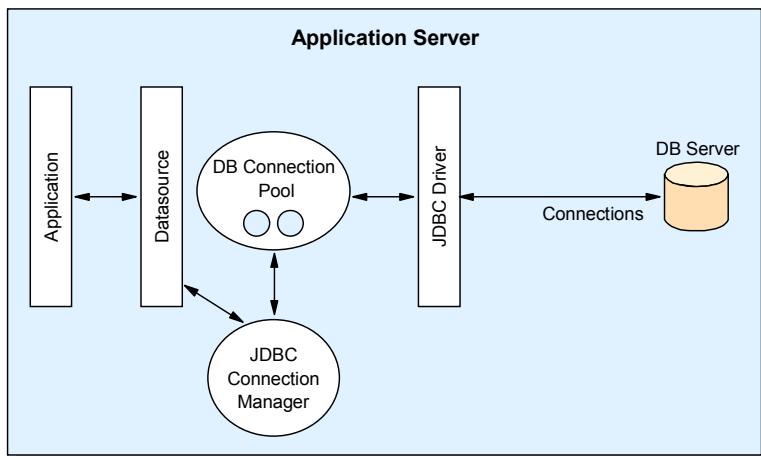


Figure 7-4 Connection pooling in WebSphere Application Server Version 4

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.

7.2.3 Creating a data source

The following steps are involved in creating a data source:

1. Create a JDBC provider resource.

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 resource.

The JDBC data source encapsulates the database-specific connection settings.

7.2.4 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 **Scope** and click **Apply**.

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 7-5 on page 327, you can see that there are four JDBC providers defined at the server level.

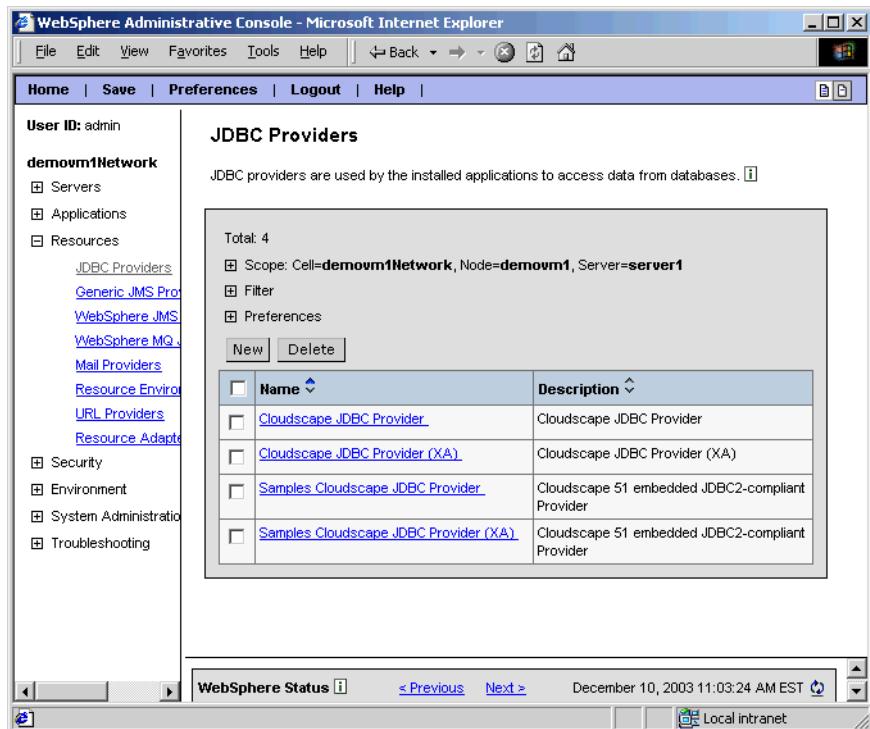


Figure 7-5 JDBC providers

4. Select **New** to create a new JDBC provider.

5. Use the list boxes to select the type of provider you want to create. See Figure 7-6.

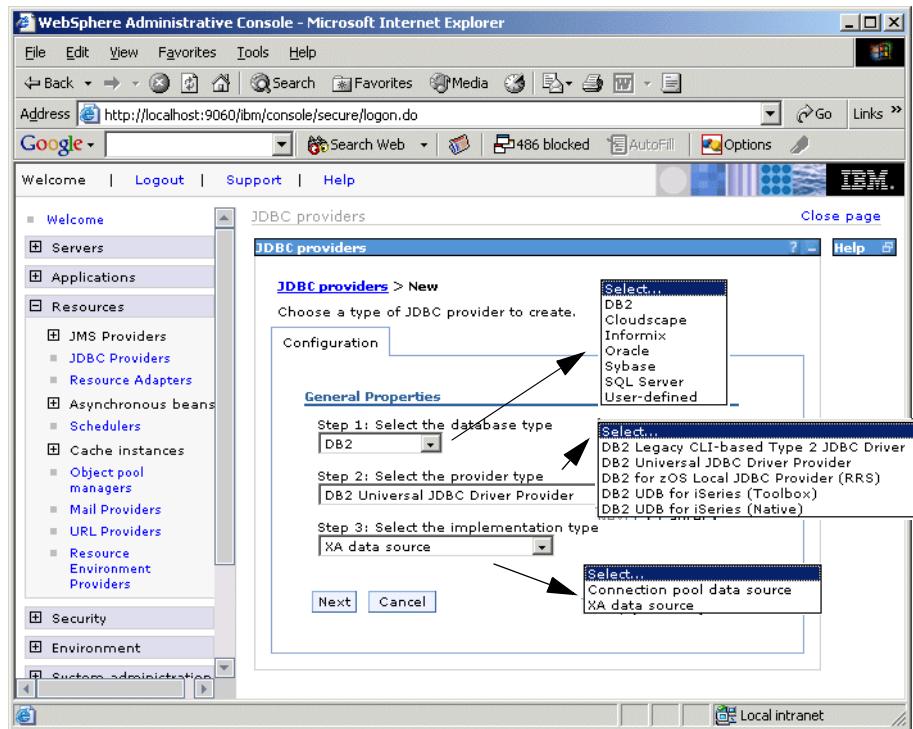


Figure 7-6 Define a new JDBC provider: Panel 1

- 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.

6. Click **Next**. The settings page for your JDBC provider appears. Figure 7-7 on page 329 shows the configuration page for a DB2 JDBC Provider.

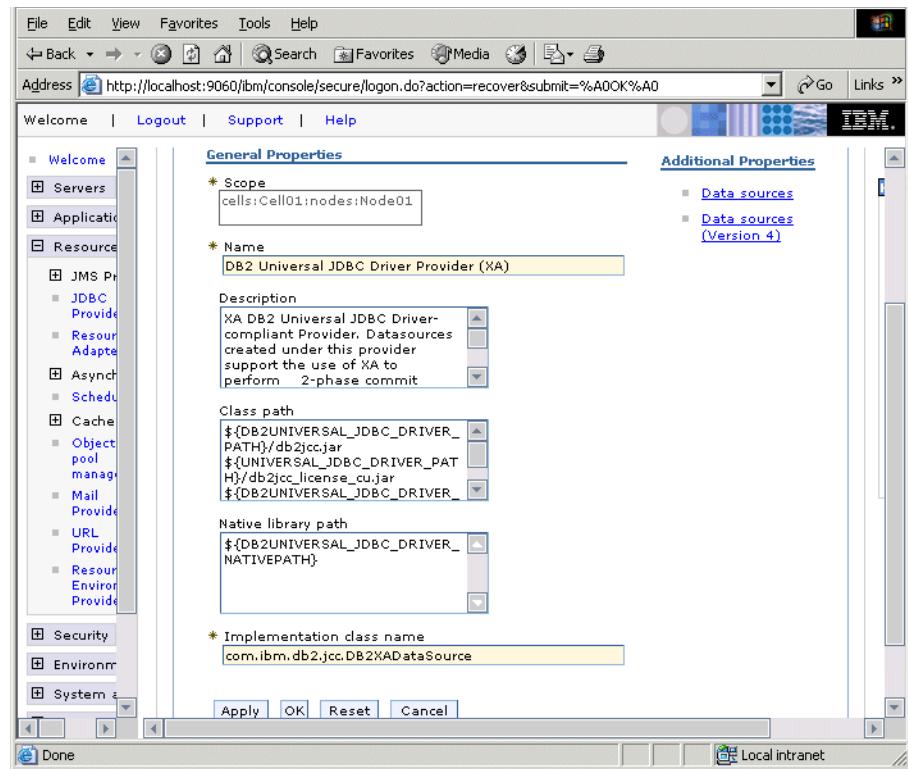


Figure 7-7 Define a new JDBC provider: Panel 2

Enter the JDBC provider properties.

– Name

Enter a name for the provider. It is good practice to enter a name that is suggestive of the database product you are using.

– Classpath

This field is a list of paths or JAR file names which 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 the Enter between each one

– 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.

- Implementation class name

The name of the Java data source class used to connect to the database, as provided by the database vendor. This is provided for you when you select the type of JDBC provider.

This class must be available in the driver file specified by the Classpath property.

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, 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 5.1.10, “Using variables” on page 179 for more information about WebSphere environment variables.

7. After verifying the settings, click **Apply**. 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 7.2.5, “Creating JDBC data source” on page 331. 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.

7.2.5 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** from the navigation tree.
2. Click **JDBC Providers**.
3. Select **Scope** and click **Apply**.
4. Select the JDBC provider to be used by your data source.
5. Select **Data Sources** under the Additional Properties section.
6. Click **New** to create a new data source, or select an existing one to modify the data source properties.

The configuration panel for the data source contains general property settings, including those that you see in Figure 7-8 on page 332, plus security settings shown in Figure 7-9 on page 333, and the data source-specific properties shown in Figure 7-10 on page 334.

The links to the additional and related properties shown on the right of Figure 7-8 on page 332 will be enabled once you complete the required settings and click **Apply**.

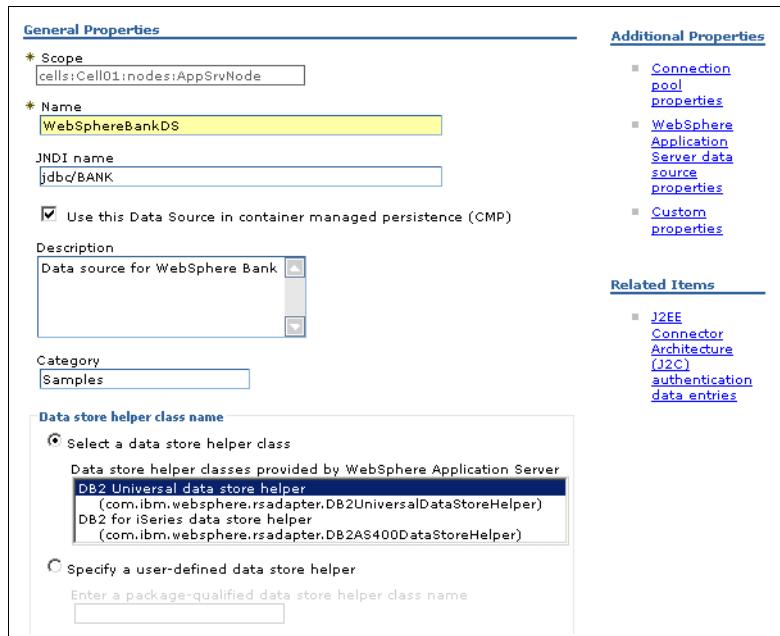


Figure 7-8 Data source general properties

► 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>`.

► 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** → **WebSphere Relational Resource Adapter** → **CMP Connection Factories**. Be sure to set the scope so it is the same as that for the data source.

► **Datasource Helper Classname**

This field specifies the data store helper used to perform database-specific functions. This is used by the relational resource adapter at runtime. The default DataStoreHelper implementation class is set based on the JDBC driver implementation class, using the structure:

```
com.ibm.websphere.rsdadapter.<database>DataStoreHelper
```

For example, if the JDBC provider is DB2, then the default DataStoreHelper class is:

```
com.ibm.websphere.rsdadapter.DB2DataStoreHelper
```

You can change the value to refer to a specific subclass of this DataStoreHelper if necessary.

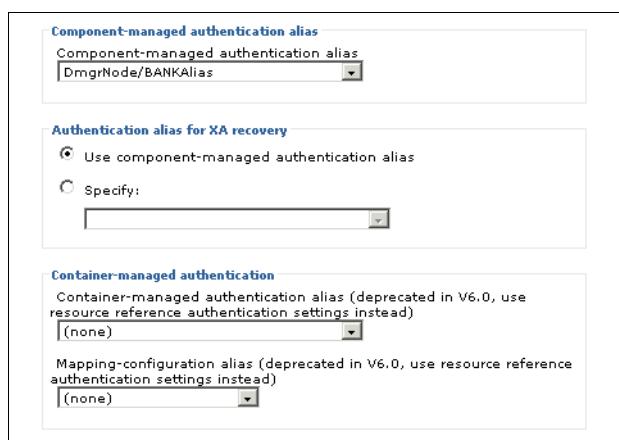


Figure 7-9 Data source security settings

The settings shown in Figure 7-9 deal with authentication to the data source. For information about how these setting are used see 7.7, “Resource authentication” on page 374.

► **Component-managed authentication alias**

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 panel. See Figure 7-8 on page 332.

- ▶ Authentication alias for XA recovery
This optional field is used to specify the authentication alias that should be used during XA recovery processing. You can use the same J2C authentication entry as specified for the component-managed authentication alias, or you can specify another J2C authentication entry.
- ▶ Container-managed Authentication Alias (deprecated)
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. This setting has been deprecated in V6.
- ▶ Mapping-Configuration Alias (deprecated)
Through this selection, allows users to select from the **Security → JAAS Configuration → Application Logins Configuration** list. The DefaultPrincipalMapping JAAS configuration maps the authentication alias to the user ID and password. You can define and use other mapping configurations.

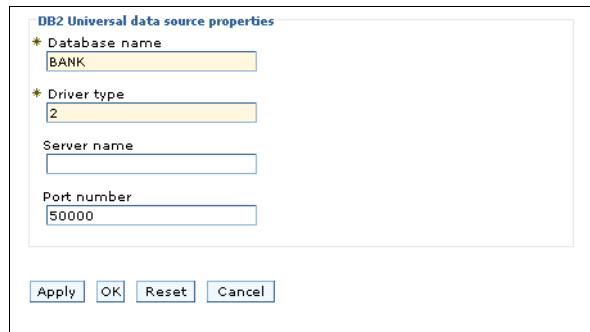


Figure 7-10 Data source-specific properties

The last settings, shown in Figure 7-10, are specific to the JDBC driver and data source type. Figure 7-10 shows the properties for the DB2 Universal data source.

In this case, the DB2 Universal data source requires the database name and driver type to be specified.

7. Click **Apply**.

The links under the Additional Properties and Related Items sections become enabled. Use the **Test Connection** button at the top of the page to make sure the connection to the data source works.

Adding or updating custom properties

To add or update custom properties, do the following:

1. 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.
2. Click **New** to add a custom property, or click a property name to modify it.

Figure 7-11 shows the first few custom properties configured for a data source connecting to a DB2 database.

Select	Name	Value	Description	Required
<input type="checkbox"/>	description		The description of this datasource.	false
<input type="checkbox"/>	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_PARAMETER_META_DATA = 128,TRACE_RESULT_SET_META_DATA = 256,TRACE_DIAGNOSTICS = 512,TRACE_SQLJ = 1024,TRACE_ALL = -1.	false
<input type="checkbox"/>	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
<input type="checkbox"/>	fullyMaterializeLobData	true	This setting controls whether or not LOB locators are used to fetch LOB data. If enabled, LOB data is not streamed, but is fully materialized with information when the user requests a stream.	false

Figure 7-11 Data Source custom properties

3. 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 panel. See Figure 7-8 on page 332.

The screenshot shows a configuration dialog for a data source connection pool. The left pane, titled 'General Properties', contains the following settings:

- Scope:** cells:Cell01:nodes:Node01
- 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

The right pane, titled 'Additional Properties', contains two links:

- Advanced connection pool properties
- Connection pool custom properties

At the bottom of the dialog are four buttons: Apply, OK, Reset, and Cancel.

Figure 7-12 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 5 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 Reap Time 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 which 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 in the Additional Properties section of the connection pool configuration page (Figure 7-12 on page 336) allows you to modify the properties shown in Figure 7-13 on page 339.

Configuration

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 7-13 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

These properties apply to the WebSphere Application Server connection, rather than to the database connection.

The WebSphere Application Server data source properties are accessed through the link under the Additional Properties section of the data source configuration page. See Figure 7-8 on page 332. Clicking the link gives you the window in Figure 7-14 on page 340.

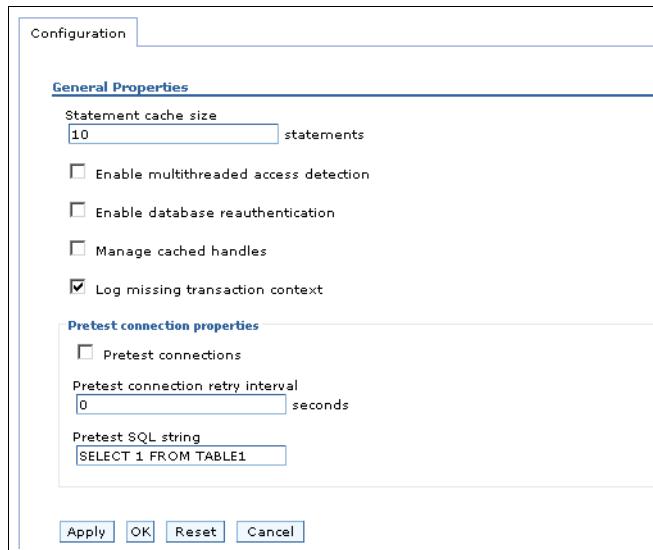


Figure 7-14 WebSphere data source custom properties

► 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 5 SQL statements, set the statement cache size to 5, so that each connection has 5 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 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.

- ▶ 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.

- ▶ Pretest connection

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.

7.3 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 legacy 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 internal 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 7-15.

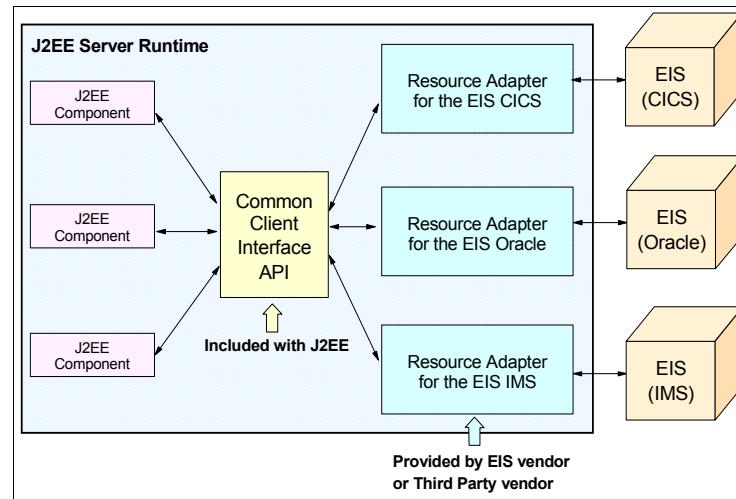


Figure 7-15 Common Client Interface API

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.

7.3.1 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 .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 runtime
- ▶ 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.

7.3.2 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.
2. Click **Resource Adapters**. The administrative console shows all the configured resource adapter objects. In Figure 7-16 on page 346 you see the two resource adapters supplied with WebSphere.

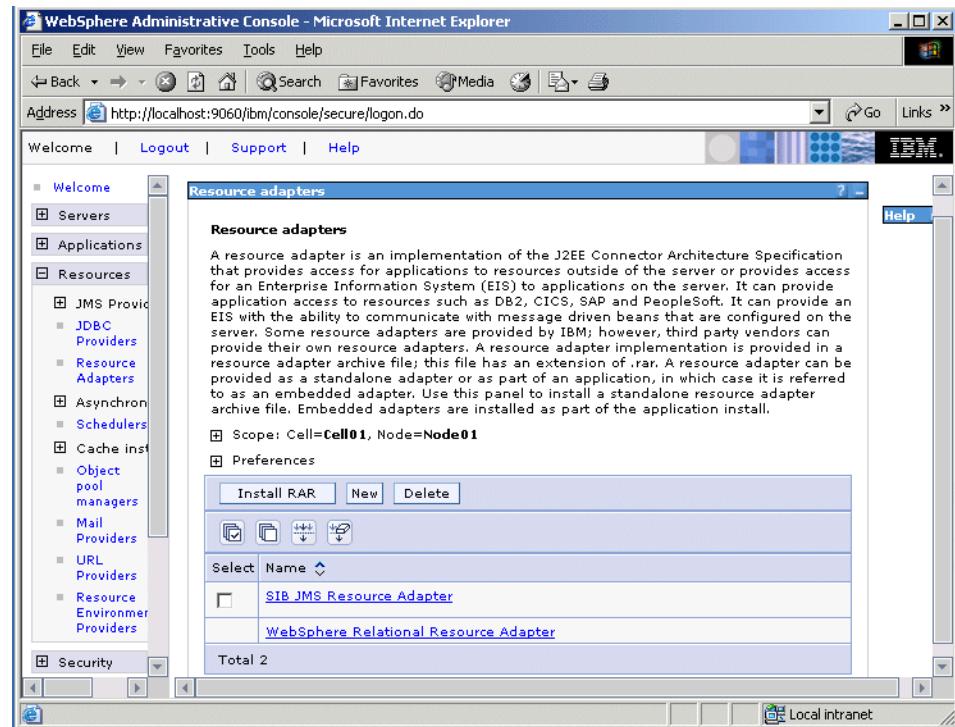


Figure 7-16 JCA resource adapters

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 7-17 on page 347.

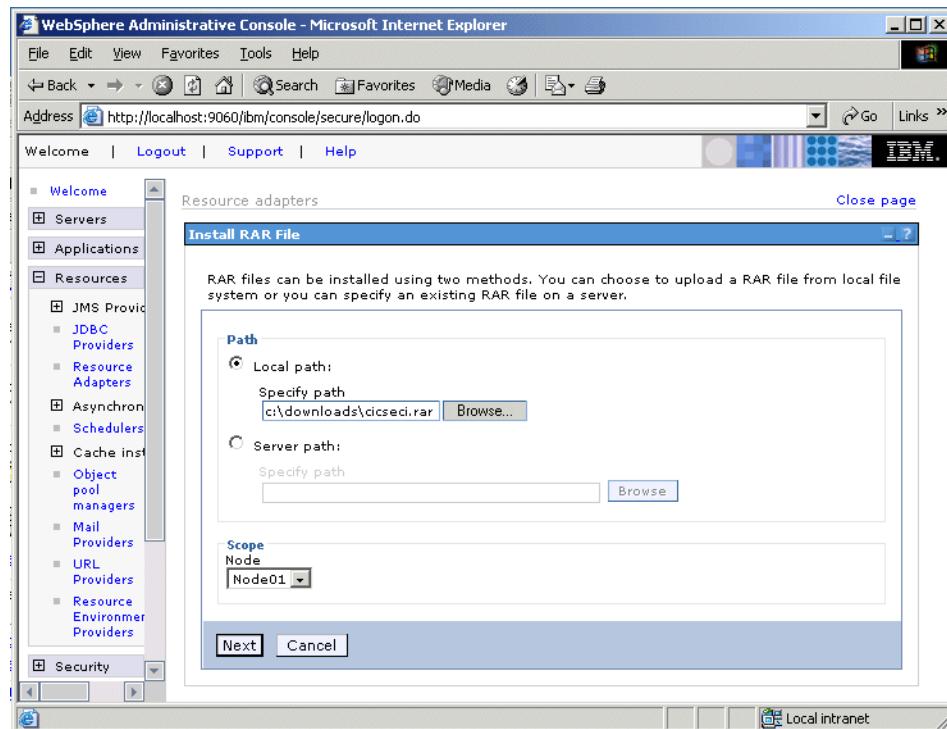


Figure 7-17 RAR file location

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 7-18 on page 348.

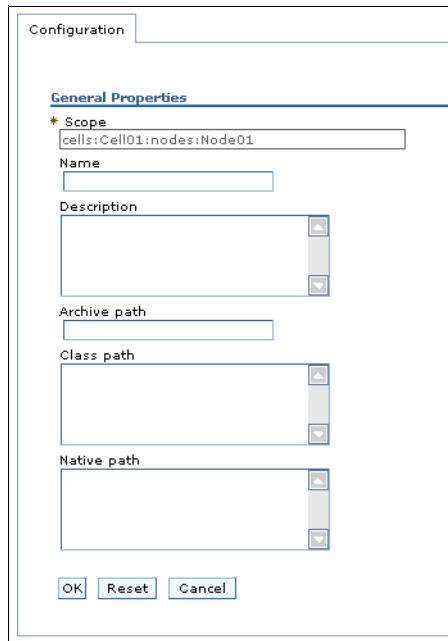


Figure 7-18 JCA resource adapter properties

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 which 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**.

7.3.3 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 runtime and are not used by the vendor supplied resource adapter code.

To create a J2C connection factory, do the following:

1. Select the J2C resource adapter just created. You now have entries in the Additional Properties section, allowing you access to connection factories, custom properties, and the deployment descriptor (ra.xml).

See Figure 7-19 on page 350

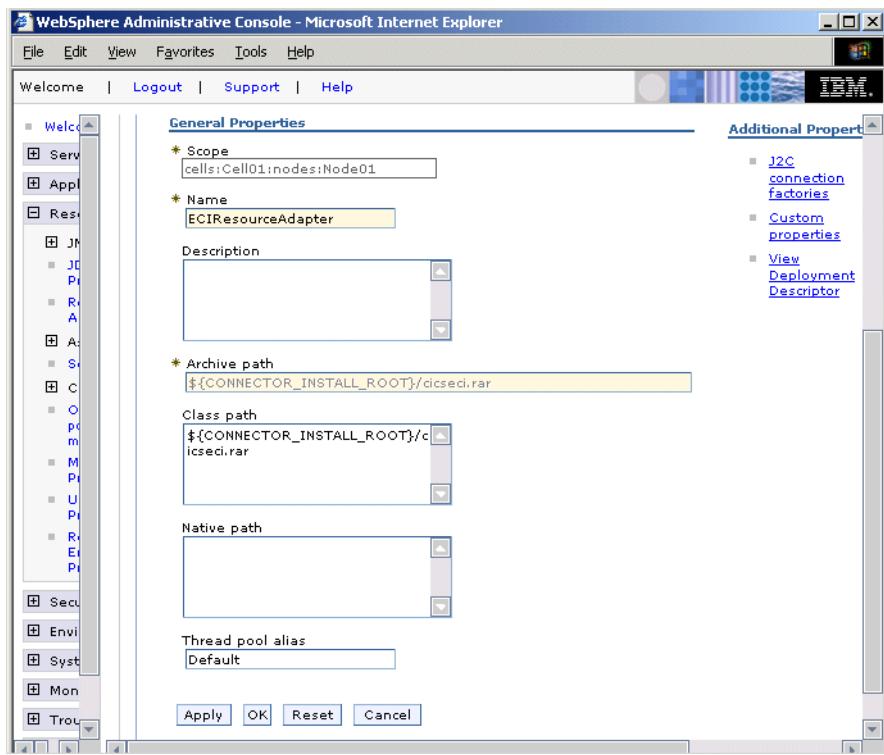


Figure 7-19 Configurable resources for the JCA connector

2. Click **J2C Connection Factories** under the Additional Properties.
3. 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 7-20 on page 351.

General Properties

- * Scope
cells:Cell01:nodes:Node01
- * Name
ECICICS
- JNDI name
eis/ECICICS
- Description

Additional Properties

- [Connection pool properties](#)
- [Advanced connection factory properties](#)
- [Custom properties](#)

Related Items

Component-managed authentication alias
Component-managed authentication alias
(none)

Container-managed authentication
Container-managed authentication alias (deprecated in V6.0, use resource reference authentication settings instead)
(none)

Authentication preference (deprecated in V6.0, use resource reference authentication settings instead)
None

Mapping-configuration alias (deprecated in V6.0, use resource reference authentication settings instead)
(none)

Buttons: Apply, OK, Reset, Cancel

Figure 7-20 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.

Note: The following security settings are deprecated in V6:

- ▶ Container managed authentication alias
- ▶ Authentication preference
- ▶ Mapping configuration alias

Resource authentication settings should be used instead. For more information, see 7.7, “Resource authentication” on page 374.

4. 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 335.

The advanced connection factory properties are shown in Figure 7-21 on page 353.

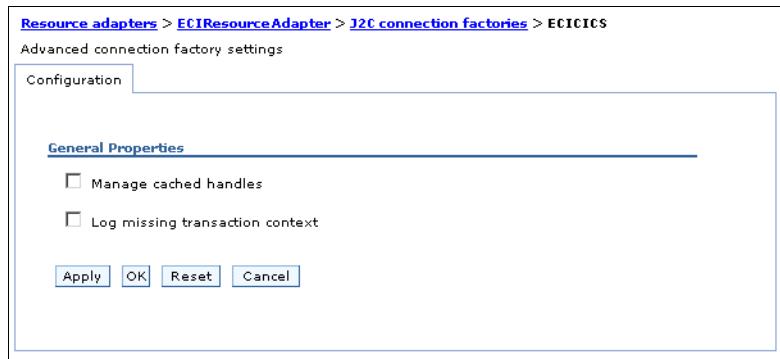


Figure 7-21 Advanced connection factory properties

- ▶ 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.

7.3.4 Using resource adapters from an application

Example 7-1 shows how you might access the CICS ECI resource adapter we just installed 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 JNDI name `eis/ECICICS`. It is a minimal sample, with no connection factory caching, and so on.

Example 7-1 Using resource adapters from an application. Code sample

```
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.ECIIInteractionSpec interactionSpec = new
com.ibm.connector2.cics.ECIIInteractionSpec();

// 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;

}

```

7.4 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 7-22 on page 356 illustrates the relationship among the different JavaMail components.

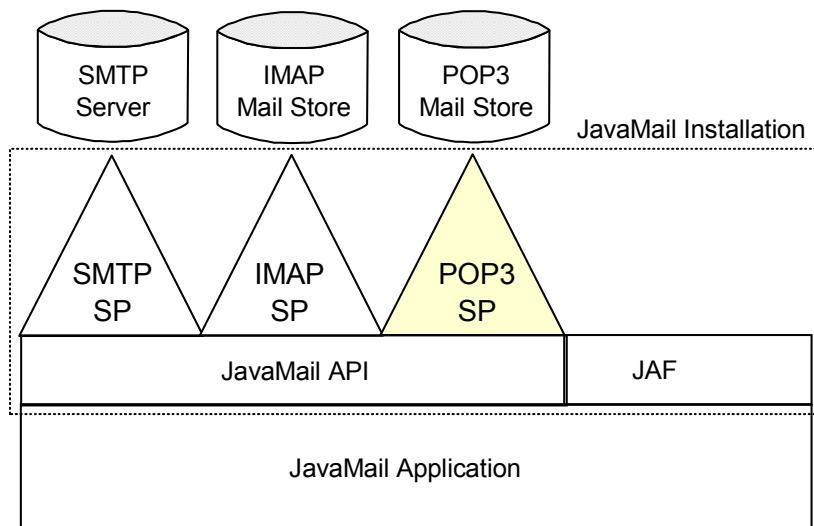


Figure 7-22 JavaMail components

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.

7.4.1 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.

7.4.2 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.
2. Click **Mail Providers**.
3. 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 7-23, shows the mail provider installed with WebSphere. The built-in mail provider is available to all the application servers in the cell.

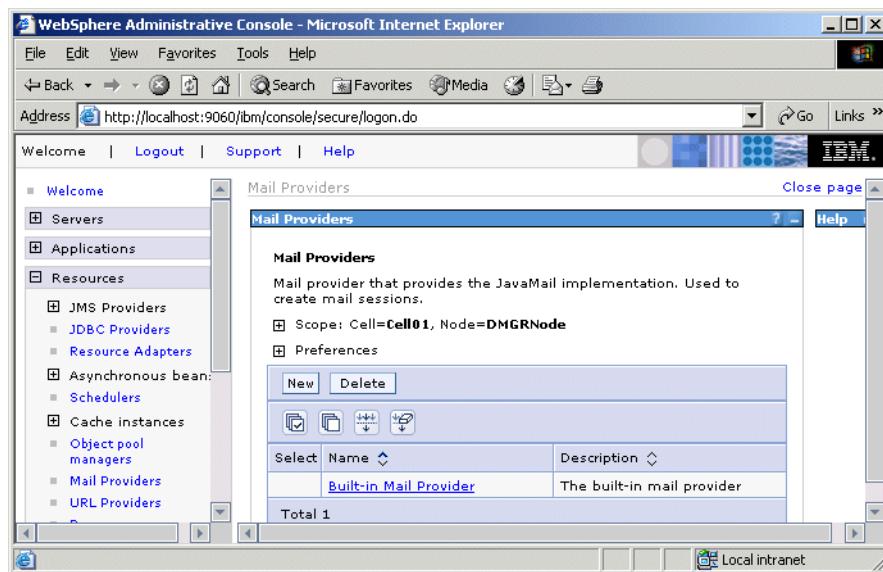


Figure 7-23 Mail provider page

4. Click **New** to configure a new mail provider.
5. Enter a name and a description, then click **Apply**. The properties required to configure a new mail provider are shown in Figure 7-24 on page 358.

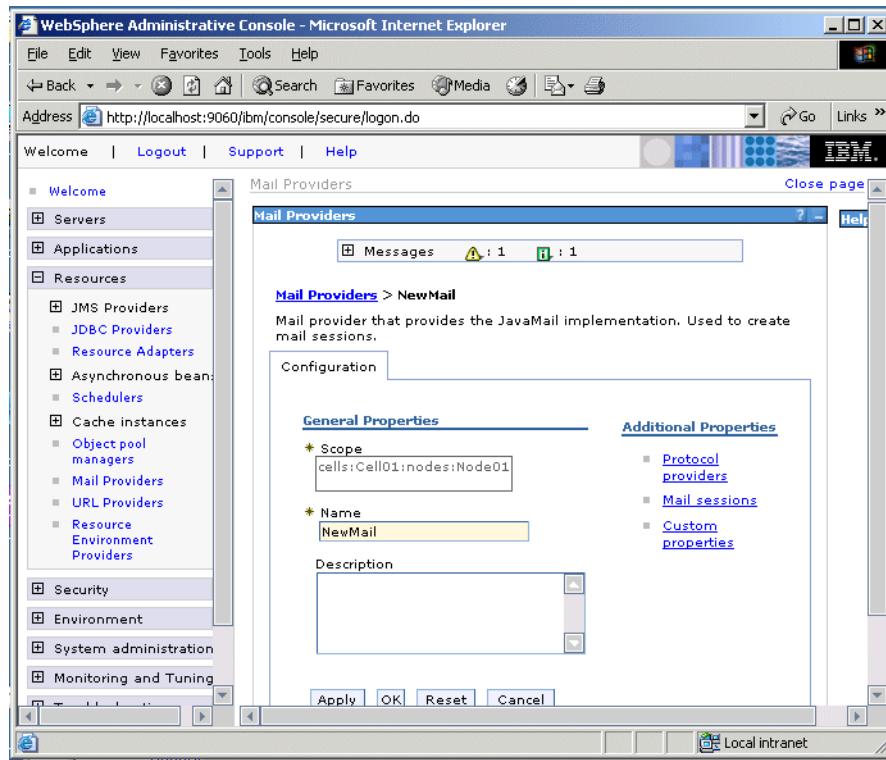


Figure 7-24 Mail Provider general properties

6. Click **Protocol Providers** under the Additional Properties section.
7. Click **New** to add a protocol provider.

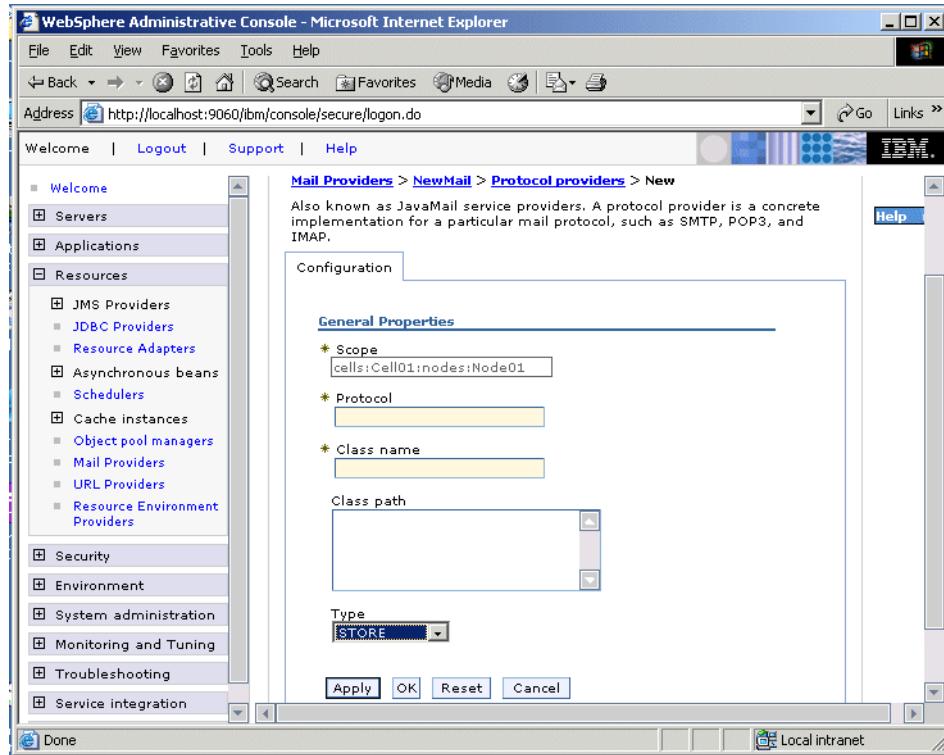


Figure 7-25 Protocol provider configuration page

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, shown in Figure 7-26.

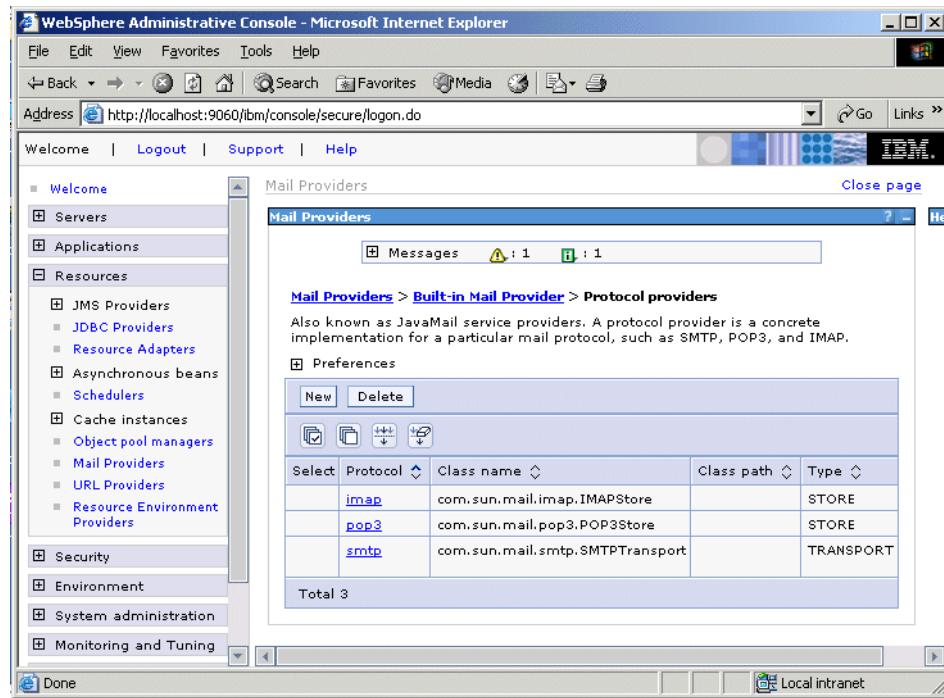


Figure 7-26 Protocol providers

8. Click **OK** and save the configuration.

7.4.3 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 7-24 on page 358.
6. Select **New** to create a new mail session object. Figure 7-27 on page 361 shows the configuration page for the PlantsByWebSphere sample application.

Mail Providers

[Mail Providers](#) > [Built-in Mail Provider](#) > [Mail Sessions](#) > **PlantsByWebSphere Mail Session**

Configurations for mail support. Define protocol providers before configuring mail sessions.

Configuration

General Properties

* Scope
cells:CARLAVM2Node01Cell:nodes:AppSrvNode01

* Name
PlantsByWebSphere Mail Ses

* JNDI name
mail/PlantsByWebSphere

Description

Category

Mail transport host
yourcompany.ComOrNet

Mail transport protocol
smtp

Mail transport user ID

Mail transport password

Enable strict Internet address parsing

Mail from
userid@yourcompany.ComOr

Mail store host

Mail store protocol
pop3

Mail store user ID

Mail store password

Enable debug mode

Additional Properties

Custom properties

Buttons

Apply OK Reset Cancel

The screenshot shows the 'Mail Providers' configuration interface. At the top, the path is shown: Mail Providers > Built-in Mail Provider > Mail Sessions > PlantsByWebSphere Mail Session. Below this, a note says 'Configurations for mail support. Define protocol providers before configuring mail sessions.' A 'Configuration' tab is selected. The main area is divided into 'General Properties' and 'Additional Properties'. Under General Properties, there are fields for Scope (cells:CARLAVM2Node01Cell:nodes:AppSrvNode01), Name (PlantsByWebSphere Mail Ses), and JNDI name (mail/PlantsByWebSphere). There is also a large Description text area, a Category dropdown, and fields for Mail transport host (yourcompany.ComOrNet) and Mail transport protocol (smtp). Below these are fields for Mail transport user ID and Mail transport password. A checkbox for 'Enable strict Internet address parsing' is checked. Under Mail from, the value is userid@yourcompany.ComOr. There are also fields for Mail store host, Mail store protocol (pop3), Mail store user ID, and Mail store password. A checkbox for 'Enable debug mode' is unchecked. At the bottom, there are buttons for Apply, OK, Reset, and Cancel.

Figure 7-27 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 userid

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 userid

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.

7.4.4 Example code

The code segment shown in Example 7-2 illustrates how an application component sends a message and saves it to the Sent folder.

Example 7-2 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});
```

7.5 URL providers

A URL provider implements the functionality for a particular URL protocol, such as HTTP, by extending the `java.net.URLStreamHandler` and `java.netURLConnection` 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.

7.5.1 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.
2. Click **URL Providers**.
3. Select **Scope** and click **Apply**.
4. Click **New** to configure a new URL provider, or select an existing one to edit it.

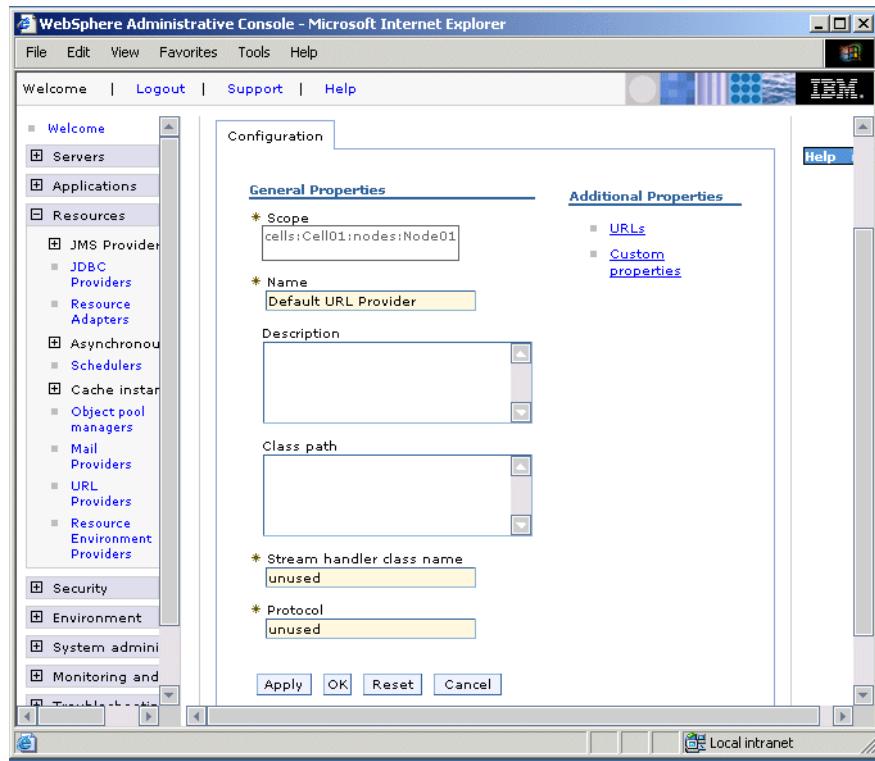


Figure 7-28 URL provider configuration page

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.
5. 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.

7.5.2 Configuring URLs

To configure a URL administrative object, do the following from the administrative console:

1. Expand **Resources** from the navigation tree.
2. Click **URL Providers**.
3. Select **Scope** and click **Apply**.
4. Select the URL provider that implements the protocol required to access the URL resource.
5. Select **URLs** under Additional Properties. Select **New**. See Figure 7-29 on page 367.

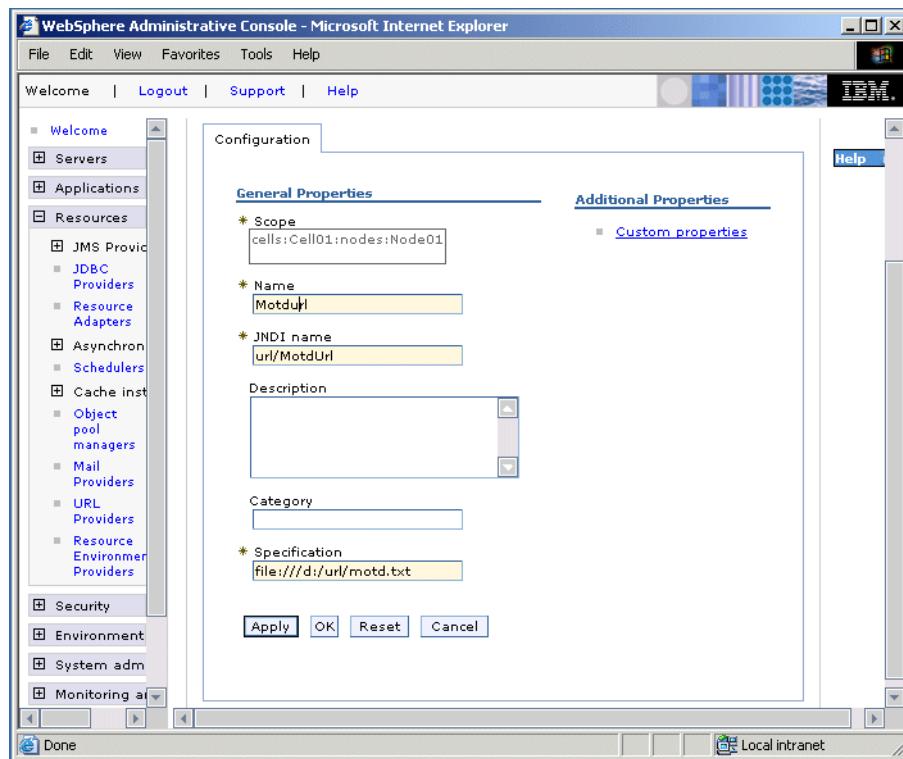


Figure 7-29 Defining URLs

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.

6. Click **OK** and save the configuration.

7.5.3 URL provider sample

Example 7-3 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 7-3 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 7-3 code into a JSP, added the JSP to a Web module, added a URL resource reference to the Web module, 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 7-4. 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 7-4 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.

7.6 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 7-5 shows a resource environment entry defined in an application's Web module deployment descriptor, `web.xml`.

Example 7-5 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 7-6 shows how this resource environment entry could be accessed from Java code in the Web module.

Example 7-6 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);
.....
```

7.6.1 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 7-7.

Example 7-7 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 7-8.

Example 7-8 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>
```

7.6.2 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 and click **Apply**.
3. Click **New**.
4. Enter a name and description for the new resource environment provider.
5. Click **Apply**. See Figure 7-30.

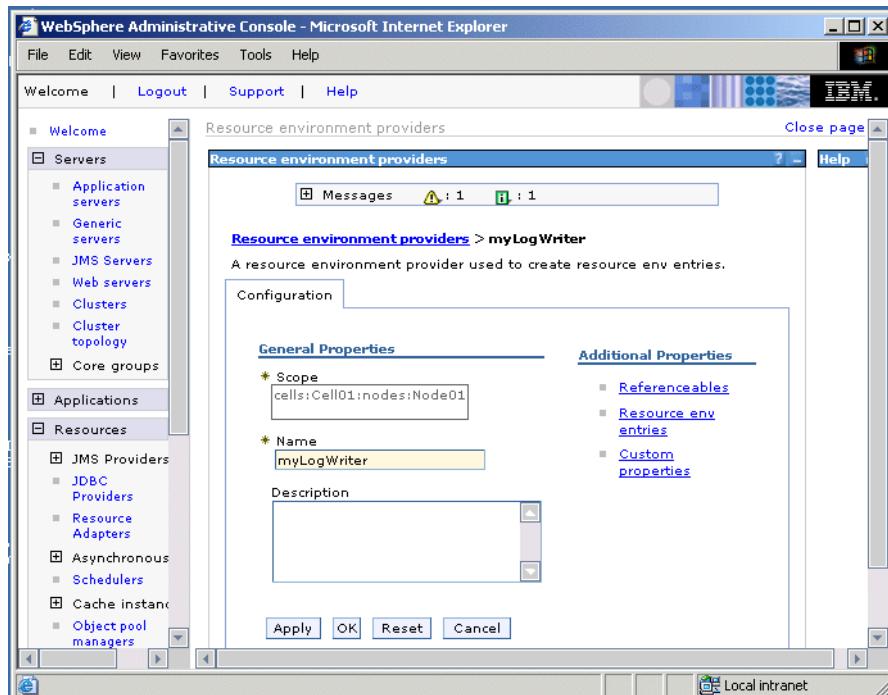


Figure 7-30 Creating a resource environment provider

6. Click **Referenceables** in the Additional Properties section.

7. 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 7-31.

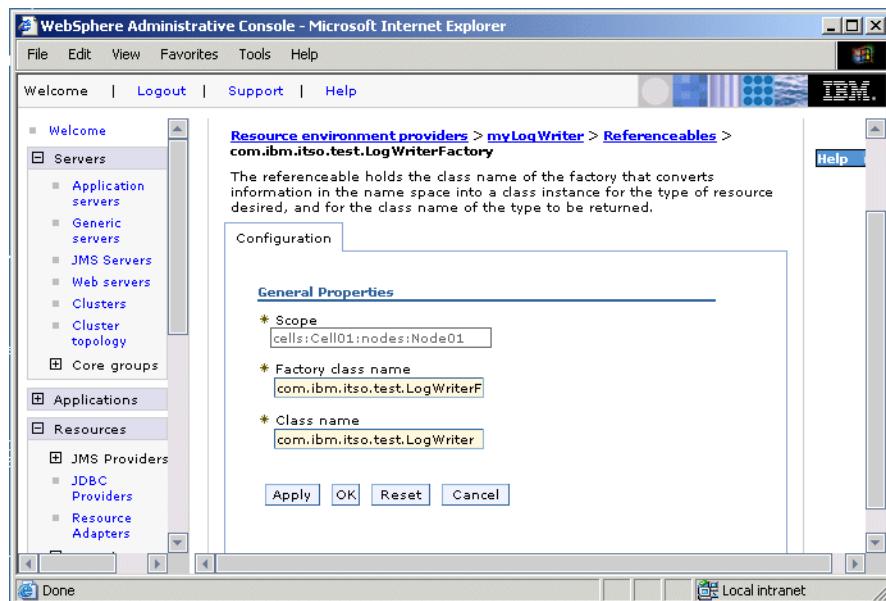


Figure 7-31 Create a reference

- 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.
8. Click **OK**.
 9. Select the resource environment provider (in the top navigation path) and click **Resource Env Entries** under Additional Properties.
 10. Click **New**. See Figure 7-32 on page 373.

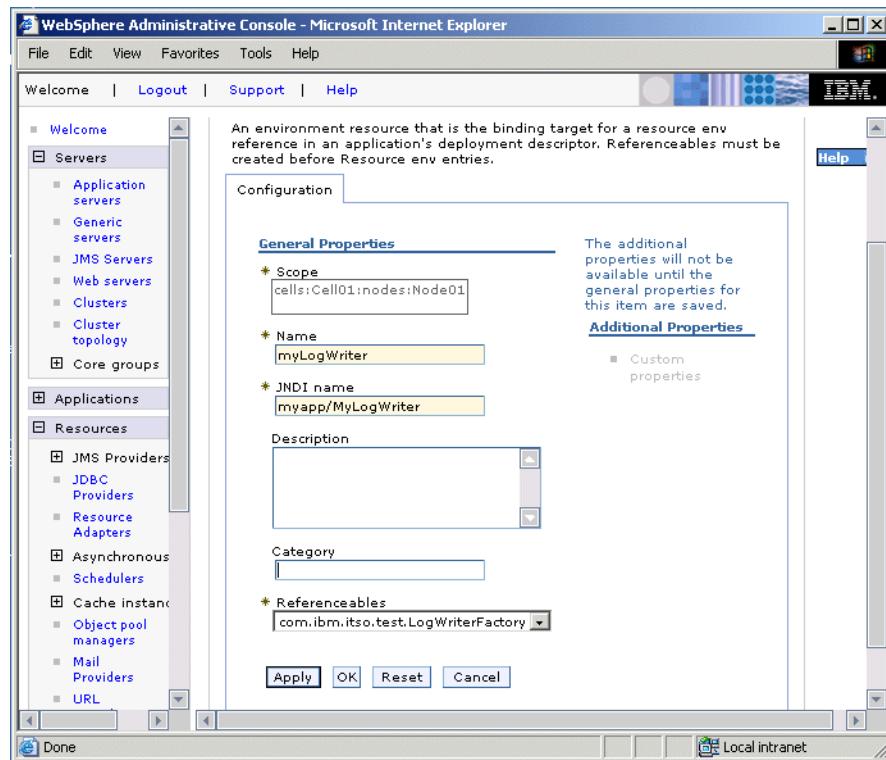


Figure 7-32 Creating a resource environment entry

– 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.

11. Click **OK**.

12. Save your configuration.

7.7 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 Security V6 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 7-1 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. Lookup 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.

7.8 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:

- ▶ 8.1, “Web server support overview” on page 378
- ▶ 8.2, “Web server installation examples” on page 383
- ▶ 8.3, “Working with Web servers” on page 389
- ▶ 8.3.5, “Mapping modules to servers” on page 402
- ▶ 8.4, “Working with the plug-in configuration file” on page 404

For information regarding the topology of the Web server installation, refer to section 8.4, *Planning and Designing for WebSphere Application Server V6*, SG24-6446.

8.1 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.

New in WebSphere Application Server V6: These features are new with this release.

- ▶ Web servers are defined to WebSphere Application Server. A Web server resides on a managed or unmanaged node.

If located on a managed node in a distributed server environment only, a node agent is installed on the Web server system and belongs to a WebSphere Application Server administrative cell. The administrative tools communicate with the Web server through the node agent.

If located on an unmanaged node, the Web server is defined to the cell, but does not have a node agent running to manage the process.

In either case, the Web server definition allows you to generate the plug-in configuration file for the Web server.

- ▶ Web applications can be mapped to Web servers. This mapping is used to generate routing information during plug-in configuration generation.
- ▶ The Web server plug-in installation provides a wizard that takes you through the installation process. The wizard defines and installs the plug-in, configures the Web server, and defines the Web server to WebSphere. Depending on whether the Web server is local or remote to the application server, you can perform the Web server definition by script or automatically with the wizard.
- ▶ IBM HTTP Server (IHS) V6 is bundled with WebSphere Application Server V6. The administrative functionality is integrated into WebSphere Application Server to provide remote administration through the administrative console. This enhanced administrative function is only available to the IBM HTTP Server.

The following are the supported Web servers for WebSphere Application Server V6:

- ▶ 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>

8.1.1 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 8-1 on page 380.

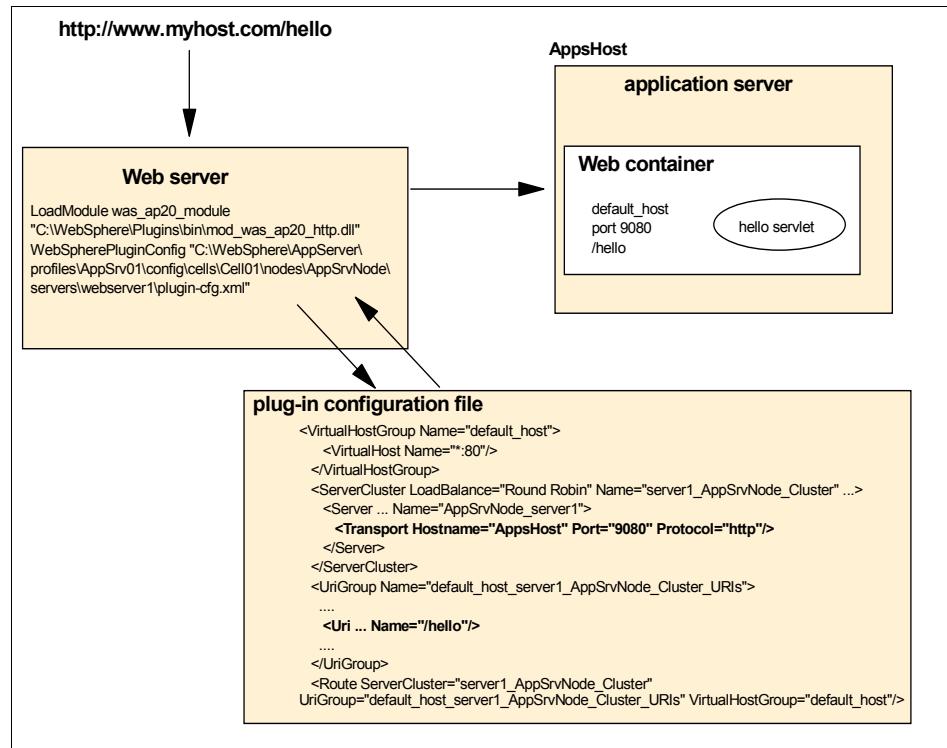


Figure 8-1 Web server plug-in routing

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.

8.1.2 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. Each Web server is associated with a node, either managed or unmanaged. The Web server definition is configured as part of the plug-in installation process. The Web server definition is also used 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.

Web server definitions are located under **Servers** → **Web servers** in the administrative console. See Figure 8-2.

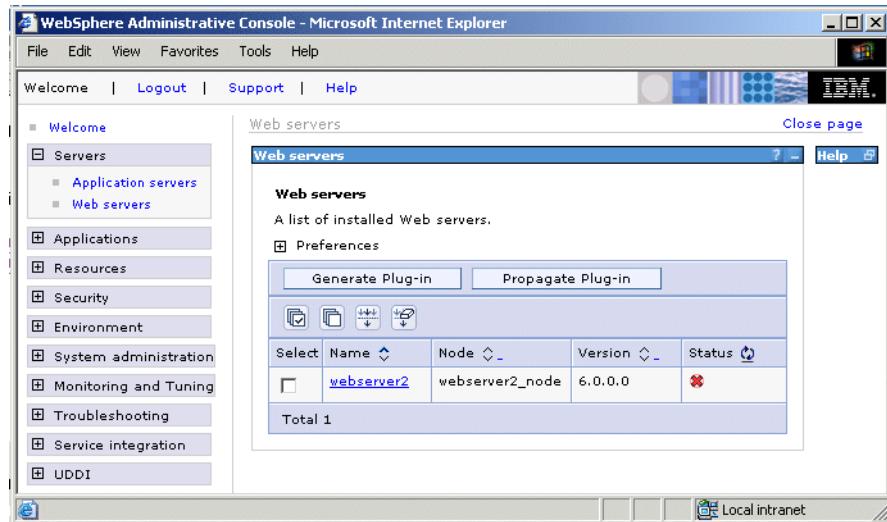


Figure 8-2 Web server definition

Contrasting Managed and 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 in 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 instance, 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 standalone 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, 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, 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 managed node, 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>/Plugins/bin/crossPlatformScripts` directory.

Note: Always open a new command window in which to execute the `configureWeb_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 need to create a Web server definition for a distributed server environment, you must federate your standalone application servers to the deployment manager first. Any Web server definitions created for a standalone application server will be lost when they are federated into a cell.

Using administrative tools: In a distributed server environment, the administrative console can also be used to define the nodes and Web servers. See 8.3.1, “Defining nodes and Web servers” on page 389.

8.2 Web server installation examples

The options for defining and managing Web servers depend on your chosen Web server topology and your WebSphere Application Server package. Decisions to make include whether to collocate the Web server with other WebSphere Application Server processes, and whether to make the Web server managed or unmanaged.

The following examples outline the process required to create each sample topology. Note that each example assumes that only the WebSphere processes shown in the diagrams are installed on each system and that the profile for the process is the default profile.

This is not a substitute for using the product documentation, rather it is intended to help you understand the process. For detailed information about how the Plug-ins installation wizard works and the logic it follows to determine how to create the configuration scripts, see the *Getting Started with Web server plug-ins* guide that comes with the plug-in.

8.2.1 Standalone server environment

In a standalone server environment, a Web server can be remote to the application server machine or local, but there can only be one defined to WebSphere Application Server. The Web server always resides on an unmanaged node.

Remote Web server

In this scenario, the application server and the Web server are on separate machines. The Web server machine can reside in the internal network, or more likely, will reside in the DMZ. See Figure 8-3.

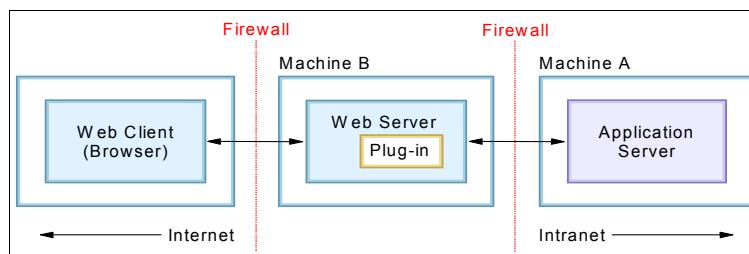


Figure 8-3 Remote Web server in a standalone server environment

Assume the application server is already installed and configured on Machine A. Perform the following tasks:

1. Install the Web server on Machine B.
2. Install the Web server plug-in on Machine B by doing the following:
 - a. Select **Remote** installation.
 - b. Enter a name for the Web server definition. The default is webserver1.
 - c. Select the location for the plug-in configuration file. By default, the location is under the config directory in the plug-in install directory. For example,

when the name specified for the Web server definition in the previous step is webserver1, the default location for the plug-in file is:

```
<plugin_home>/config/webserver1/plugin-cfg.xml
```

During installation, the following tasks are performed:

- a. Create a temporary plug-in configuration file and places it in the location specified.
- b. Update the Web server configuration file with the plug-in configuration, including the location of the plug-in configuration file.
- c. Generate a script to define the Web server to WebSphere Application Server. The script is located in:

```
<plug-in_home>/bin/configure<web_server_name>
```
3. At the end of the plug-in installation, copy the script to the `<was_home>/bin` directory of the application server machine, Machine A. Start the application server, then execute the script.
4. When the Web server is defined to WebSphere Application Server, the plug-in configuration file is generated automatically. For the IBM HTTP Server, the new plug-in file will be propagated to the Web server automatically. For other Web server types, you need to propagate the new plug-in configuration file to the Web server.

Local Web server

In this scenario, a standalone application server exists on machine A. The Web server and Web server plug-in will also be installed on machine A. This topology is suited to a development environment or for internal applications. See Figure 8-4.

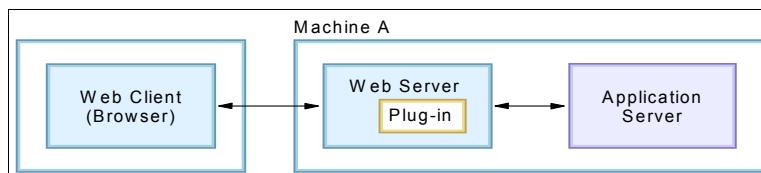


Figure 8-4 Local Web server in a standalone server environment

Assume the application server is already installed and configured. Perform the following tasks:

1. Install the Web server on Machine A.
2. Install the Web server plug-in on Machine A by doing the following:
 - a. Select **Local** installation.

- b. Enter a name for the Web server definition. The default is webserver1.
- c. Select the location for the plug-in configuration file. By default, the location under the config directory in the profile for the standalone application server will be selected. For example, when the name specified for the Web server definition in the previous step is webserver1, the default location for the plug-in file is:

```
<profile_home>/config/cells/<cell_name>/nodes/webserver1_node/servers/we  
bserver1/plugin-cfg.xml
```

Be aware that in a local scenario, the plug-in configuration file does not need to be propagated to the server when it is regenerated. The file is generated directly in the location the Web server reads it from.

During installation, the following tasks are performed:

- a. Create the plug-in configuration file and places it in the location specified.
- b. Update the Web server configuration file with the plug-in configuration, including the location of the plug-in configuration file.
- c. Update the WebSphere Application Server configuration to define the new Web server.

The plug-in configuration file is automatically generated. Because this is a local installation, you don't have to propagate the new plug-in configuration to the Web server.

8.2.2 Distributed server environment

Web servers in a distributed server environment can be local to the application server or remote. The Web server can also reside on the deployment manager system. You have the possibility of defining multiple Web servers and the Web servers can reside on managed or unmanaged nodes.

Remote Web server

The deployment manager and the Web server are on separate machines. The Web server machine can reside in the internal network, or more likely, it resides in the DMZ.

Note that this scenario and the process are almost identical to that outlined for a remote Web server in a standalone server environment. The primary difference is that the script that defines the Web server is run against the deployment manager and you will see an unmanaged node created for the Web server node. In Figure 8-5 on page 387, the node is unmanaged because there is no node agent on the Web server system.

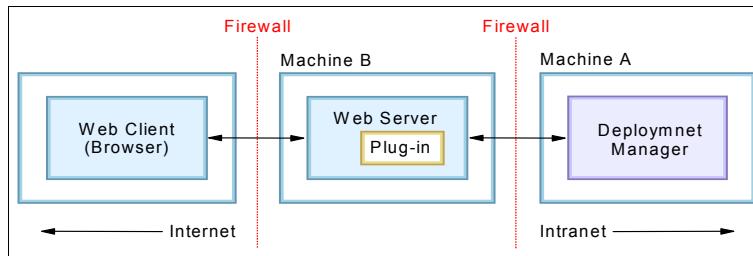


Figure 8-5 Remote Web server in a standalone server environment

Assume that the deployment manager is already installed and configured on Machine A. Perform the following tasks:

1. Install the Web server on Machine B.
2. Install the Web server plug-in on Machine B.
 - a. Select **Remote** installation.
 - b. Enter a name for the Web server definition. The default is `webserver1`.
 - c. Select the location for the plug-in configuration file. By default, the location will be under the config directory in the plug-in install directory. For example, when the name specified for the Web server definition in the previous step is `webserver1`, the default location for the plug-in file is:

`<plugin_home>/config/webserver1/plugin-cfg.xml`

During installation, the following tasks are performed:

- a. Create a temporary plug-in configuration file and places it in the location specified.
- b. Update the Web server configuration file with the plug-in configuration, including the location of the plug-in configuration file.
- c. Generate a script to define the Web server and an unmanaged node to WebSphere Application Server. The script is located in:
`<plug-in_home>/bin/configure<web_server_name>.`
3. At the end of the plug-in installation, you need to copy the script to the `<was_home>/bin` directory of the deployment manager machine (Machine A), start the deployment manager and execute the script.

When the Web server is defined to WebSphere Application Server, the plug-in configuration file is generated automatically. For the IBM HTTP Server, the new plug-in file is propagated to the Web server automatically. For other Web server types, you need to propagate the new plug-in configuration file to the Web server.

Local to a federated application server

In this scenario the Web server is installed on a machine that also has a managed node. Note that this scenario would also be the same if the deployment manager was also installed on Machine A. See Figure 8-6.

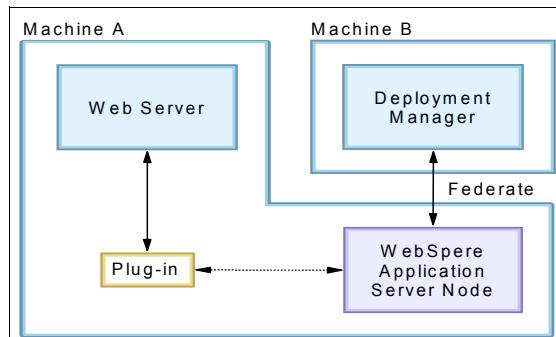


Figure 8-6 Web server installed locally on an application server system

Assume that the application server is already installed, configured and federated to the deployment manager cell. Perform the following tasks:

1. Install the Web server on Machine A.
2. Install the Web server plug-in on Machine A.
 - a. Select Local installation.
 - b. Enter a name for the Web server definition. The default is webserver1.
 - c. Select the location for the plug-in configuration file. By default, the location will be under the config directory in the profile for the application server will be selected. For example, when the name specified for the Web server definition in the previous step is webserver1, the default location for the plug-in file is:
`<profile_home>/config/cells/<cell_name>/nodes/<AppSrv_node>/servers/webs
erver1/plugin-cfg.xml`

During installation, the following tasks are performed:

- a. Create the plug-in configuration file and places it in the location specified.
- b. Update the Web server configuration file with the plug-in configuration, including the location of the plug-in configuration file.
- c. Generate a script to define the Web server and an unmanaged node to WebSphere Application Server. The script is located in:
`<plug-in_home>/bin/configure<web_server_name>.`

3. At the end of the plug-in installation, you need to execute the script to define the Web server from the location the wizard stored it in on Machine A. Make sure the deployment manager is running on Machine B. The deployment manager configuration will be updated and propagated back to Machine A at node synchronization.

The plug-in configuration file will be generated automatically and will be propagated at the next node synchronization.

8.3 Working with Web servers

With the introduction of Web server definitions to the WebSphere Application Server administrative tools, comes 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.

8.3.1 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. As you have seen, 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**. See Figure 8-7.

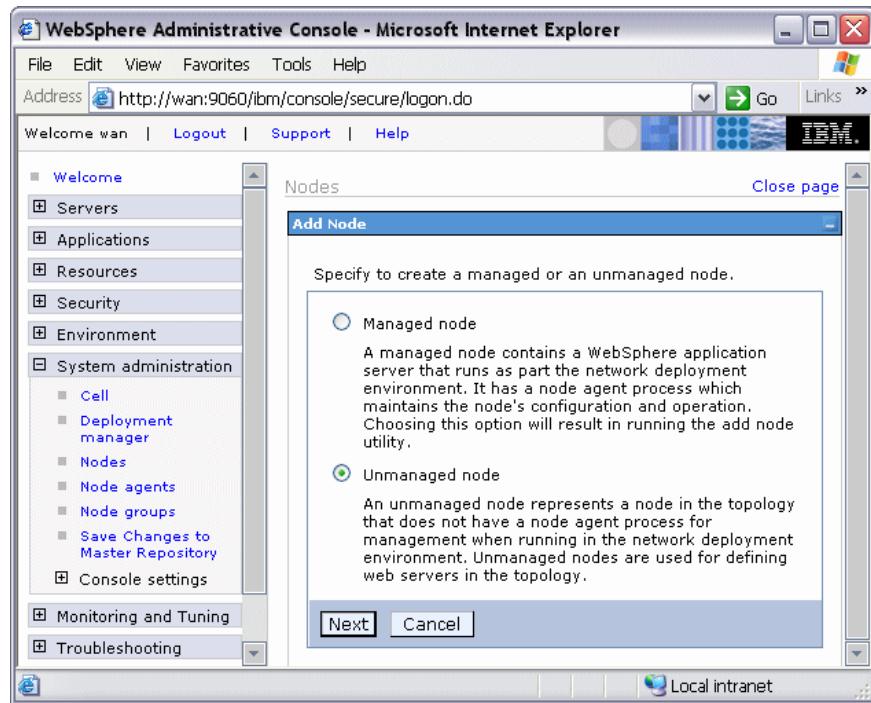


Figure 8-7 Add node page

4. Click **Next**.
5. Enter the following values in the General Properties page. See Figure 8-8 on page 391.
- 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.
 - 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

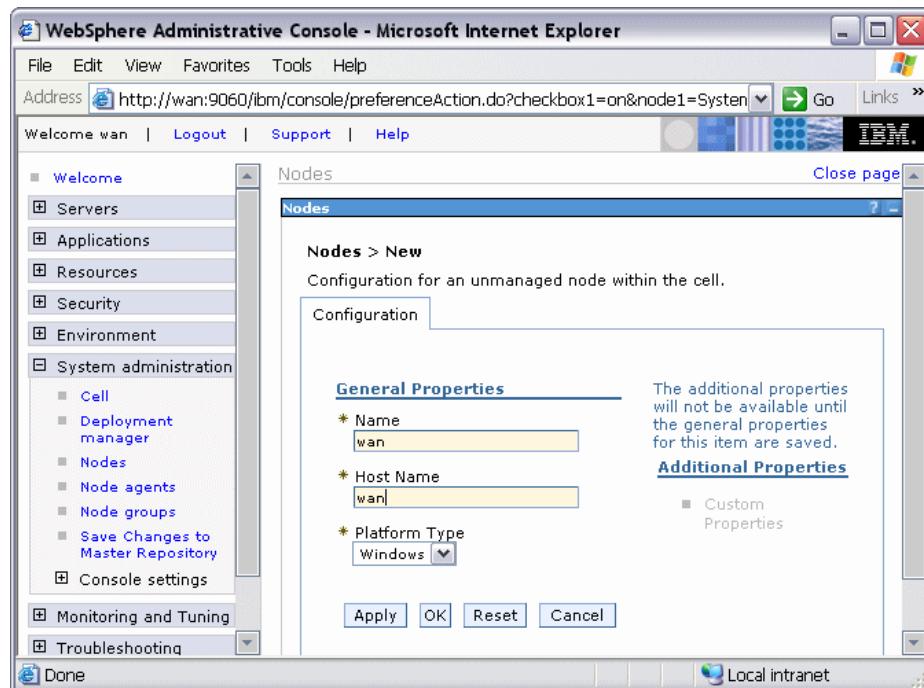


Figure 8-8 General properties for an unmanaged node

6. Click **OK**. The node is added and the name is displayed in the collection on the Nodes page. See Figure 8-9 on page 392.

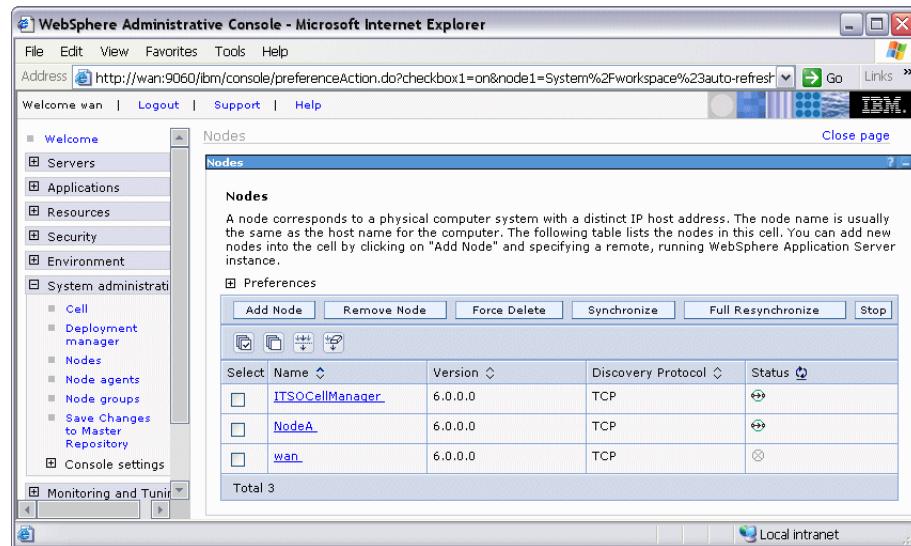


Figure 8-9 Nodes in a cell

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 8-10.
3. Select the node and enter the server name.

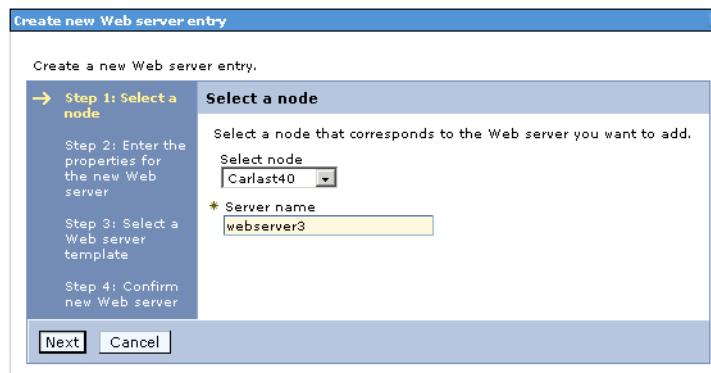


Figure 8-10 Defining a Web server: Step 1

4. Enter the properties for the Web server. See Figure 8-11 on page 393.

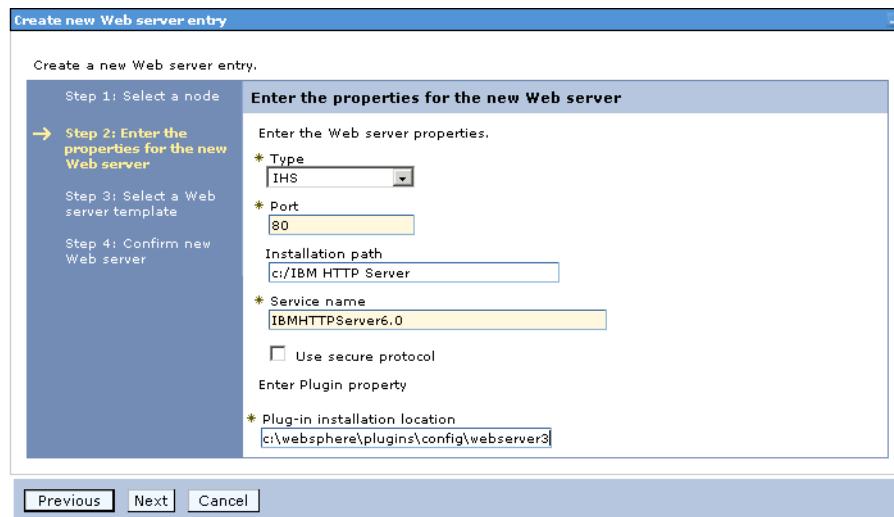


Figure 8-11 Defining a Web server: Step 2

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.

5. Enter the parameters required for remote administration. See Figure 8-12.



Figure 8-12 Defining a Web server: Step 3

6. As in Figure 8-13 on page 394, select a template. 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.

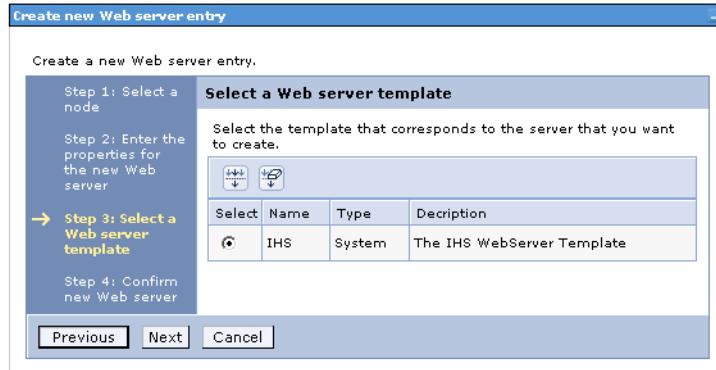


Figure 8-13 Defining a Web server: Step 4

7. Review the options as in Figure 8-14, and click **Finish**.

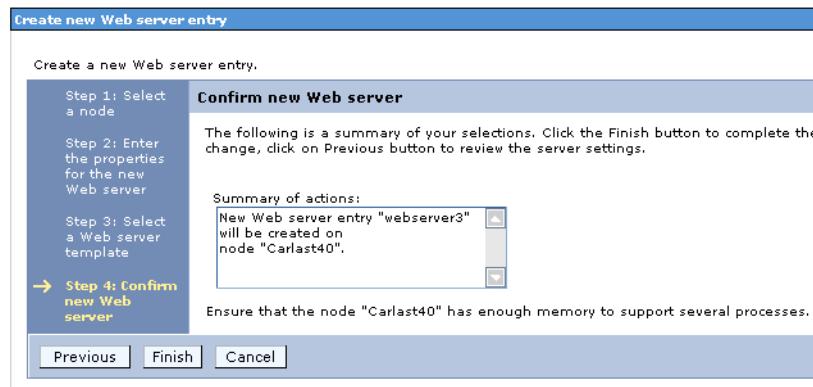


Figure 8-14 Defining a Web server: Step 5

8.3.2 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.

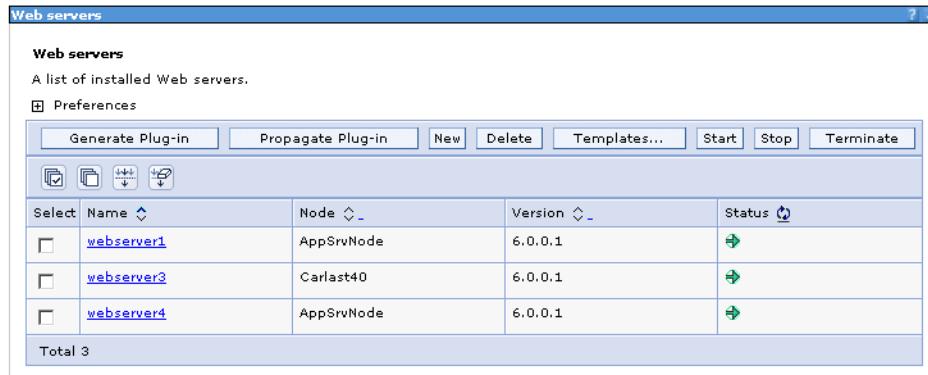


Figure 8-15 Web server status

WebSphere Application Server reports server status using the Web server host name and port that you have defined. See Figure 8-10 on page 392 and Figure 8-11 on page 393. 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 8-17 on page 398.

8.3.3 Starting and stopping a Web server

A Web server can be started or stopped in 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 8-16 on page 396.
2. Check the box to the left of each Web server you want.
3. Click **Start** or **Stop**.

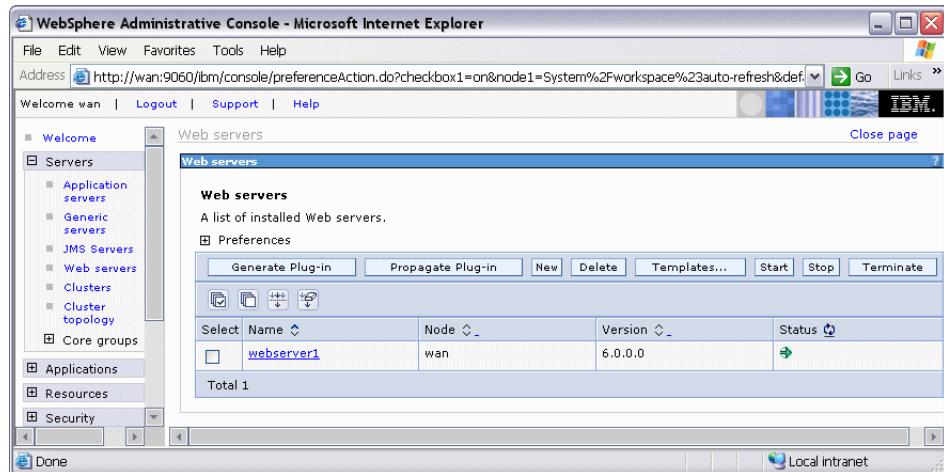


Figure 8-16 Web server definitions

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 6.0** service from the Services panel 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.

8.3.4 IBM HTTP Server remote administration

You can administer and configure IBM HTTP Server V6.0 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

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 8-17 on page 398.

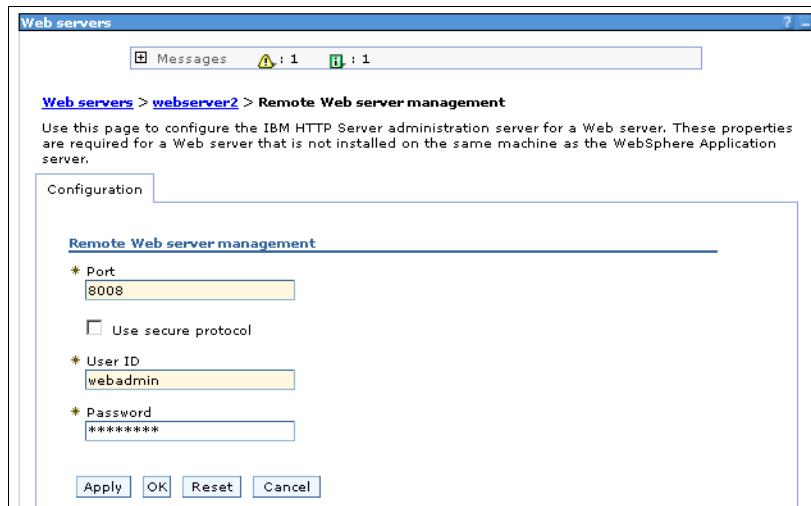


Figure 8-17 IHS remote management properties

- a. Enter the port number for the IHS administration server. The default is 8008.
- b. Check the **Use secure protocol** box if the port is secure. The default is not set.
- c. 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 in the WebSphere administrative console 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 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.
- ▶ 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 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 status for 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 are the following lines in Example 8-1 at the bottom of the httpd.conf file.

Example 8-1 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 is specified in these lines. When you generate the Web server plug-in configuration from the managed 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 8-18 on page 401.

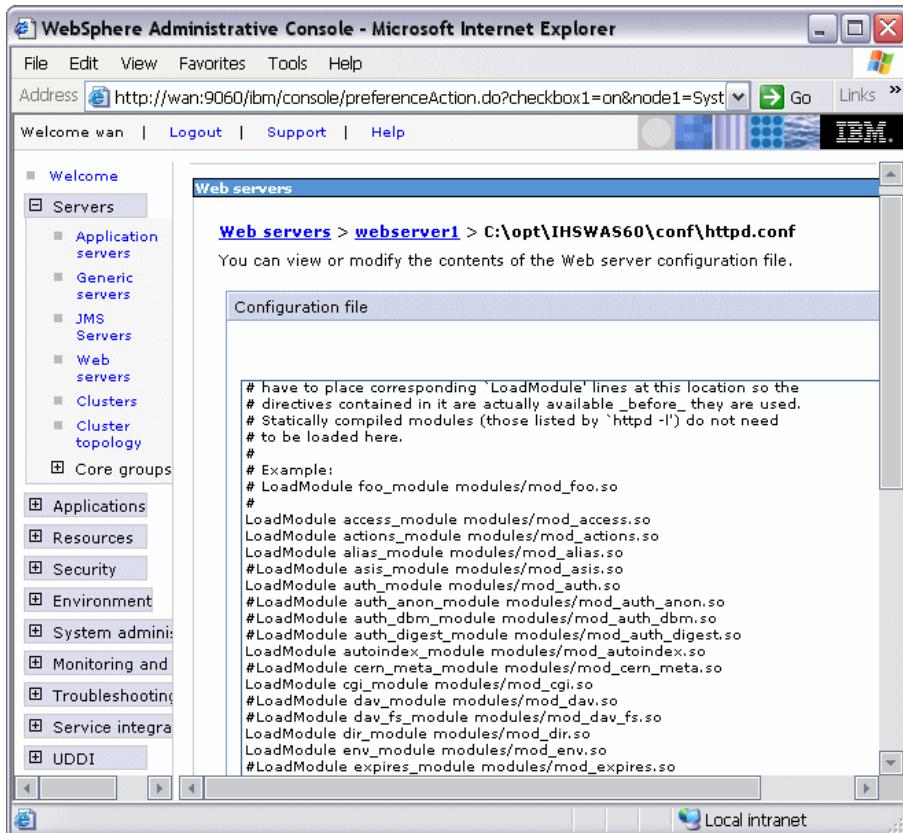


Figure 8-18 IBM HTTP Server configuration file *httpd.conf*

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 8-19 on page 402.

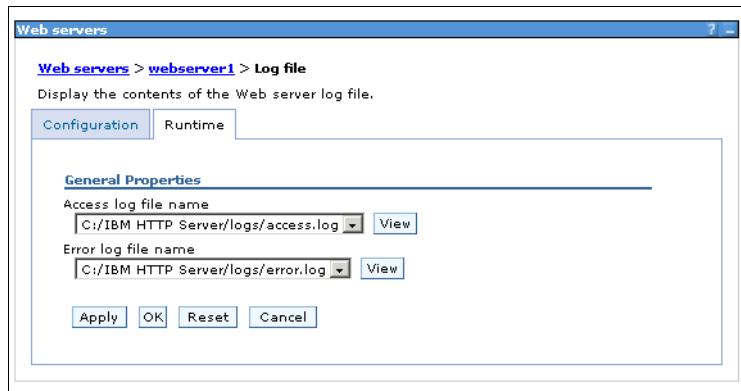


Figure 8-19 Web server Runtime page for logs

5. Click **View** beside the log you want to view. See Figure 8-20.

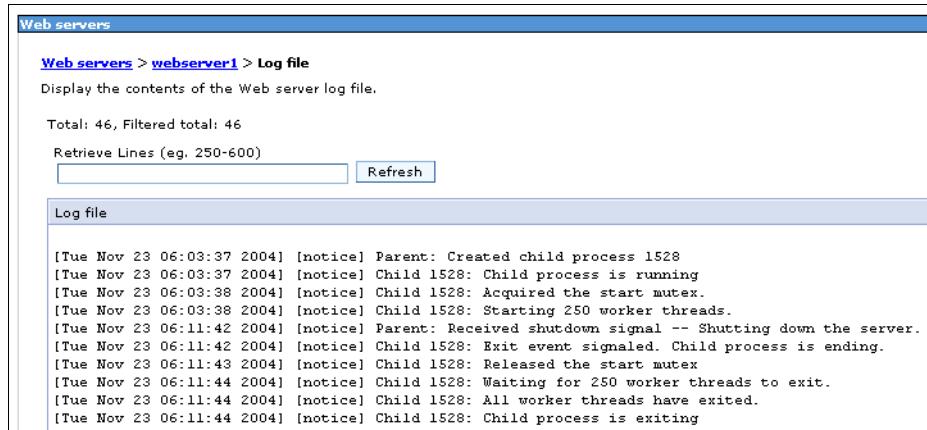


Figure 8-20 Viewing the error log

8.3.5 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. Select **Map modules to servers** in the Additional Properties section.
4. The **Selecting servers** panel is displayed, as in Figure 8-21.
5. Examine the list of mappings. Ensure that each Module entry is mapped to all targets identified under Server.

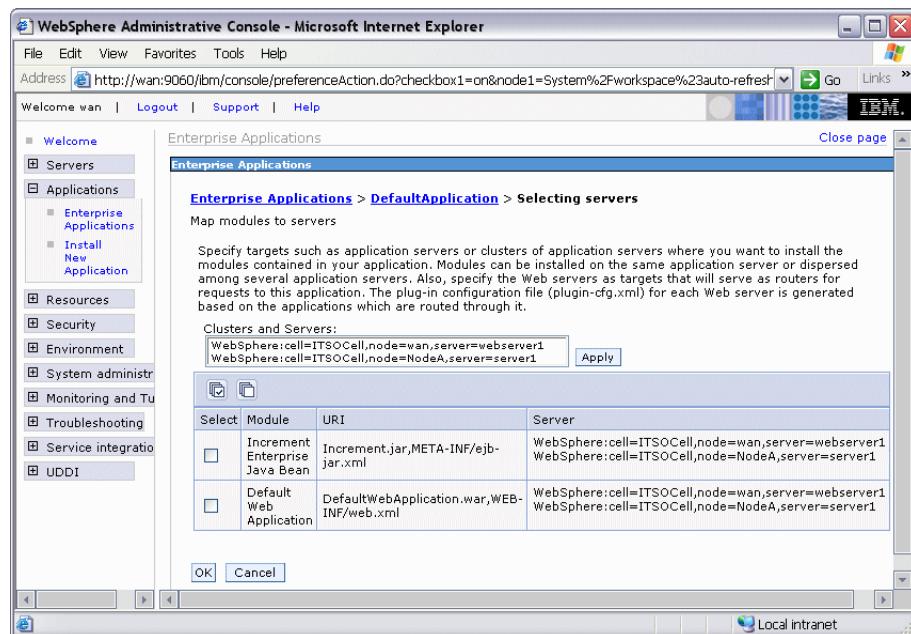


Figure 8-21 Map modules to selected servers

6. 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.
7. Click **Apply**.

8. Repeat step 6 on page 403 until each module maps to the desired targets.
9. Click **OK** and save your changes.
10. 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.

8.4 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 plugin 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 8-2 shows an excerpt from a generated plug-in configuration file.

Example 8-2 An excerpt from the plugin-cfg.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?><!--HTTP server plugin config file
for the webserver ITSOCell.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 = <webappuri>/servlet/ is generated.

Both these options apply for both the 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.

8.4.1 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 410.

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 8-22 on page 407.

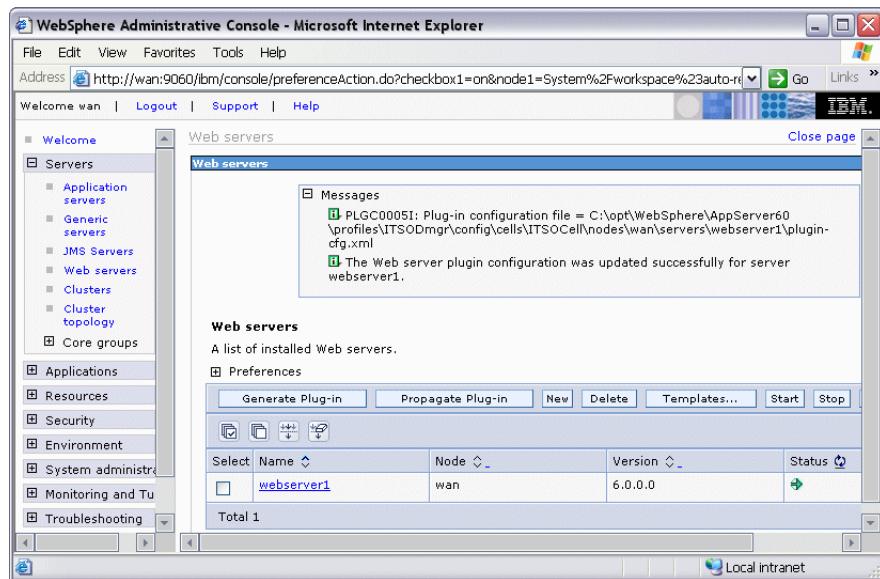


Figure 8-22 Web server definitions

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 8-23 on page 408. You can also open it with a text editor.

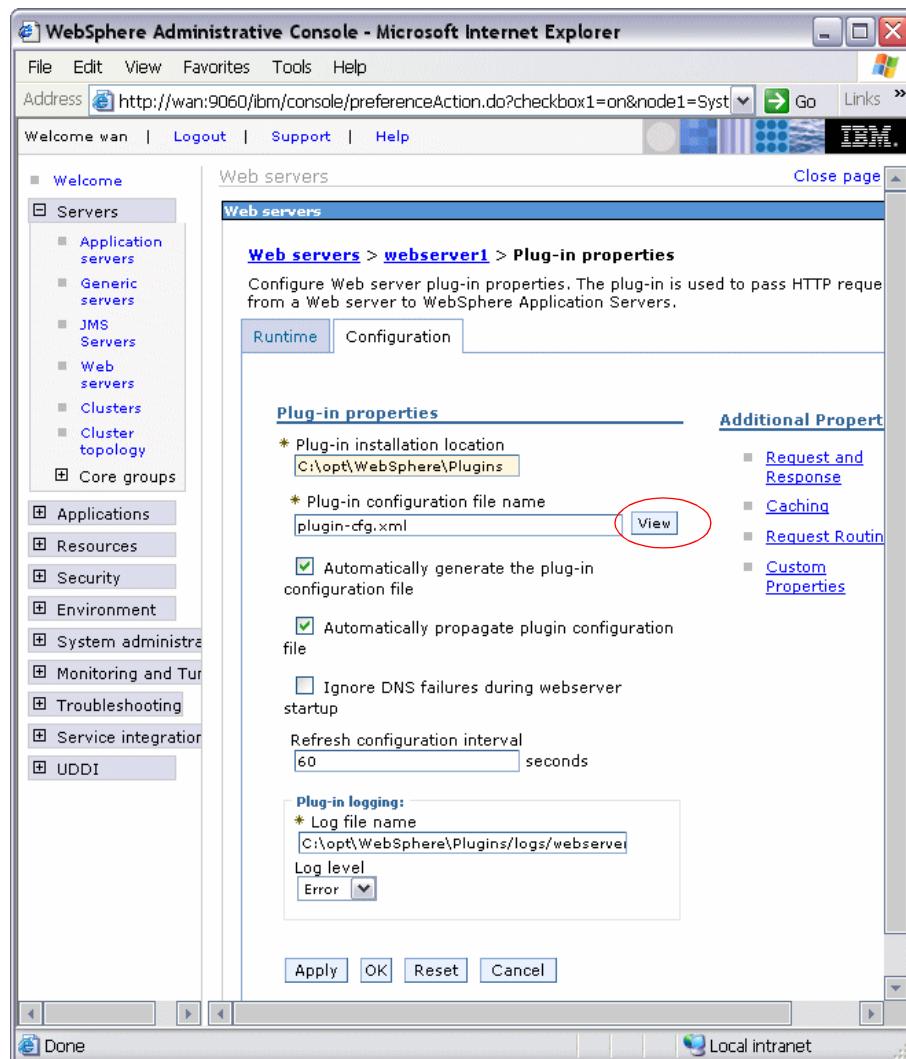


Figure 8-23 Plug-in properties

To use the new plugin-cfg.xml file you must propagate it to the Web server system. See 8.4.2, “Propagating the plug-in configuration file” on page 411.

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 8-1.

Table 8-1 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.

Option	Description
-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.
-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:

```
GenPluginCfg -cell.name BaseApplicationServerCell -node.name appServerNode  
-server.name appServerName
```

To generate a plug-in configuration file for a Web server:

```
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 8-22 on page 407. 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.

8.4.2 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.

```
<plug-ins_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 8-22 on page 407.
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 8-22 on page 407 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.

8.4.3 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 8-24.

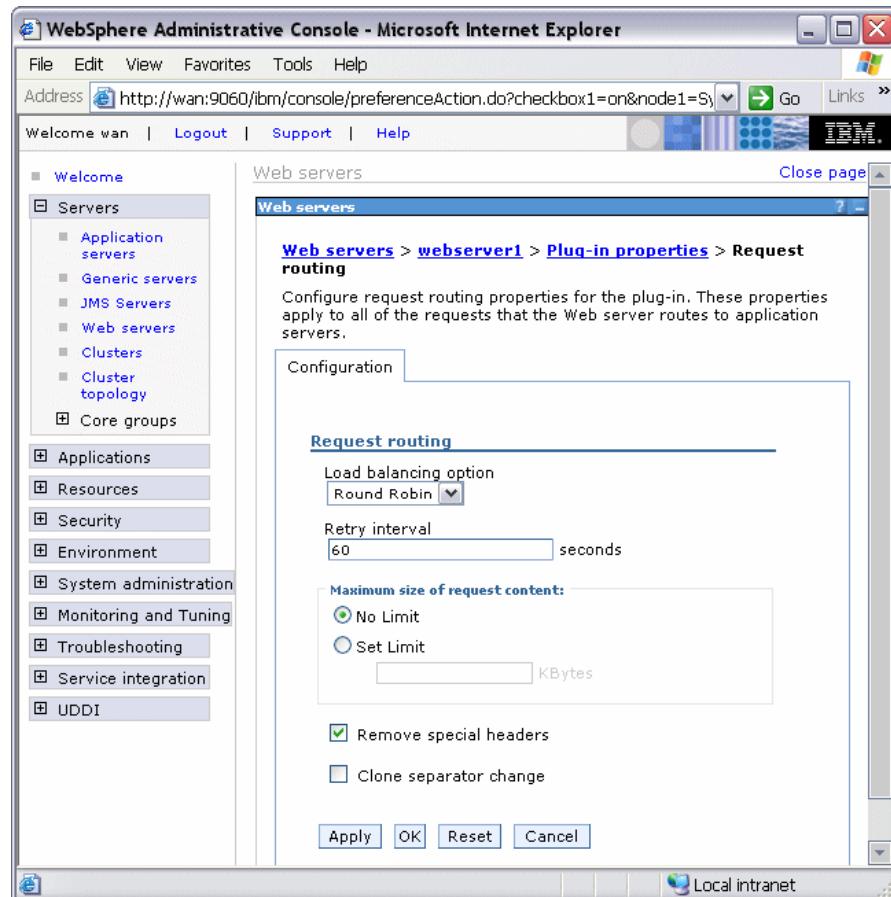


Figure 8-24 Request routing properties

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 a cluster member at random 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.



Problem determination

Problems within an e-business environment can take many forms, including poor performance, application unavailability, or unexpected results. The first step in resolving a problem is to isolate and understand it.

In this chapter, we introduce tools and techniques that can be used to analyze and correct problems. Included in this chapter is information about:

- ▶ 9.1, “Resources for identifying problems” on page 418
- ▶ 9.2, “Administrative console messages” on page 419
- ▶ 9.3, “Log files” on page 420
- ▶ 9.4, “Traces” on page 430
- ▶ 9.5, “Log Analyzer” on page 443
- ▶ 9.6, “Collector tool” on page 451
- ▶ 9.7, “First Failure Data Capture logs” on page 452
- ▶ 9.8, “Dumping the contents of the name space” on page 453
- ▶ 9.9, “HTTP session monitoring” on page 454
- ▶ 9.10, “Application debugging and tracing” on page 455
- ▶ 9.11, “Product installation information” on page 456
- ▶ 9.12, “Resources for problem determination” on page 459

9.1 Resources for identifying problems

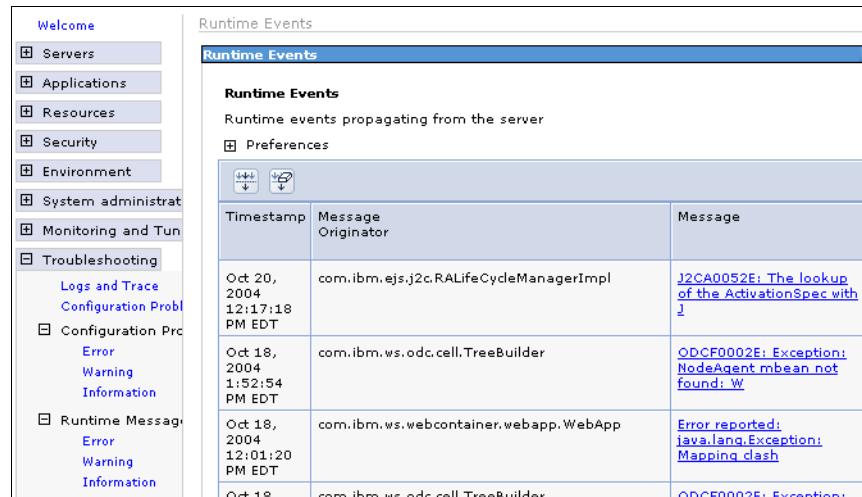
WebSphere provides the following sources of feedback to help with problem determination. Each are described separately in subsequent sections:

- ▶ *Administrative console messages* provide important information regarding runtime events and configuration problems. They are an important starting point to determine the cause of any configuration problem.
- ▶ Several general-purpose *log files* are provided, such as JVM standard logs, process (native) logs, and IBM service logs.
- ▶ *Traces* provide more detailed information about WebSphere components to determine what is wrong with your WebSphere environment.
- ▶ The *Log Analyzer* is a GUI tool that permits the user to view any logs generated with log analyzer trace format, such as the IBM service log file. This tool gives the user error message explanations and information such as why the error occurred and how to recover from it.
- ▶ The *Collector tool* gathers information about the application server installation and packages it in an output JAR file. The information in the file includes logs, property files, configuration files, operating system and Java data, and prerequisite software presence and levels. If the need arises, you can send this file to IBM Customer Support to assist in problem determination and analysis.
- ▶ The *First Failure Data Capture (FFDC)* function preserves the information generated from a processing failure and returns control to the affected engines. The captured data is saved automatically for use in analyzing the problem, and could be collected by the Collector tool.

9.2 Administrative console messages

The **Troubleshooting** section of the administrative console displays Runtime status messages. You can view the messages from the **Configuration Problems** and **Runtime Messages** folder structures, respectively.

Figure 9-1 shows the runtime messages pane containing the Error list.



The screenshot shows the administrative console's left sidebar with various navigation options like Servers, Applications, Resources, Security, Environment, System administration, Monitoring and Tuning, Troubleshooting, Logs and Trace, Configuration Problems, Configuration Procs, and Runtime Messages. The Runtime Messages section is expanded, showing Error, Warning, and Information levels. The main pane is titled 'Runtime Events' and contains a table with columns for Timestamp, Message Originator, and Message. There are four rows of data:

Timestamp	Message Originator	Message
Oct 20, 2004 12:17:18 PM EDT	com.ibm.ejs.j2c.RALifeCycleManagerImpl	J2CA0052E: The lookup of the ActivationSpec with J
Oct 18, 2004 1:52:54 PM EDT	com.ibm.ws.odc.cell.TreeBuilder	ODCF0002E: Exception: NodeAgent mbean not found: W
Oct 18, 2004 12:01:20 PM EDT	com.ibm.ws.webcontainer.webapp.WebApp	Error reported: java.lang.Exception: Mapping clash
Oct 18	com.ibm.ws.odc.cell.TreeBuilder	ODCF0002E: Exception:

Figure 9-1 Runtime message errors view

View a message in detail by clicking the message text in the Message column. Figure 9-2 on page 420 shows sample message details.

Runtime Events

[Runtime Events](#) > **Message Details**

Runtime events propagating from the server

General Properties

Message	handleServerChangeEvent(): PLGC0055E: Unable to get the list of Web servers defined on the node PlatoCell.ates/servertypes/APPLICATION_SERVER. PLGC0026E: An exception occurred while reading the server index for the server ates/servertypes/APPLICATION_SERVER.
Message type	Error
Explanation	No explanation found for ID=handleServerChangeEvent()
User action	No user action found for ID=handleServerChangeEvent()
Message Originator	com.ibm.websphere.plugincfg.initializers.ServerIndexChangePluginTask
Source object type	RasLoggingService
Timestamp	Oct 19, 2004 5:38:04 PM EDT
Thread Id	1149
Node name	PlatoCellManager
Server name	dmgr

Figure 9-2 Message details

This same layout structure is available for the configuration problem views. The problem entries are removed from the configuration problem list as the problems are corrected.

9.3 Log files

WebSphere Application Server can write system messages to several general-purpose logs. Here is a list of log types and where they are stored:

- ▶ *JVM logs* are created by redirecting the System.out and System.err streams of the JVM. By default, these files are stored as `<profile_home>/logs/<server_name>/SystemOut.log` and `SystemErr.log`.
- ▶ *Process (native) logs* are created by redirecting the stdout and stderr streams of the process's native module (.dlls, .so, UNIX libraries, and other JNI native

- modules), including the JVM native code itself. These logs can contain information relating to problems in native code, or diagnostic information written by the JVM. By default, these files are stored as `<profile_home>/logs/<server_name>/native_stderr.log` and `native_stdout.log`.
- ▶ *Service log* is a special log named, by default, `activity.log`. This log is written in a binary format and cannot be viewed directly using a text editor. Use Log Analyzer or the Showlog tool to view the log.

9.3.1 JVM (standard) logs

The JVM (standard) logs are created by redirecting the `System.out` and `System.err` streams. WebSphere Application Server writes formatted messages to the `System.out` stream. In addition, applications and other code can write to these streams using the `print()` and `println()` methods defined by the streams. Some JDK built-ins such as the `printStackTrace()` method on the `Throwable` class can also write to these streams.

Typically, the `System.out` log is used to monitor the health of the running application server. The `System.err` log contains exception stack trace information that is useful when performing problem analysis.

Configuring the JVM logs

To view and modify the settings for the JVM `System.out` and `System.err` logs using the administrative console:

1. Click **Troubleshooting** → **Logs and Trace** in the navigation tree
2. Select a server by clicking the server name.
3. Click **JVM Logs**.

Note: You can also reach this point by selecting **Servers** → **Application Servers**. Open the server configuration page and select **Logging and Tracing** under the Troubleshooting section.

4. Select the **Configuration** tab, if it is not selected by default.

5. Scroll through the window to display the attributes. See Figure 9-3.

[Logging and Tracing > SocratesServer1 > JVM Logs](#)

Configuration Runtime

General Properties

System.out

* File Name:
\${SERVER_LOG_ROOT}/Syste

File Formatting
Basic (Compatible) ▾

Log File Rotation

File Size Time

Maximum Size
1 MB Start Time
24

Repeat Time
24 hours

Maximum Number of Historical Log Files
1

Installed Application Output

Show application print statements
 Format print statements

System.err

* File Name:
\${SERVER_LOG_ROOT}/Syste

Log File Rotation

File Size Time

Maximum Size
1 MB Start Time
24

Repeat Time
24 hours

Maximum Number of Historical Log Files
1

Installed Application Output

Show application print statements
 Format print statements

Apply OK Reset Cancel

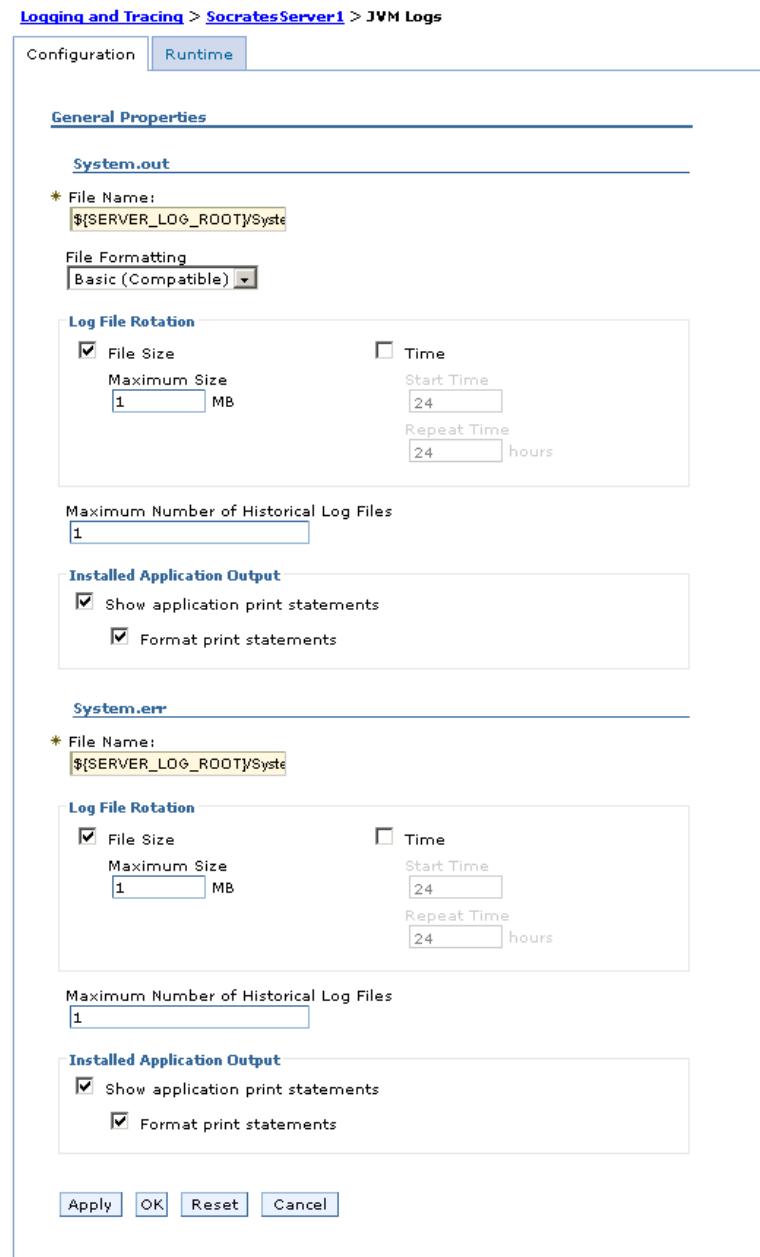


Figure 9-3 Configuring the JVM logs setting

Fill in the information for the following fields.

► **File Name:**

You can enter the name of the System.out or System.err file. The file name specified on the Configuration tab must have one of the following values:

- file name

The name of a file in the file system. You can use a fully qualified file name. If the file name is not fully qualified, it is considered to be relative to the current working directory (*<profile_home>*) for the server. The file will be created if it does not exist. See Figure 9-4.

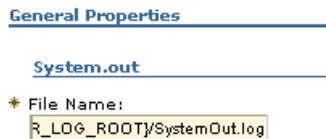


Figure 9-4 Enter the file name

The default is \${SERVER_LOG_ROOT} with the following options:

- \${SERVER_LOG_ROOT} = \${LOG_ROOT}/server1 (server scope)
- \${LOG_ROOT} = \${USER_INSTALL_ROOT}/logs (node scope).
- \${USER_INSTALL_ROOT} = <profile_home> (node scope)

For example, the default name for the System.out log on a Windows system might look like this:

C:\WebSphere\AppServer\profiles\AppSrv01\logs\server1\SystemOut.log

- console

This is a special file name used to redirect the stream to the corresponding process stream. If this value is specified for System.out, the file is redirected to stdout. If this value is specified for System.err, the file is redirected to stderr. See Figure 9-5.

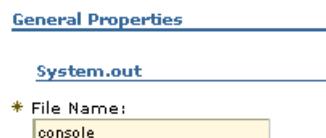


Figure 9-5 Enter console

- none

Using the value of none discards all data written to the stream and is equivalent to redirecting the stream to dev/null on a UNIX system. See Figure 9-6.

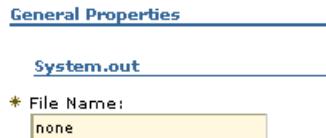


Figure 9-6 Enter none

► **File formatting:**

The File formatting field specifies the format to use in saving the System.out file. Your choices are:

- Basic records only basic information. This is the default:

```
<timestamp><threadID><shortName><eventType>[class] [method]<message>
```

- Advanced extends the basic format by adding information about an event, when possible:

```
<timestamp><threadID><eventType><UOW><source=longName>[class] [method]<organization><Product><Component><message>
```

The addition of the unit of work information is particularly valuable when debugging in a distributed environment.

► **Log file rotation:**

A self-managing log file writes messages to a file until some criteria, either size or time, is reached. At the specified time, or when the file reaches the specified size, the current file is closed and renamed to a name consisting of the current name plus a time stamp. The stream then reopens a new file reusing the original name and continues writing.

– **File size**

If this option is selected, the file automatically performs self-maintenance by rolling over the file when it reaches the specified maximum size.

– **Maximum size**

This attribute specifies the maximum size in megabytes to which the file is allowed to grow.

- **Time**

Selecting this attribute allows the log file to manage itself based on the age of the file. If this option is selected, the file will roll itself over after the specified time period.

- **Start time**

This attribute specifies the hour of the day, from 1 to 24, from which the periodic rollover algorithm begins. The periodic rollover algorithm uses this hour to load the algorithm at application server startup. Once started, the rollover algorithm runs without adjustment until the application server is stopped.

- **Rollover period**

Specify the hour of the day, from 1 to 24, when the periodic rollover algorithm starts. The rollover always occurs at the beginning of the specified hour of the day. The first hour of the day, which starts at 00:00:00 (midnight), is hour 1 and the last hour of the day, which starts at 23:00:00, is hour 24. Therefore, if you want log files to roll over at midnight, set the start time to 1.

Note that if both file size and time are selected, the file is rolled over based on the criteria met first.

- ▶ **Maximum Number of Historical Log Files:**

In this field, you can specify the number of rolled-over files to keep.

- ▶ **Installed Application Output:**

Specify whether the System.out or System.err print statements issued from the application code are logged and formatted. Your choices are:

- Show application print statements

This option causes application messages written to this stream using the print and println stream methods to be shown. This will have no effect on system messages written to the stream by the WebSphere Application Server.

- Format print statements

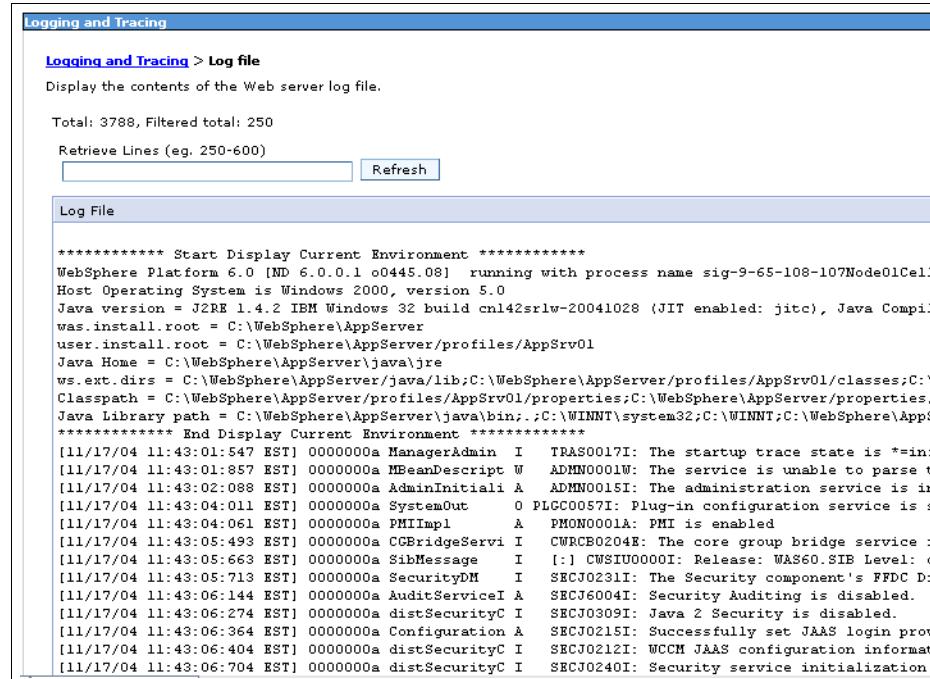
This option causes application messages written to this stream using the print and println stream methods to be formatted like WebSphere system messages.

6. Change the appropriate configuration attributes and click **Apply**.

7. Save your configuration changes.

The JVM logs are written as plain text files, so you can open and view the files directly using your own text editor.

You can also view the JVM logs using the Runtime tab as shown in Figure 9-7, enabling viewing the JVM logs from a remote machine.



The screenshot shows the 'Logging and Tracing' interface with the 'Log file' tab selected. It displays the contents of the 'Web server log file'. The log output includes:

```
***** Start Display Current Environment *****
WebSphere Platform 6.0 [ND 6.0.0.1 o0445.08] running with process name sig-9-65-108-107Node01Cell
Host Operating System is Windows 2000, version 5.0
Java version = J2RE 1.4.2 IBM Windows 32 build cnl42sr1w-20041028 (JIT enabled: jitc), Java Compil.
was.install.root = C:\WebSphere\AppServer
user.install.root = C:\WebSphere\AppServer\profiles\AppSrv01
Java Home = C:\WebSphere\AppServer\java\jre
ws.ext.dirs = C:\WebSphere\AppServer\java\lib;C:\WebSphere\AppServer\profiles\AppSrv01\classes;C:\\
Classpath = C:\WebSphere\AppServer\profiles\AppSrv01\properties;C:\WebSphere\AppServer\properties;
Java Library path = C:\WebSphere\AppServer\java\bin.;C:\WINNT\system32;C:\WINNT;C:\WebSphere\Apps
***** End Display Current Environment *****

[11/17/04 11:43:01:547 EST] 00000000 ManagerAdmin I TRAS0017I: The startup trace state is *=inf
[11/17/04 11:43:01:857 EST] 00000000 MBeanDescriptor W ADMN0001W: The service is unable to parse t
[11/17/04 11:43:02:088 EST] 0000000a AdminInitiali A ADMN0015I: The administration service is in
[11/17/04 11:43:04:011 EST] 0000000a SystemOut O PLGC0057I: Plug-in configuration service is s
[11/17/04 11:43:04:061 EST] 0000000a PMImpl A PMON0001A: PMI is enabled
[11/17/04 11:43:05:493 EST] 0000000a CCBridgeServi I CWRCB0204E: The core group bridge service i
[11/17/04 11:43:05:663 EST] 0000000a SibMessage I [:] CWSIU0000I: Release: WAS60.SIB Level: o
[11/17/04 11:43:05:713 EST] 0000000a SecurityDM I SECJ0231I: The Security component's FFDC Di
[11/17/04 11:43:06:144 EST] 0000000a AuditServiceI A SECJ6004I: Security Auditing is disabled.
[11/17/04 11:43:06:274 EST] 0000000a distSecurityC I SECJ0309I: Java 2 Security is disabled.
[11/17/04 11:43:06:364 EST] 0000000a Configuration A SECJ0215I: Successfully set JAAS login prov
[11/17/04 11:43:06:404 EST] 0000000a distSecurityC I SECJ0212I: WCCM JAAS configuration informat
[11/17/04 11:43:06:704 EST] 0000000a distSecurityC I SECJ0240I: Security service initialization
```

Figure 9-7 System.out log

JVM log message formats

Analyzing and understanding logs is a significant step in the problem determination process. Messages logged by application server components and associated IBM products start with a unique message identifier that indicates the component or application that issued the message.

Example 9-1 illustrates the basic format of a log or trace entry.

Example 9-1 JVM logs (basic format)

```
[10/25/04 11:58:46:518 EDT] 0000000a TCPPort E TCPC0003E: TCP Channel
TCP_1 initialization failed. The socket bind failed for host * and port 9061.
The port may already be in use.
```

A description of each field is shown in Table 9-1.

Table 9-1 Log entry format description

Field	Example	Description
Time stamp	[10/25/04 11:58:46:518 EDT]	The time stamp in fully qualified date, time and time zone format
Thread ID	0000000a	The thread ID or the hash code of the thread issuing this message
Component	TCPPort	The short name of component issuing this message
Event Type	E	<p>The type of the message or trace event, of which possible values are:</p> <ul style="list-style-type: none"> A Audit I Informational W Warning E Error F Fatal O System.out by the user application or internal components R System.err by the user application or internal components u A special type used by the message logging component of the WebSphere Application Server runtime Z A placeholder to indicate the type was not recognized
Message ID	TCPC0003E	The identifier of the message.
Message	TCP Channel TCP_1 initialization failed. The socket bind failed for host * and port 9061. The port may already be in use.	The text of the message and message arguments

The message identifier (message ID) can be either eight or nine characters in length and has the form:

CCCC1234X

To decode the message ID, understand the following:

- ▶ CCCC is a four-character alphabetic component or application identifier.
- ▶ 1234 is a four-character numeric identifier used to identify the specific message for that component.

- ▶ X is an optional, alphabetic severity indicator:
 - I = Informational,
 - W = Warning,
 - E = Error

To view the message IDs, or the meaning of the messages generated by WebSphere Application Server components, select the **Reference** view on the Information Center and from the **Troubleshooter** branch, expand the **Messages** topic.

9.3.2 Process (native) logs

The stdout and stderr streams written by native modules (.dlls, .so, UNIX libraries, and other JNI modules) are redirected to the native log files at application server startup. By default, these files are stored as <profile_home>/logs/<server_name>/native_stderr.log and native_stdout.log.

To view or change the log settings or to view the log, do the following:

1. Click **Troubleshooting** → **Logs and Trace** in the navigation tree.
2. Select a server by clicking the server name.

Note: You can also reach this point by selecting **Servers** → **Application Servers**. Open the server configuration page and select **Logging and Tracing** under the Troubleshooting section.

3. Click **Process Logs**.
4. To view the settings, select the **Configuration** tab. To view the logs, select the **Runtime** tab.

9.3.3 IBM service (activity) log

The IBM service log is a special log written in a binary format to capture events that show a history of WebSphere Application Server activities, also known as the *activity log*.

By default, the IBM service log is shared among all server processes for a node. The configuration values for the IBM service log are inherited by each server process from the node configuration.

Note: You can configure a separate IBM service log for each server process by overriding the configuration values at the server level.

Follow these steps to view or change the IBM service log settings using the administrative console:

1. Select **Troubleshooting** → **Logs and Trace**.
2. Select the server by clicking the name.
3. Select **IBM Service Logs**. See Figure 9-8.

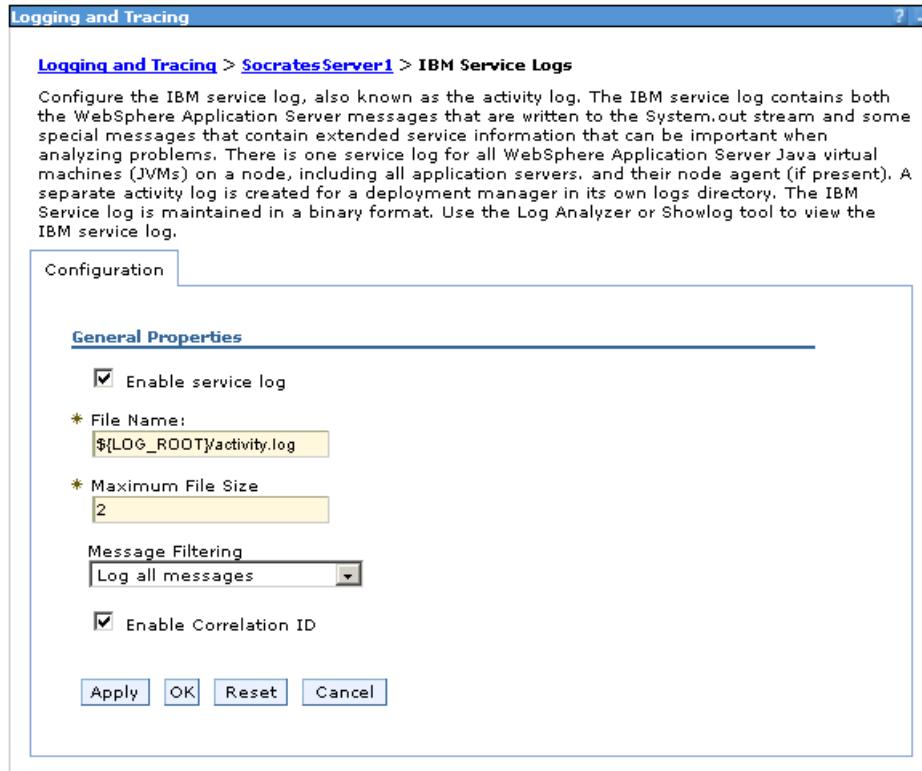


Figure 9-8 IBM Service log

Define the following fields:

- The service log is enabled by default. Clear the **Enable service log** check box to disable it.
- **File Name:** sets the name for the service log. The default location is `<profile_home>/logs/activity.log`. If the name is changed, the runtime requires write access to the new file, and the file must use the .log extension.

- **Maximum File Size** specifies the number of megabytes to which the file can grow. When the file reaches this size, it wraps, replacing the oldest data with the newest data.
 - **Message Filtering** sets the message filter level to the desired state. You can select to store all messages or select a combination of service, warning, and error message.
 - The **Enable Correlation ID** option allows you to specify whether a correlation ID should be generated and included in message events and diagnostic trace entries. If you check this box, each application client request is assigned a unique identifier that is propagated to all servers touched as part of servicing that request. This allows correlation of events across multiple server processes.
4. Save the configuration.
 5. Restart the server to apply the configuration changes.

Viewing the service log

To view the service log, use the **showlog** command in the `<profile_home>/bin` directory. This command dumps the binary log file to standard out or a file. The format is:

```
showlog [option] binaryFilename [outputFilename]
```

`outputFilename` is optional. If no file name is given, **showlog** dumps the service log file to standard out. For example:

```
showlog ..../logs/activity.log
```

The `option` can specify the output to be formatted in XML syntax. Use the option `-format CBE-XML-1.0.1.` for XML syntax.

Another powerful way to view the service log file is to use Log Analyzer, described in 9.5, “Log Analyzer” on page 443.

9.4 Traces

Tracing can be useful if you have problems with particular components of WebSphere Application Server, clients and other processes, and the log files do not provide you with enough information to determine the problem.

Note: Traces need manual activation, and you need to remember to turn off tracing when you have finished collecting the information. Traces can impact performance rather severely.

Tracing is most likely to be used by IBM Service for diagnosing WebSphere problems and not by the typical user diagnosing application problems.

9.4.1 Diagnostic trace service

By default, the trace for all WebSphere Application Server components is disabled.

Enabling trace at server startup

The trace configuration settings are read at server startup time and used to configure the trace service. To configure or change the diagnostic trace settings, using the administrative console:

1. Select **Troubleshooting** → **Logs and Trace**.
2. Click the server name. You can select an application server, node agent, or the deployment manager.
3. Select **Diagnostic Trace**.
4. Select the **Configuration** tab, if it is not shown by default. See Figure 9-9 on page 432.

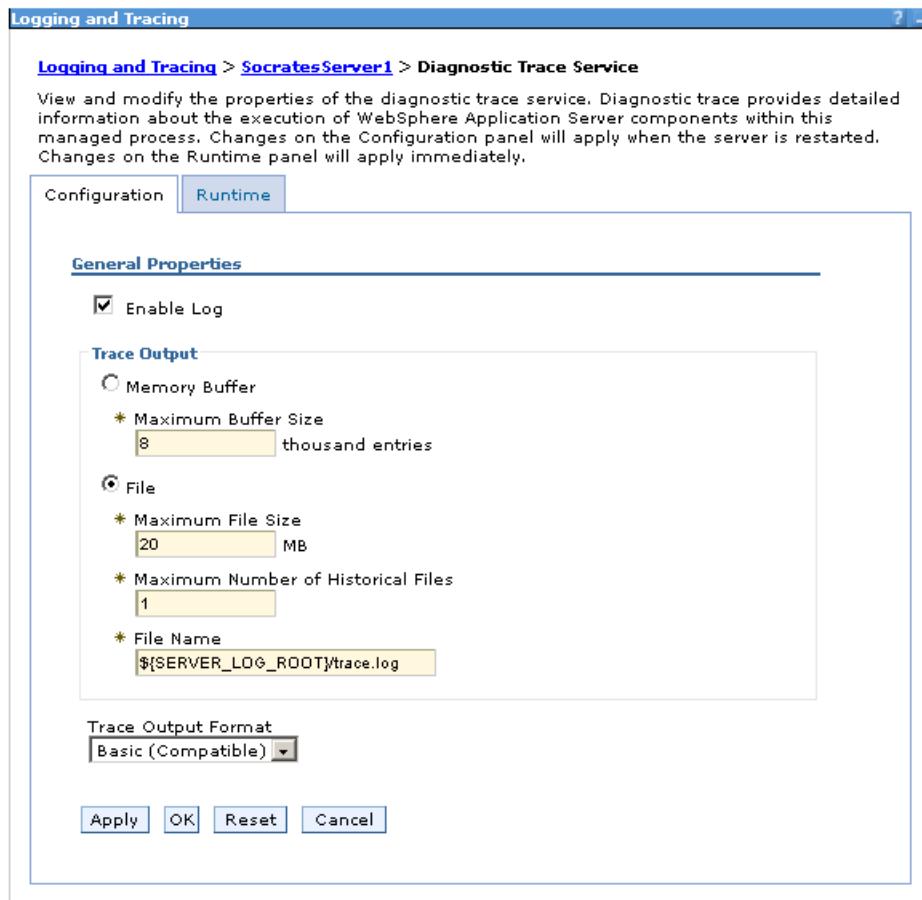


Figure 9-9 Trace service configuration

Check the information in the following fields:

- Check the **Enable Trace** box to enable tracing.
- Trace Output

Select whether to direct trace output to either a file or an in-memory circular buffer.

- If the in-memory buffer is selected, set the size of the buffer, expressed in thousands of lines. When using the in-memory circular buffer, the buffer must be dumped to a file before it can be viewed. This can be done using the **Dump** button on the Runtime tab.

- If a file is selected, set the maximum size in megabytes to which the file should be allowed to grow. When the file reaches the size, the existing file will be closed, renamed and a new file with the original name reopened.

– **Trace output format**

Select the desired format for the generated trace. The options are **Basic (Compatible)** for a minimum trace, **Advanced** for detailed traces, and **Log Analyzer**.

5. Save the changed configuration.
6. Set the proper trace strings. The options for this are described in “Trace string specification” on page 433.
7. Start or restart the server.

Note: You can also enable a trace for a running server. See “Enabling trace on a running server” on page 437.

Trace string specification

The trace information logged from the Diagnostic trace service depends on the monitoring level set on the WebSphere components. The level is configured by use of a trace specification string. To configure the trace level using the administrative console, do the following:

1. Select **Troubleshooting** → **Logs and Trace**.
2. Click the server name. You can select an application server, node agent, or the deployment manager.
3. Select **Change Log Detail Levels**.
4. Select the **Configuration** tab, if it is not shown by default. See Figure 9-10 on page 434.

Logging and Tracing > SocratesServer1 > Change Log Detail Levels

Log levels allow you to control which events are processed by Java logging. Click Components to specify a log detail level for individual components, or Groups to specify a log detail level for a predefined groups of components. Click a component or group name to select a log detail level. Log detail levels are cumulative; a level near the top of the list includes all levels below it.

Configuration Runtime

General Properties

Change Log Detail Levels

Components	[i] IMPORTANT: To view log events that are below the Detail Level, you must enable the Diagnostic Trace Service. Log events that are at Detail Level or above can be viewed in the SystemOut log, IBM Service Log (when enabled), or the Diagnostic Trace Service (when enabled).
Groups	<div style="border: 1px solid #ccc; padding: 5px; min-height: 150px; width: 100%;"> <p style="margin-top: 0;">*=info</p> </div>

*** [All Components]**

- ConfigError**
- ConnLeakLogic**
- JaasWCCMHelper**
- ORBRas**
- SASRas**
- SystemErr**
- SystemOut**
- WAS.clientinfo**
- WAS.clientinfopluslogging**
- com.ibm.debug.***
- com.ibm.ejs.***
- com.ibm.etools.***
- com.ibm.websphere.***
- com.ibm.ws.***
- com.ibm.wsspi.***
- com.ibm.xml.***
- sun.rmi.loader**

Figure 9-10 Log details levels properties

Every WebSphere component offers a comprehensive detail of information for tracing. The list shown in Figure 9-10 shows all base components with a default log level specification of info. You can specify the log level either by manually entering the trace string into the text box, or by clicking the name of the component from the list and selecting the log level from the pop-up box. Log level can also be configured from the administrative console on Groups instead (logical grouping of the components). Simply select the **Groups** link on the properties pane on the left. See Figure 9-11 on page 435.

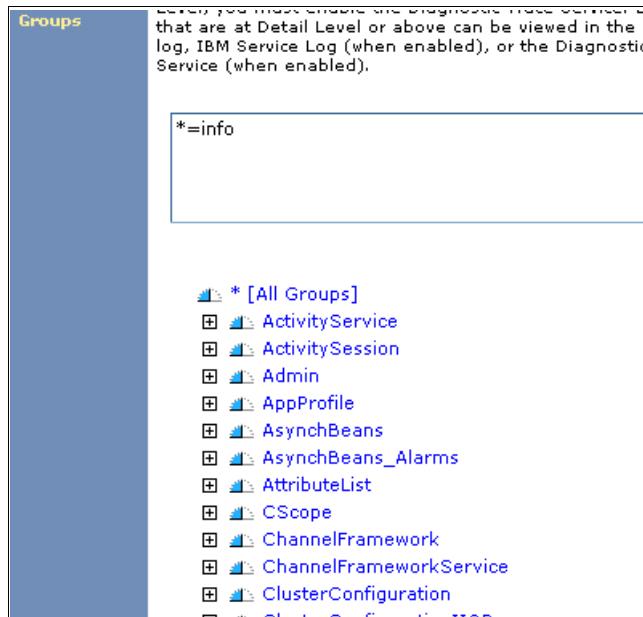


Figure 9-11 Log level details by Group

Trace strings must conform to a specific grammar for processing by the trace service:

```
COMPONENT_TRACE_STRING[:COMPONENT_TRACE_STRING]*
```

For example, you can use this phrase:

```
COMPONENT_TRACE_STRING = COMPONENT_NAME=LEVEL
```

The elements in this string are as follows:

- ▶ **COMPONENT_NAME** is the name of a component or group registered with the trace service. Typically, WebSphere Application Server components register using a fully qualified Java class name, for example `com.ibm.ws.webcontainer.servlet.ServletWrapper`. In addition, you can use a wildcard character of asterisk (*) to terminate a component name and indicate multiple classes or packages. For example, use a component name of `com.ibm.ws.webcontainer.servlet.*` to specify all components whose names begin with `com.ibm.ws.webcontainer.servlet`.
- ▶ **LEVEL** = <level> represents the type of tracing to perform. The possible level values are listed in the Version 6 Logging Level column of Table 9-2 on page 436. Only the levels Fine, Finer, Finest and All generate trace information for the Diagnostic trace service.

Table 9-2 Logging level

Version 6 Logging Level	Logging Level pre-Version 6	Trace Level pre-Version 6	Content / Significance
Off	Off	All disabled*	<p>Logging is turned off.</p> <p>*In Version 6, a trace level of All disabled will turn off trace, but will not turn off logging. Logging will be enabled from the Info level.</p>
Fatal	Fatal	-	Task cannot continue and component/ application/ server cannot function.
Severe	Error	-	Task cannot continue but component/ application/ server can still function. This level can also indicate an impending fatal error.
Warning	Warning	-	Potential error or impending error. This level can also indicate a progressive failure (for example, the potential leaking of resources).
Audit	Audit	-	Significant event affecting server state or resources
Info	Info	-	General information outlining overall task progress
Config	-	-	Configuration change or status
Detail	-	-	General information detailing subtask progress
Fine	-	Event	Event Trace information - General trace + method entry / exit / return values
Finer	-	Entry/Exit	Trace information - Detailed trace
Finest	-	Debug	Trace information - Most detailed trace that includes all the detail that is needed to debug problems
All	-	All enabled	All events are logged

Examples of legal trace strings include:

Example 9-2 Lega trace strings

```
com.ibm.wsspi.*=detail  
com.ibm.ws.tcp.channel.impl.TCPChannel=all:com.ibm.ws.webcontainer.*=fine  
com.ibm.ws.wlm.*=finest  
com.ibm.ws.wlm.*=finest:com.ibm.wsspi.*=detail:com.ibm.ws.tcp.channel.impl.  
TCPChannel=all
```

Trace strings cannot contain blanks and are processed from left to right. Specify a trace string such as:

```
abc.*=all
```

This string enables the trace for all components whose names start with abc.

Enabling trace on a running server

You can also trace a server that is already active:

1. Select **Troubleshooting** → **Logs and Trace**.
2. Click the server name. You can select an application server, node agent, or the deployment manager.
3. Select **Change Log Detail Levels**.
4. Select the **Runtime** tab. The options are similar to those in the Configuration tab, described in “Trace string specification” on page 433.
 - Check the box for **Save runtime changes...** if you want to write your changes back to the server configuration. If this option is not selected, the changes you make will apply only for the life of the server process that is currently running.
 - Specify the trace string as described in “Trace string specification” on page 433.
5. Click **Apply**.
6. To configure the trace output select **Troubleshooting** → **Logs and Trace**.
7. Click the server name. You can select an application server, node agent, or the deployment manager.
8. Select **Diagnostic Trace Service**.
9. Select the **Runtime** tab. The options are similar to those in the Configuration tab, described in “Enabling trace at server startup” on page 431.
 - Click the **Save Trace** check box if you want to write your changes back to the server configuration. If this option is not selected, the changes you

make will apply only for the life of the server process that is currently running.

- If you are using the in-memory buffer, use the **Dump** button to dump the buffer to the specified file. This is necessary before viewing the trace output. The dump buffer file will be placed in the <profile_home> directory, for example:

```
c:\websphere\appserver\profiles\myNode
```

Looking at trace output

Traces in basic format have the following format in Example 9-3:

Example 9-3 Basic format

```
<timestamp><threadId><shortName><eventType>[className] [methodName]<textmessage>
[parameter 1]
[parameter 2]
```

Traces in advanced format have the following format in Example 9-4:

Example 9-4 Advanced format

```
<timestamp><threadId><eventType><UOW><source=longName>[className] [methodName]
<Organization><Product><Component>[thread=threadName]<textMessage>
[parameter 1=parameterValue] [parameter 2=parameterValue]
```

The EventType field is a one-character field that indicates the type of the trace event. Table 9-3 shows the possible values:

Table 9-3 Trace event types

Type	Description
>	method entry
<	method exit
1	fine or event trace type
2	finer trace type
3	finest, debug or dump trace type
Z	type was not recognized

Example 9-5 on page 439 shows the trace output in basic format.

Example 9-5 Trace output (basic format)

```
[10/25/04 16:40:17:386 EDT] 000010a9 LocalNotifica < handleNotification Exit
[10/25/04 16:40:17:386 EDT] 000010a9 NotificationD 3 Returned from listener
#0
[10/25/04 16:40:17:386 EDT] 0000085f jsp           1
com.ibm.ws.jsp.webcontainerext.JSPExtensionServletWrapper checkForTranslation
Exiting checkForTranslation sync block for /jsp/template.jsp
[10/25/04 16:40:17:386 EDT] 0000085f BNFHeadersImp 3 getHeaderAsString(h,i):
$WSIS 0 [null]
[10/25/04 16:40:17:386 EDT] 0000085f WebAppTransac 3
WebAppTransactionCollaborator.preInvoke() -->
/WebSphereBank/jsp/searchbycustomer.jsp
```

9.4.2 Web server logs and traces

If a problem is suspected with the Web server or between the Web server and the Web container, there are several tools you can use.

Web server plug-in generation

The Web server plug-in configuration file controls what content is transferred from the Web server to an application server. This file must be regenerated after certain changes to the WebSphere configuration server and then moved or propagated to the proper location on the Web server.

If there is a problem with requests being routed to WebSphere Application Server from the Web server, make sure the Web server plug-in has been properly generated and moved to the Web server. For information about how to do this, see 8.4.1, “Regenerating the plug-in configuration file” on page 406.

Web server plug-in log

The Web server plug-in creates a log file containing error and informational messages. The level of information placed in this log is determined by a setting in the Web server plug-in configuration file. The possible values in order of significance are:

- ▶ Trace
- ▶ Stats
- ▶ Warn
- ▶ Error (Default)

Specifying one value for the level means you get that level plus the functions below it. With Trace, you also get Stats, Warn and Error. Choosing Stats gives you Stats, Warn and Error. Example 9-6 on page 440 gives a sample of a file log setting.

Example 9-6 Web server plug-in configuration file log setting

```
<

<?xml version="1.0" encoding="ISO-8859-1"?>
.....
<Log LogLevel="Error" Name="....\Plugins\logs\http_plugin.log"/>
...
</Config>
```

This setting can be changed manually or from the administrative console. Configuring the plug-in trace level from the administrative console is done from the plug-in configuration page, found from:

1. Select **Servers →Web servers.**
2. Select the web server by clicking the name from the list.
3. Select **Plug-in properties.**
4. The properties page has a plug-in logging section for configuring the logging level.

By changing this setting manually in the plug-in file, the setting is only changed temporarily, until you generate the Web server plug-in configuration using the administrative console.

Note: Be careful when setting the level to Trace. A lot of error messages are logged at this level that can cause the disk space to fill up very quickly. A Trace setting should never be used in a normally functioning environment because it affects performance.

IBM HTTP Server logs

The IBM HTTP Server has the following log files that aid in problem diagnosis:

- ▶ Error log
- ▶ Access logs

Error log

The error log records IBM HTTP Server errors. The location for the error log is specified with the ErrorLog directive. The LogLevel directive determines the level of logging. You can specify one of the following values for LogLevel:

emerg:	Emergencies - system is unusable
alert:	Action must be taken immediately
crit:	Critical conditions
error:	Error conditions
warn:	Warning conditions

```
notice:      Normal but significant condition
info:       Informational
debug:      Debug level messages
```

When a particular level is specified, messages from all other levels of higher significance will be reported as well. A level of at least `crit` is recommended.

Access log

The access log records all Web server activity, including the following information for each request:

- ▶ What was requested
- ▶ Who requested it
- ▶ When it was requested
- ▶ The method used
- ▶ The type of file sent in response
- ▶ The return code

Often, the access log is combined with two other logs called the *referrer* and *agent* logs by specifying a log format that includes all the information normally found in each log. The information in these last two logs is of more interest to a Webmaster who is gathering statistical information.

The format and location of the logs is determined by the log settings in the `<ihs_home>/conf/httpd.conf` configuration file. Example 9-7 shows the settings for the log files.

The `LogFormat` directives define the content of the log records. At the end of each `LogFormat` directive is a name identifying the format. Looking at the `LogFormat` directives in Example 9-7, the names are `combined`, `common`, `referer`, and `agent`. If you would like to customize the log formats, refer to the Apache Directives section of the IBM HTTP Server online Information Center.

The `CustomLog` directive indicates that the *custom* log format is to be used to store the log records. These records will be stored in `/usr/IBMHttpServer/logs/access_log`.

Example 9-7 IHS configuration file, httpd.conf

```
...
# ErrorLog: The location of the error log file. If this does not start
# with /, ServerRoot is prepended to it.
```

```
ErrorLog /usr/IBMHttpServer/logs/error_log
```

```
# LogLevel: Control the number of messages logged to the error_log.
# Possible values include: debug, info, notice, warn, error, crit,
# alert, emerg.
```

```

LogLevel warn

# The following directives define some format nicknames for use with
# a CustomLog directive (see below).

LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\""
combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent

# The location of the access logfile (Common Logfile Format).
# If this does not start with /, ServerRoot is prepended to it.

CustomLog /usr/IBMHttpServer/logs/access_log common
...

```

Including elapsed times in the log records

The default LogFormat record used (common) does not include the setting for service elapsed time. The elapsed time is often useful in understanding performance problems. To add the elapsed time option, add %T at the LogFormat entry. This will report the time taken to serve the request, in seconds.

Example 9-8 Elapsed time option %T

```

...
#LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%h %l %u %t \"%r\" %>s %b %T" common
...

```

The elapsed time is reported in seconds.

IBM HTTP Server log rotation

On even a moderately busy server, the quantity of information stored in the log files is very large. It will, consequently, be necessary to periodically rotate the log files by moving or deleting the existing logs. This cannot be done while the server is running, because the server will continue writing to the old log file as long as it holds the file open. Instead, the server must be restarted after the log files are moved or deleted so that it will open new log files.

IBM HTTP Server is capable of writing error and access log files through a pipe to another process, rather than directly to a file. Example 9-9 on page 443 is a simple example using piped logs:

Example 9-9 Piped logs

```
# compressed logs
CustomLog "|/usr/bin/gzip -c >> /var/log/access_log.gz" common
# almost-real-time name resolution
CustomLog "|/usr/local/apache/bin/logresolve >> /var/log/access_log" common
```

One important use of piped logs allows log rotation without having to restart the server. The IBM HTTP Server includes a simple program called rotatelogs for this purpose. For example, to rotate the logs every 24 hours, you can use:

```
CustomLog "|/usr/local/apache/bin/rotatelogs /var/log/access_log 86400" common
```

ignore unnecessary log records in the access log file

Like all Web servers, the IBM HTTP Server records all HTTP access traffic in a log file. To manage the amount of information recorded, you can configure the log to ignore certain entries. Example 9-10 shows how to code the directives so that log entries for .gif and .jpg files are not recorded.

Example 9-10 Ignoring image entries in the access_log

```
...
SetEnvIf Request_URI \.gif$ ignore=gif
SetEnvIf Request_URI \.jpg$ ignore=jpg

#CustomLog /usr/IBMHttpServer/logs/access_log common
CustomLog /usr/IBMHttpServer/logs/access_log common env=!ignore
...
```

9.5 Log Analyzer

The Log Analyzer is a GUI tool that permits the user to view service and activity logs. It can take one or more logs, merge all the data, and display the entries in sequence.

More importantly, this tool is shipped with an XML database, the *symptom database*, which contains strings for some common problems, reasons for the errors, and recovery steps. The Log Analyzer compares every error record in the log file to the internal set of known problems in the symptom database and displays all the matches. From this, you can get error message explanations, why the error occurred and how to recover from it, as shown in Figure 9-12 on page 444.

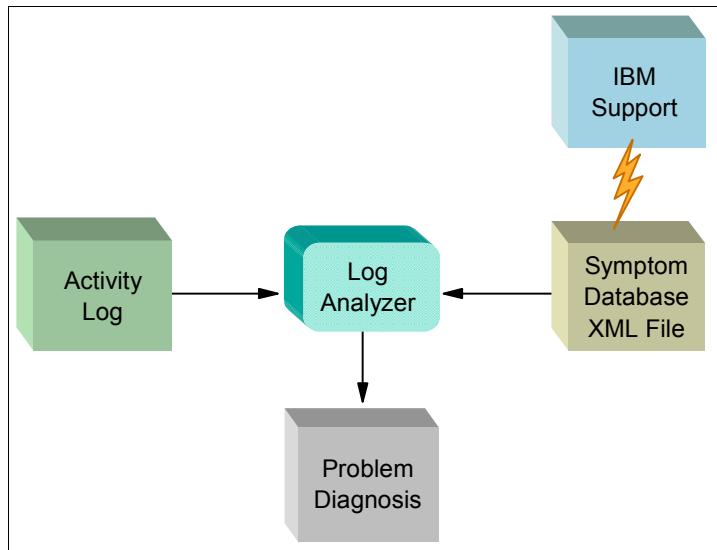


Figure 9-12 Log Analyzer concept

Note: The symptom database is maintained by IBM. We recommend that you refresh your copy often. To do this, see 9.5.3, “Updating the symptom database” on page 450.

9.5.1 Using Log Analyzer

To start using the Log Analyzer:

1. Run the **waslogbr.bat (.sh)** command from the `<was_home>/bin` directory.
 2. When the Log Analyzer GUI starts, select **File → Open** from the main menu. Navigate to the `<profile_home>/logs` directory, select **activity.log** and click **Open**.

You should now see the open activity log, similar to Figure 9-13 on page 445. All servers write log records to this file.

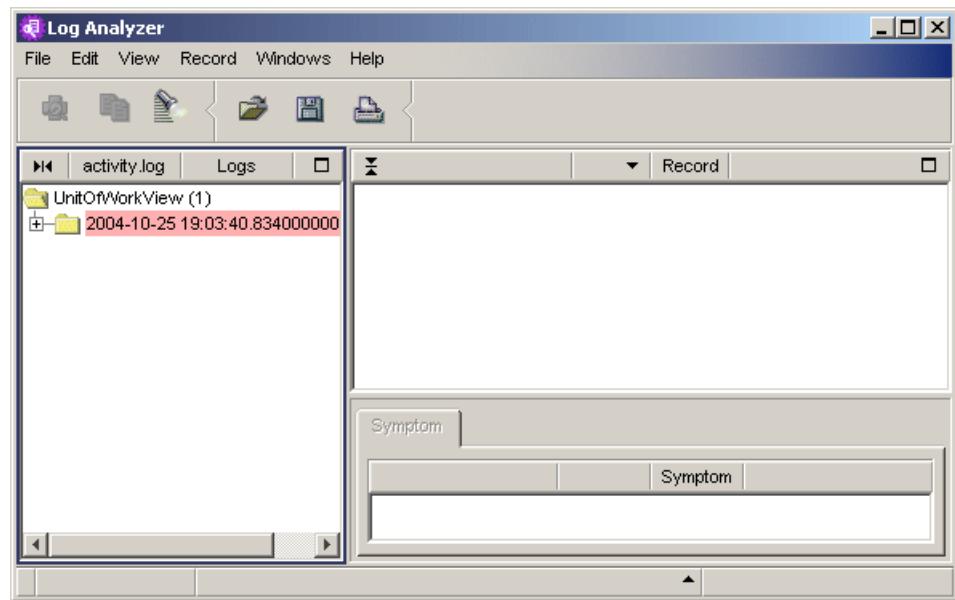


Figure 9-13 Log Analyzer

3. Select an entry in the UnitOfWorkView folder and the details appear in the upper-right pane.
4. To analyze a log entry, right-click the entry and select **Analyze** from the pop-up menu, as shown in Figure 9-14 on page 446. The entry is compared to the symptom database. If there is a match, the information appears in the lower-right pane.

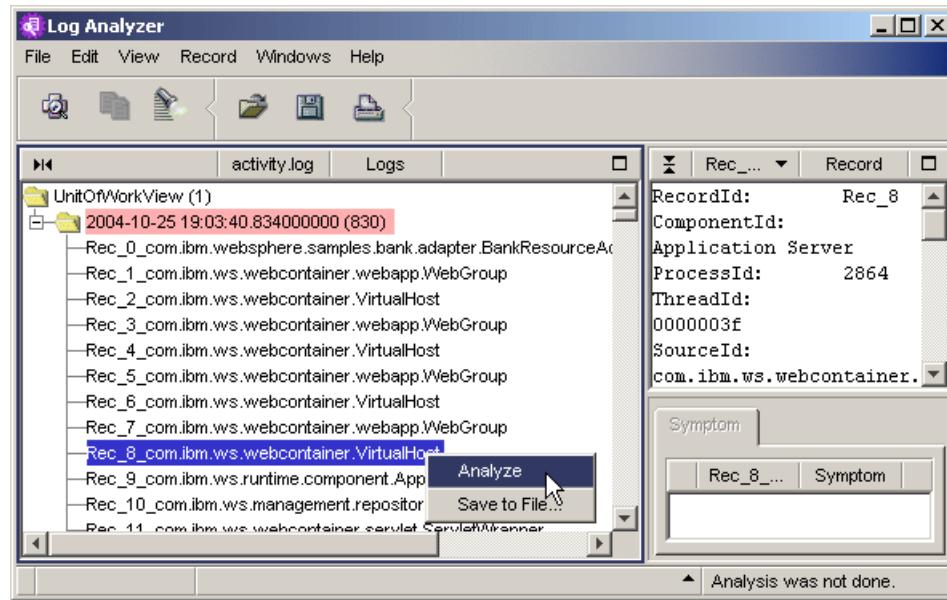


Figure 9-14 Selecting an entry to analyze

After the analyze action has been invoked, each analyzed log entry has an icon indicating whether analysis information is available. The check icon to the left of the entry in Figure 9-15 on page 447 indicates that analysis information is available.

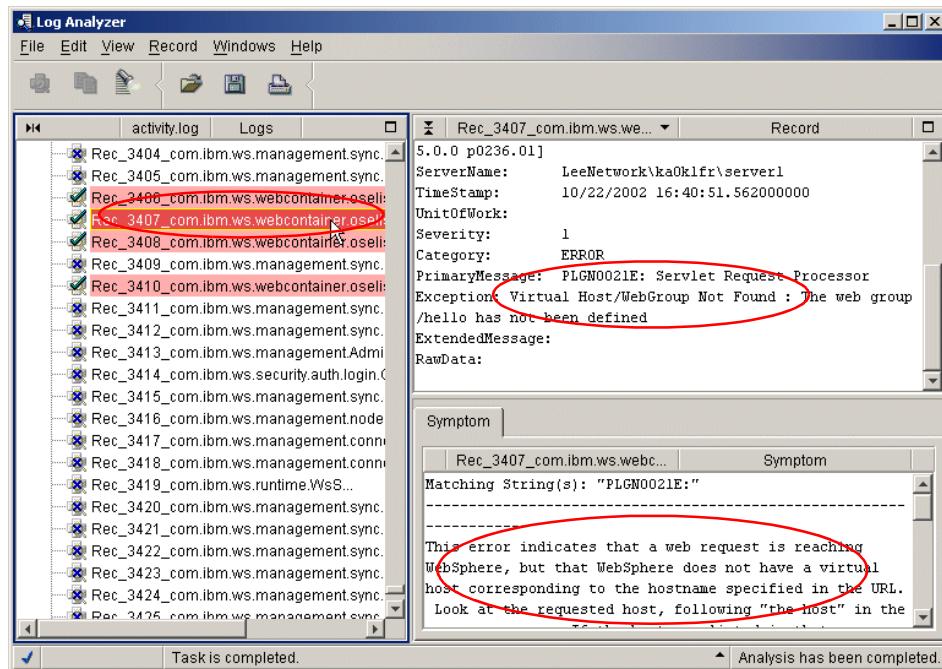


Figure 9-15 Log Analyzer information

Log entries (left pane)

By default, the pane on the left displays log entries by *unit of work* (*UOW*). It lists all the UOW instances and associated entries from the logs that you have opened. You might find the UOW grouping useful when you are trying to find related entries in the service or activity log, or when you are diagnosing problems across multiple machines.

The file name of the first log that you opened is shown in the pane's title bar. There is a root folder and under it, each UOW has a folder icon that you can expand to show all the entries for that UOW. All log entries without any UOW identification are grouped into a single folder in this tree view.

The UOW folders are sorted to show the UOW with the latest time stamp at the top of the list. The entries within each UOW are listed in the reverse sequence, that is the first (earliest) entry for that UOW is displayed at the top of the list. If you have merged several logs, all the log entries are merged in time stamp sequence within each UOW folder, as though they all came from the same log.

Each UOW folder name has the following format (see Figure 9-13 on page 445):

2004-10-22 17:33:16.359000000 (3469)

This example is comprised of these elements:

- ▶ 2004-10-22 17:33:16.359000000 is the time stamp.
- ▶ (3469) is the number of entries in the UOW.

Click the + icon next to the UOW folder to expand the folder. See Figure 9-15 on page 447. Each log entry's identification has the following format:

Rec_3407_com.ibm.ws.webcontainer.oselistener.OSELListenerDispatcher

In this example:

- ▶ Rec_3407 is the entry number.
- ▶ com.ibm.ws.webcontainer.oselistener.OSELListenerDispatcher is the class name.

Every log entry is assigned an entry number, Rec_nnnn, when a log is opened in the Log Analyzer. If more than one file is opened, as in merged files, the Rec_nnnn identification will not be unique because the number is relative to the entry sequence in the original log file and not to the merged data that is displayed. This Rec_nnnn also appears in the first line in the Records pane.

By default, each entry in this pane is color-coded to help you quickly identify the ones that have high severity errors.

Non-selected log entries have a background color of:

- ▶ Pink, if it has a severity 1 error
- ▶ Yellow, if it has a severity 2 error
- ▶ White, if it has a severity 3 error

Selected log entries have a background color of:

- ▶ Red, if it has a severity 1 error
- ▶ Green, if it has a severity 2 error
- ▶ Blue, if it has a severity 3 error

These colors are configurable and can be changed in the Log Analyzer's Preferences Log page. Select **File** → **Preferences** → **Logs** → **Severity**.

The Log Analyzer can also display the log entries in different sorting sequences. Select **File** → **Preferences** → **Logs**.

After the analyze action has been invoked, each analyzed log entry has the following icons:

-  The check icon indicates that the entry has some analysis information in one or more pages in the analysis pane.
-  The plus icon indicates that the entry has some analysis information. Look at the log entry prior to this one when diagnosing problems.
-  The question mark icon indicates that the entry has either a severity 1 or 2 error, but no additional analysis information is available for it.
-  The cross icon indicates that the entry has a severity 3 error and it has no analysis information.

Record pane (upper right)

When you select an entry under the unit of work in the logs pane, you see the details of the entry (process ID, thread ID, server name, severity, and so on) in the Record pane. The entry's identification is shown in the pane's title bar.

Right-click in this record pane to see the actions that you can perform on the entry. These actions include Analyze, Save to File, Find, Select All.

There is a drop-down arrow to the left of Record in the pane's title bar. By clicking **Record**, you can look at the last 10 records that you have viewed. The default cache size for the historical data is 10. Select **File → Preferences → General** to modify this number.

Analysis pane (lower right)

When the analyze action has been invoked, any information found in the symptoms database for the selected log entry appears in the symptom page. If the page tab is grayed out, there is no information in that page.

There is a status line at the bottom of the pane showing the status of actions.

9.5.2 Merging logs on multiple application servers

The correlation ID can be used to correlate activity to a particular client request, or correlate activities on multiple application servers.

To merge different service or activity.log files from different machines where your transaction occurred, do the following:

1. Make sure that Enable Correlation ID box is checked as discussed in 9.3.3, “IBM service (activity) log” on page 428.

2. Open one of the files in Log Analyzer and select **File → Preferences → Logs** to sort the log records in UnitOfWork and TimeStamp order to have a distributed log view.
3. Use the **File → Merge with** option to merge files.

9.5.3 Updating the symptom database

The symptom database included in the Log Analyzer package contains entries for common events and errors. New versions of the symptom database provide additional entries.

To download the symptoms database for your version using the Log Analyzer GUI, do the following:

- ▶ For WebSphere Application Server or Express, select **File → Update Database → WebSphere Application Server Symptom Database**.
- ▶ For WebSphere Application Server Network Deployment, select **WebSphere Application Server Network Deployment Symptom Database** from the main menu, as shown in Figure 9-16.

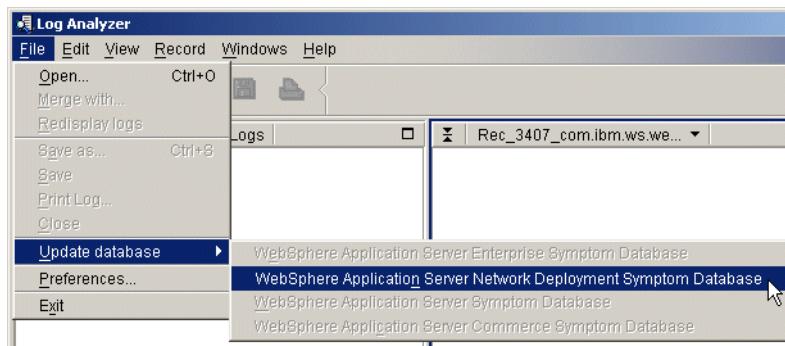


Figure 9-16 Updating the symptom database

Alternatively, you can download new versions of the database from the IBM FTP site. The URL for the FTP site is:

```
<was_home>/properties/logbr/ivblogbr.properties file
```

The symptom files are located in the `<was_home>/properties/logbr/symptoms` file.

If your organization uses an FTP or SOCKS proxy server, you can add a proxy definition to the Proxy Preferences page by doing the following:

1. Select **File → Preferences → Proxy**.

2. Select the appropriate proxy type.
3. Enter the host name and port number of the proxy server on the Proxy window.

9.6 Collector tool

The Collector tool gathers information about a WebSphere Application Server installation and packages it in an output JAR file. The file can be sent to IBM Customer Support to assist in problem determination and analysis. The information in the file includes logs, property files, configuration files, operating system and Java data, and prerequisite software presence and levels.

The -Summary option, is useful for determining the features installed. It produces a plain text file.

Running the Collector tool

Run the Collector tool under the root or administrator user ID because some of the commands require system access authority. However, if this is not possible and you proceed without administrator authority, most of the Collector functions work just fine.

The Collector tool writes its output files to the current directory, so it is good practice to create a new directory from which to run the tool. You cannot run the Collector tool in a directory under the WebSphere Application Server installation directory.

- ▶ For Windows systems, log on to the system as administrator or another user with administrator authority and enter the following, as in Example 9-11:

Example 9-11 Windows Collector tool logon

```
C:\> mkdir work
C:\> cd work
C:\work> <profile_home>\bin\collector.bat
```

- ▶ For UNIX systems, log on to the system as root and type as in Example 9-12 on page 452:

Example 9-12 Unix Collector tool logon

```
itsosvr:/home/# id  
root(...)  
itsosvr:/home/# mkdir work  
itsosvr:/home/# cd work  
itsosvr:/home/work# <profile_home>/bin/collector.sh
```

To collect information in a distributed server environment, invoke the Collector tool from the deployment manager profile directory.

Results

The Collector program creates an output .jar file in the current work directory. The .jar file name is based on the host name and package of the server on which the Collector tool was run, in the format:
<hostname-cellname-nodename-servername>-WASenv.jar.

What to do with the results

Send the <hostname-cellname-nodename-servername>-WASenv.jar file to IBM Customer Support for analysis.

9.7 First Failure Data Capture logs

The *First Failure Data Capture (FFDC)* function preserves the information generated from a processing failure and returns control to the affected engines. There are three property files located in <was_home>/properties which control the behavior of the FFDC filter:

- ▶ ffdcStart.properties, used while the server is starting
- ▶ ffdcRun.properties, used after the server is ready
- ▶ ffdcStop.properties, used while the server is stopping

The captured data is saved automatically in the <profile_home>/logs/ffdc directory for use in analyzing the problem, and could be collected by the Collector tool.

The First Failure Data Capture tool is intended primarily for use by IBM Service. It runs as part of the WebSphere Application Server and you cannot start or stop it. It is recommended that you not attempt to configure the FFDC tool. If you experience conditions requiring you to contact IBM Service, your IBM Service representative will assist you in reading and analyzing the FFDC log.

9.8 Dumping the contents of the name space

The name space stored by a given name server can be dumped with the dumpNameSpace utility that is shipped with WebSphere Application Server. This utility can be invoked from the command line or from a Java program. The naming service for the WebSphere Application Server host must be active when this utility is invoked.

To invoke the utility through the command line, enter the following command from the `<profile_home>/bin` directory:

- ▶ UNIX:

```
dumpNameSpace.sh [[-keyword value ]...]
```

- ▶ Windows:

```
dumpNameSpace [[-keyword value ]...]
```

The following command shows how to invoke the dumpNameSpace utility from the command line:

```
dumpNameSpace -?
```

The generated output looks like Example 9-13, which is the *short* dump format.

```
dumpNameSpace -host localhost -report short
```

Example 9-13 dumpNameSpace output

```
Getting the initial context
Getting the starting context
=====
Name Space Dump
  Provider URL: corbaloc:iiop:localhost:30006
  Context factory: com.ibm.websphere.naming.WsnInitialContextFactory
  Requested root context: cell
  Starting context: (top)=PericlesCell
  Formatting rules: jndi
  Time of dump: Wed Oct 27 15:10:09 EDT 2004
=====

=====
Beginning of Name Space Dump
=====
  1 (top)                                     javax.naming.Context
  2 (top)/nodes                               javax.naming.Context
  3 (top)/nodes/PericlesNode                  javax.naming.Context
  4 (top)/nodes/PericlesNode/cell             javax.naming.Context
  4   Linked to context: PericlesCell
  5 (top)/nodes/PericlesNode/nodename        java.lang.String
```

```

6 (top)/nodes/PericlesNode/node          javax.naming.Context
6   Linked to context: PericlesCell/nodes/PericlesNode
7 (top)/nodes/PericlesNode/domain        javax.naming.Context
7   Linked to context: PericlesCell
8 (top)/nodes/PericlesNode/persistent    javax.naming.Context
9 (top)/nodes/PericlesNode/servers       javax.naming.Context
10 (top)/nodes/PericlesNode/servers/server1 javax.naming.Context
11 (top)/nodes/PericlesNode/servers/server1/servername
11                               java.lang.String
12 (top)/nodes/PericlesNode/servers/server1/cell  javax.naming.Context
12   Linked to context: PericlesCell
13 (top)/nodes/PericlesNode/servers/server1/eis   javax.naming.Context
...
11                               java.lang.String
12 (top)/domain                      javax.naming.Context
12   Linked to context: PericlesCell
13 (top)/cellname                    java.lang.String
14 (top)/clusters                   javax.naming.Context
=====
End of Name Space Dump
=====
```

9.9 HTTP session monitoring

In the event of session-related problems, it is helpful to collect all session-related information. WebSphere Application Server provides an HTTP session tracker servlet called IBMTrackerDebug. To access the servlet from a browser, use the following URL:

`http://localhost:9080/servlet/com.ibm.ws.webcontainer.httpsession.IBMTrackerDebug`

The result is the printout in Example 9-14.

Example 9-14 Result of IBMTrackerDebug servlet

```
J2EE NAME(AppName#WebModuleName):: DefaultApplication#DefaultWebApplication.war
cloneId : -1

Number of sessions in memory: (for this webapp) : 11
use overflow : true
overflow size (for this webapp) :
Invalidation alarm poll interval (for this webapp) : 304
Max invalidation timeout (for this webapp) : 1800
Using Cookies : true
Using URL Rewriting : false
use SSLId : false
URL Protocol Switch Rewriting : false
```

```
Session Cookie Name : JSESSIONID
Session Cookie Comment : SessionManagement
Session Cookie Domain : null
Session Cookie Path : /
Session Cookie MaxAge : -1
Session Cookie Secure : false
Maximum in memory table size : 1000
current time : Wed Oct 23 18:18:37 EDT 2002
integrateWASSec :false
Session locking : false
Session locking timeout: 5
Allow access on lock timeout:true
Sessions Created:11
Active Count:0
Session Access Count:8
Invalidate Sessions Count:0
Invalidate By SessionManager:0
Garbage Collected count:0
SessionAffinity Breaks:0
Number of times invalidation alarm has run:0
Rejected Session creation requests(overflow off):0
Cache Discards:0
Attempts to access non-existent sessions:2
Number of binary reads from external store:0
Total time spent in reading from external store(ms):0
Total number of bytes read:0
Number of binary writes to external store:0
Total time spent in writing to external store(ms):0
Total number of bytes written out:0
Total size of serializable objects in memory :1859
Total number objects in memory :11
Min size session object size:169
Max size session object size :169
```

9.10 Application debugging and tracing

Debugging applications is beyond the scope of this book. However, we want to point out two facilities provided by WebSphere Application Server:

- ▶ Application Server Toolkit
- ▶ Java Logging API and Framework (Also known as JSR 47)

9.10.1 Application Server Toolkit

The Application Server Toolkit is included with WebSphere Application Server V6. It includes debugging functionality built on the Eclipse workbench. It provides the following adapters:

- ▶ WebSphere Application Server debug adapter

This adapter allows you to debug Web objects that are running on WebSphere Application Server and that you have launched in a browser. These objects include EJBs, JSPs, and servlets.

- ▶ JavaScript debug adapter

The JavaScript debug adapter enables server-side JavaScript debugging.

- ▶ Compiled language debugger

The compiled language debugger allows you to detect and diagnose errors in compiled-language applications such as C and C++.

- ▶ Java development tools (JDT) debugger

The JDT debugger allows you to debug Java.

All debug components in the Application Server Toolkit can be used for both local and remote debugging. To learn more about the debug components, launch the Application Server Toolkit and select **Help → Help Contents**. Choose the **Debugger Guide** bookshelf entry.

9.10.2 Java logging interface

With support of J2SE 1.4 in WebSphere Application Server V6, the Java logging interface is now supported. This framework enables application programmers to add logging statements to applications and categorize the information in a fine-grained fashion, for both debugging and production situations.

The log level can be configured in WebSphere to allow high-level (production) or detailed (tracing) information to be written to the system out or trace logs. See “Diagnostic trace service” on page 431. You can define specific application package strings to control the log level detail.

9.11 Product installation information

The WebSphere Application Server version and the versions of related software are important. All components need to be the correct versions for proper interoperation. In this section, we describe how to determine the versions and build levels of the various components in your environment.

9.11.1 Using the administrative console to find product information

Note: You can use this method only if the application server is running.

Perhaps the easiest way to get comprehensive information about the installation is to use the administrative console. You can use this when the server is running.

1. Select **Servers** → **Application Servers**.
2. Click the server.
3. Select the **Runtime** tab.
4. Click **Product Information** from the Additional Properties list. See Figure 9-17.

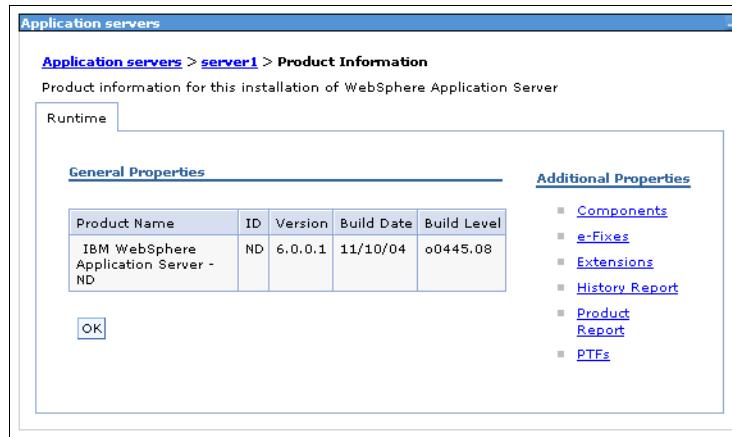


Figure 9-17 Product information

9.11.2 Locating WebSphere Application Server version information

To check the version of the product installed use one of these options:

- ▶ Look in the product version properties file of the installation:

<*was_home*>/properties/version/WAS.product

The file contains content similar to that shown in Example 9-15.

Example 9-15 WAS.product content

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE product PUBLIC "productId" "product.dtd">
<product name="IBM WebSphere Application Server - ND">
  <id>ND</id>
```

```
<version>6.0.0.1</version>
<build-info date="11/10/04" level="o0445.08"/>
</product>
```

- ▶ Look in the SystemOut.log file of one of the profile instances:

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

The file will contain content similar to that shown in Example 9-16.

Example 9-16 Node agent SystemOut.log content

```
***** Start Display Current Environment *****
WebSphere Platform 6.0 [ND 6.0.0.1 o0445.08] running with process name
dmgr01cell1\Node01\nodeagent and process id 2600
Host Operating System is Windows 2000, version 5.0
Java version = J2RE 1.4.2 IBM Windows 32 build cn142sr1w-20041028 (JIT enabled:
jitec), Java Compiler = jitec, Java VM name = Classic VM
....
....
***** End Display Current Environment *****
```

5. Execute the **versionInfo** command from the bin directory of the installation.
Output similar to Example 9-17 will be generated.

Example 9-17 versionInfo output for base installation

```
$ cd <was_home>/bin
$ versionInfo
...
...
Installation Platform
-----
Name      IBM WebSphere Application Server
Version   6.0

Technology List
-----
ND      installed

Installed Product
-----
Name      IBM WebSphere Application Server - ND
Version   6.0.0.1
ID       ND
Build Level o0445.08
Build Date 11/10/04
```

9.11.3 Finding the JDK version

To determine which version of the JDK is installed in your environment, use one of the following options:

- ▶ Look in the SystemOut.log file of one of the profile instances.
`<profile_home>/logs/server1/SystemOut.log`
- ▶ Run `java -fullversion` from the command line as in Example 9-13.
`<was_home>/java/bin/java -fullversion`

Example 9-18 java -fullversion command

```
C:\WebSphere\AppServer\java\bin>java -fullversion
java full version "J2RE 1.4.2 IBM Windows 32 build cn142sr1w-20041028"
```

9.11.4 Finding the IBM HTTP Server version

To check the version of your IBM HTTP Server on Windows platforms, run `apache -v`, as in Example 9-19.

Example 9-19 The apache -v command

```
C:\IBM HTTP Server\bin>apache -v
Server version: IBM_HTTP_Server/6.0 Apache/2.0.47
Server built:   Nov  4 2004
10:11:21
```

On UNIX platforms, run `httpd -v`.

9.12 Resources for problem determination

- ▶ WebSphere Application Server support (Fix Packs, fixes, and hints and tips)
<http://www.ibm.com/software/webservers/appserv/support.html>
- ▶ IBM alphaWorks® emerging technologies
<http://www.alphaworks.ibm.com>
- ▶ IBM developerWorks®
<http://www.ibm.com/developerworks/>
- ▶ Worldwide WebSphere User Group
<http://www.websphere.org>

- ▶ *An Introduction to Java Stack Traces*
<http://java.sun.com/developer/technicalArticles/Programming/Stacktrace/>
- ▶ *Apache HTTP Server Log Files*
<http://httpd.apache.org/docs/logs.html>
- ▶ IBM HTTP Server documentation library
<http://www.ibm.com/software/webservers/httpservers/library/>



Part 2

Messaging with WebSphere

This part of the book 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 runtime environment for messaging applications.

This part includes the following chapters:

- ▶ Chapter 10, “Asynchronous messaging” on page 463
- ▶ Chapter 11, “Default messaging provider” on page 593



Asynchronous messaging

In this chapter, we describe the concepts behind the asynchronous messaging functionality provided as part of WebSphere Application Server V6. 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

10.1 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 instance, 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.

10.1.1 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 which 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 runtime 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.

10.1.2 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.

10.1.3 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 10-1.

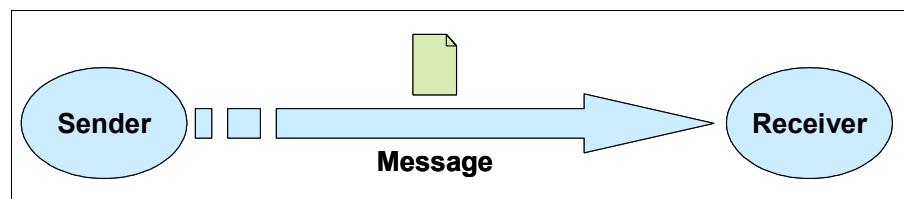


Figure 10-1 Direct communication using synchronous messaging

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 10-2.

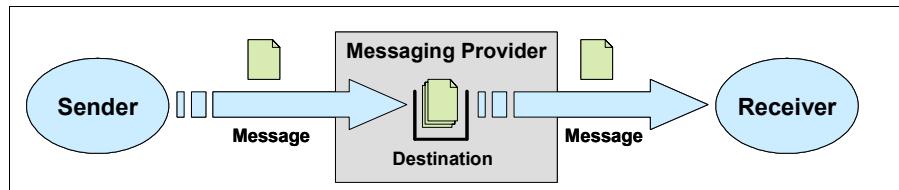


Figure 10-2 Indirect communication via a destination using asynchronous messaging

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 10-3.

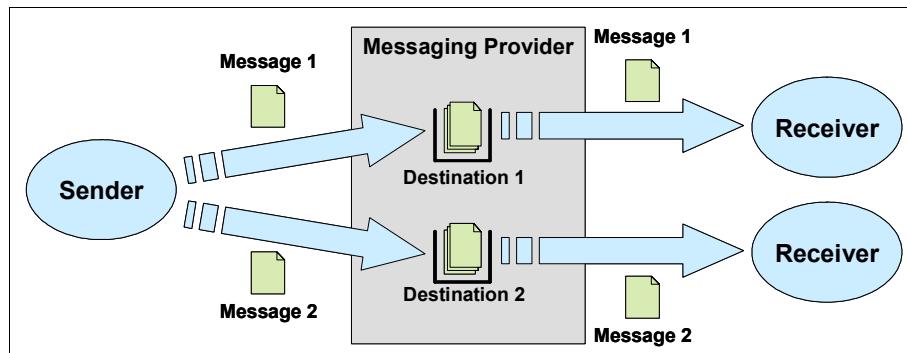


Figure 10-3 Communicating with multiple receivers using asynchronous messaging

10.1.4 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 10-2 on page 466. 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 10-3 on page 466. Figure 10-3 shows a sending application communicating with several receiving applications using the Point-to-Point messaging model with each. Figure 10-4 shows the Publish/Subscribe messaging model.

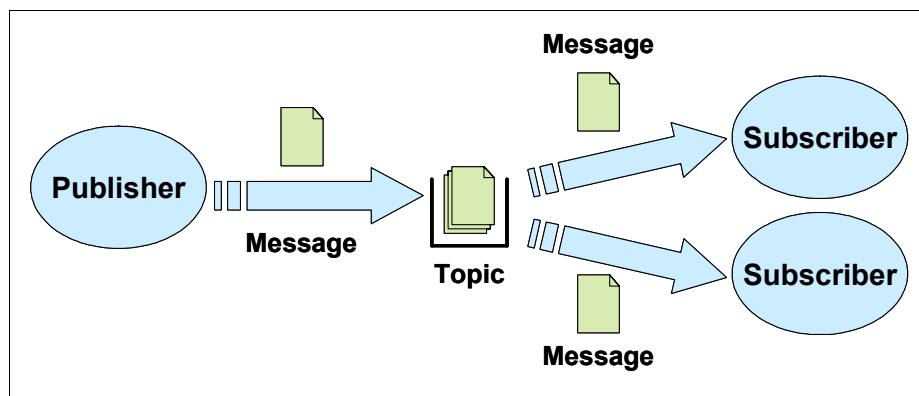


Figure 10-4 Publish/Subscribe messaging model

10.1.5 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 10-5.

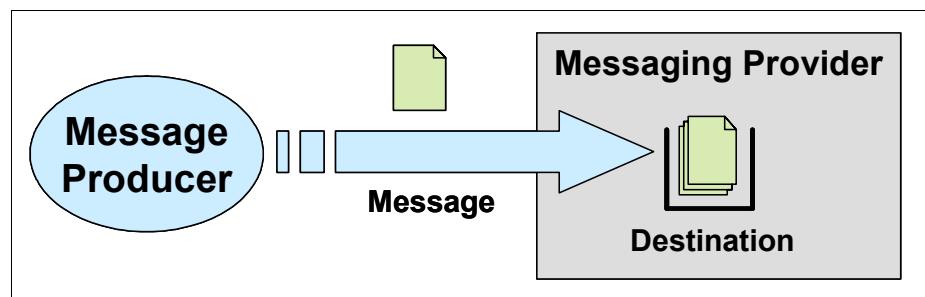


Figure 10-5 Message producer pattern

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 10-6 on page 469.

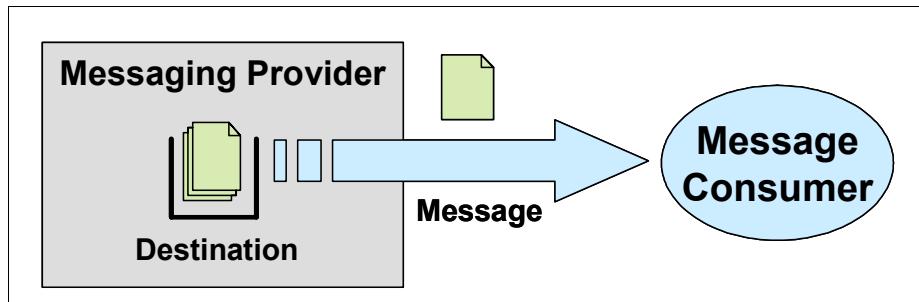


Figure 10-6 Message consumer pattern

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 10-7.

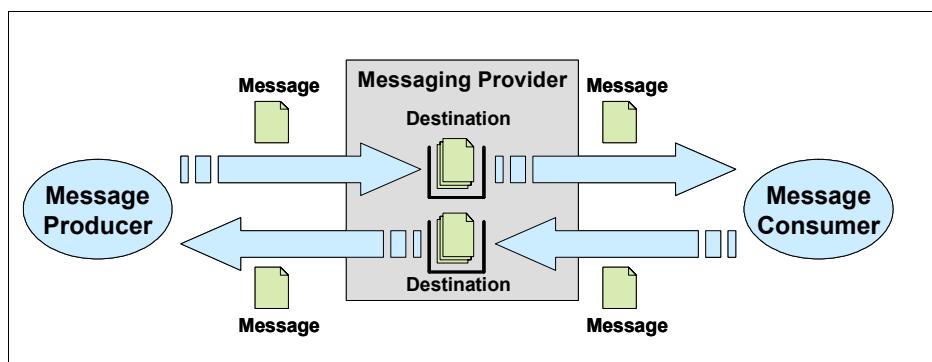


Figure 10-7 Request-reply pattern

10.2 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 specification, Version 1.1. A link for this specification is contained in 10.8, “References and resources” on page 590.

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 Section 10.2.13, “JMS and J2EE” on page 485.

10.2.1 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.

10.2.2 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.

10.2.3 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 10.1.4, “Messaging models” on page 466.

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 *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.

10.2.4 JMS administered objects

Administered objects encapsulate JMS provider-specific configuration information. They are created by an administrator and are later used at runtime 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 10-1.

Table 10-1 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 10-2.

Table 10-2 JMS destination interfaces

Common interface	Domain-specific interfaces	
	Point-to-Point	Publish/Subscribe
Destination	Queue	Topic

10.2.5 JMS and JNDI

At runtime, 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 V6 are discussed in section 10.5, “Managing WebSphere JMS providers” on page 514 and section 10.6, “Configuring WebSphere JMS administered objects” on page 526.

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 runtime. This requirement creates a dependency between the JMS client code and the runtime 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 runtime. 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 runtime, 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 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 10-1.

Example 10-1 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");
```

10.2.6 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 10-3.

Table 10-3 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 10-2.

Example 10-2 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);
```

10.2.7 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 10-4.

Table 10-4 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 10-3.

Example 10-3 Creating JMS Sessions

```
// Create a non-transacted session
Session session = conn1.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

10.2.8 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 10-8.

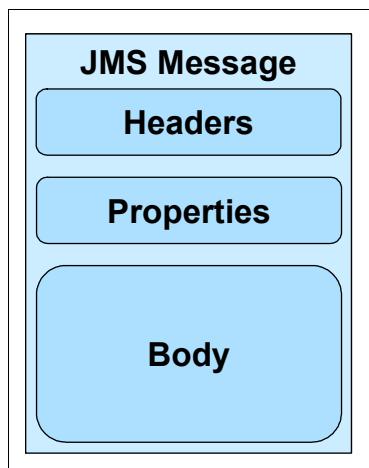


Figure 10-8 Logical format of a JMS Message

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 10-5.

Table 10-5 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.

Message type	Message body
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 which might be part of the message body. An example of a message selector is shown in Example 10-4.

Example 10-4 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 10.8, “References and resources” on page 590.

10.2.9 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 10-6.

Table 10-6 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 10-5.

Example 10-5 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);
```

10.2.10 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 10-7.

Table 10-7 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 468, 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 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 10-6.

Example 10-6 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 10-7 on page 481.

Example 10-7 The javax.jms.MessageListener interface

```
package javax.jms;

public interface MessageListener
{
    public void onMessage(Message message);
}
```

A simple class the implements the javax.jms.MessageListener interface is shown in Example 10-8.

Example 10-8 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 10-9.

Example 10-9 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 471, for more information.

10.2.11 JMS exception handling

Any runtime 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 JMSException 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 10-10.

Example 10-10 Handling a javax.jms.JMSException

```
try
{
    // Code which may throw a JMSException
}
catch (JMSException 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 10-11.

Example 10-11 The javax.jms.ExceptionListener interface

```
package javax.jms;

public interface ExceptionListener
{
    public void onException(JMSException exception);
}
```

A simple class that implements the `javax.jms.ExceptionListener` interface is shown in Example 10-12.

Example 10-12 Simple ExceptionListener implementation

```
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);
        }
    }
}
```

10.2.12 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.

10.2.13 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 section 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 10.2.12, “Application Server Facilities” on page 484.

10.3 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:

- ▶ Lifecycle management

Lifecycle 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, etc.) 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 runtime 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 ACID (Atomicity, Consistency, Isolation and Durability) 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), etc.) 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 10.8, “References and resources” on page 590.

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 10-9 on page 487.

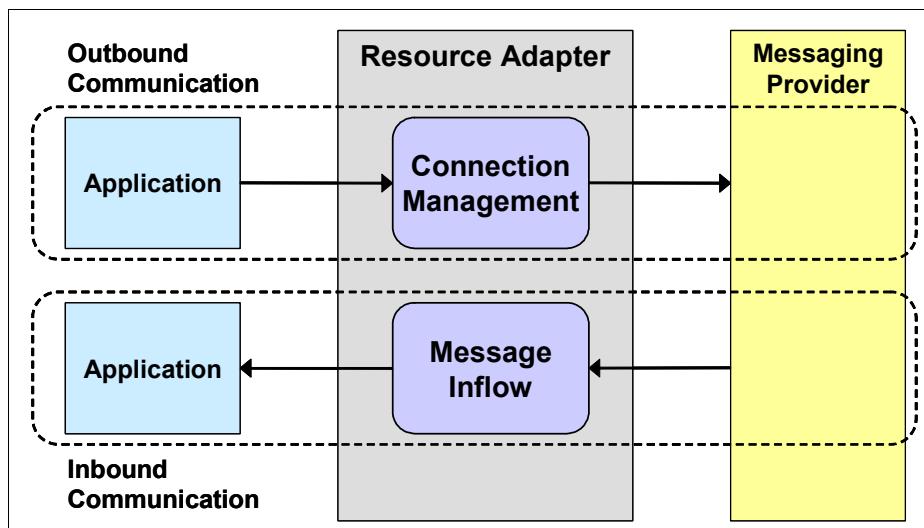


Figure 10-9 Inbound and outbound communication using a resource adapter

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

10.3.1 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 10.4.6, “Message-driven bean activation configuration properties” on page 507.

10.3.2 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.

10.3.3 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 be 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 runtime. However, it is possible to package the resource adapter within the message endpoint application.

WebSphere Application Server V6 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 runtime, 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 setup asynchronous message delivery to a message endpoint. Section 10.3.4, “JMS

ActivationSpec JavaBean” on page 491 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 10-13 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 10-13 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 10.2.4, “JMS administered objects” on page 472. In fact, for the default messaging JMS provider within WebSphere Application Server V6,

the J2EE Connector Architecture administered objects that it defines implement the relevant JMS administered object interfaces. This is shown in Example 10-14.

Example 10-14 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>
```

10.3.4 JMS ActivationSpec JavaBean

An ActivationSpec JavaBean instance encapsulates the configuration information needed to setup 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 V6 does just this, as shown in Example 10-13 on page 490.

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 V6 does just this, as shown in Example 10-14 on page 491.

- ▶ **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 V6 does just this, as shown in Example 10-13 on page 490.

- ▶ messageSelector

The JMS ActivationSpec JavaBean can optionally define a messageSelector property. JMS message selectors are discussed in “Message selectors” on page 478.

- ▶ 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 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 10.8, “References and resources” on page 590.

- ▶ 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 471, 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.

10.3.5 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 10-10 on page 495.

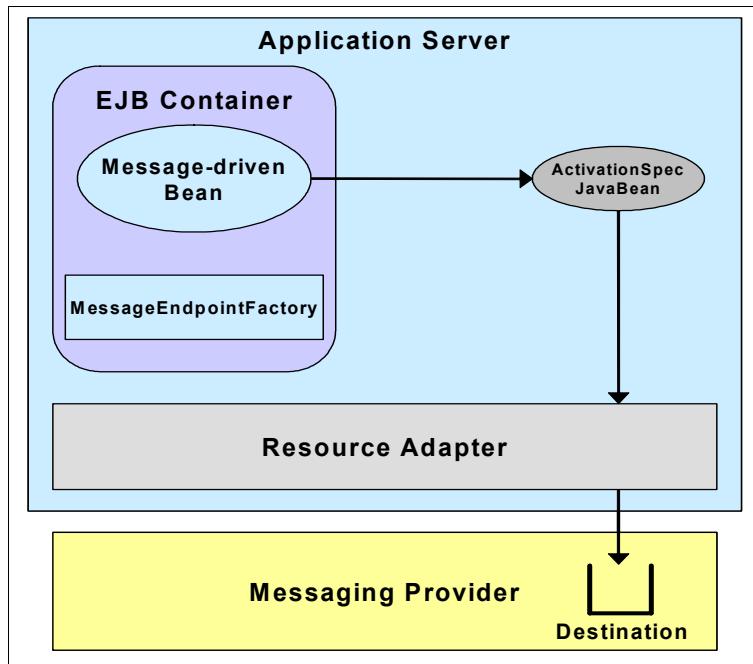


Figure 10-10 Associating an MDB with a destination using a ActivationSpec JavaBean

10.3.6 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 setup 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 10-11 on page 496.

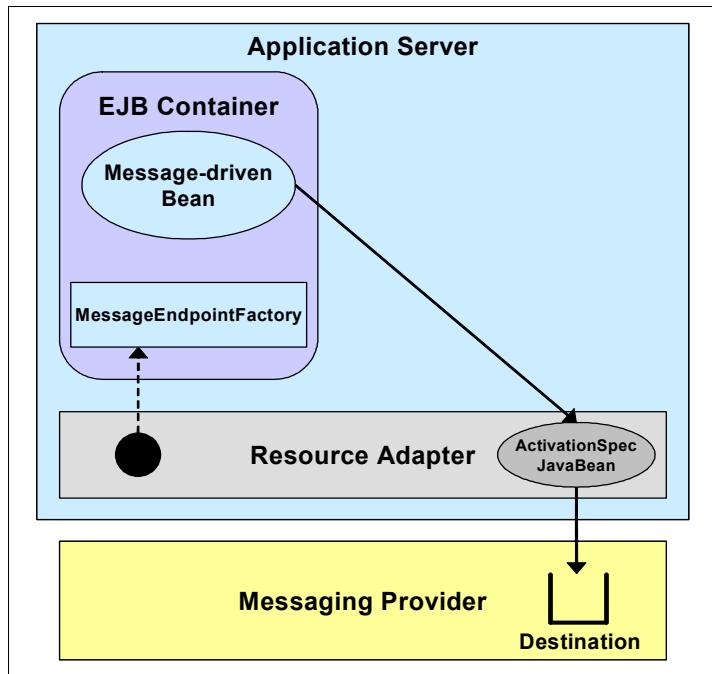


Figure 10-11 Activating a message endpoint

10.3.7 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 10-12 on page 497.

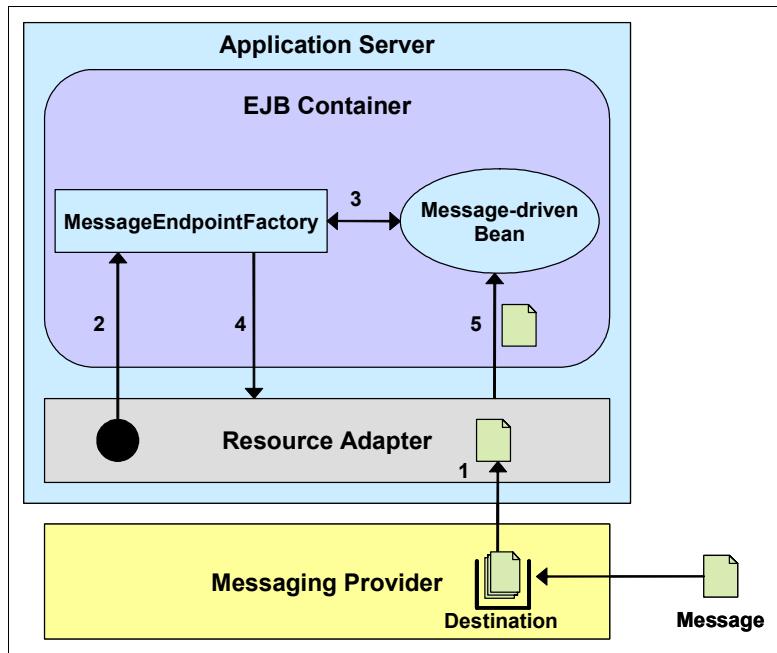


Figure 10-12 Delivering a message to a message endpoint

10.3.8 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 panels for this resource adapter. Section 10.6, “Configuring WebSphere JMS administered objects” on page 526 details the steps required to configure administered objects for the default messaging JMS provider.

10.4 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 state that is not specific to a client, for instance, 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.

10.4.1 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*.

10.4.2 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 clients perspective, the existence of the message-driven bean is completely transparent. This is shown in Figure 10-13, where the client is only able to see the messaging provider and the target destination.

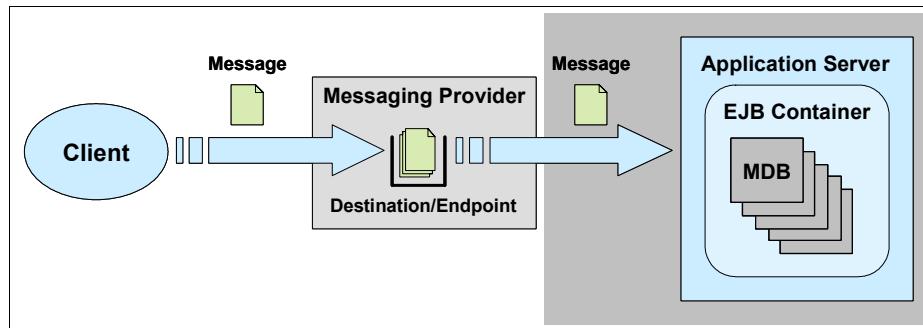


Figure 10-13 Client view of a message-driven bean

10.4.3 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 10-15.

Example 10-15 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 section 10.4.1, “Message-driven bean types” on page 498, 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 10-7 on page 481.

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 10-16.

Example 10-16 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 which 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.

10.4.4 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 10-14.

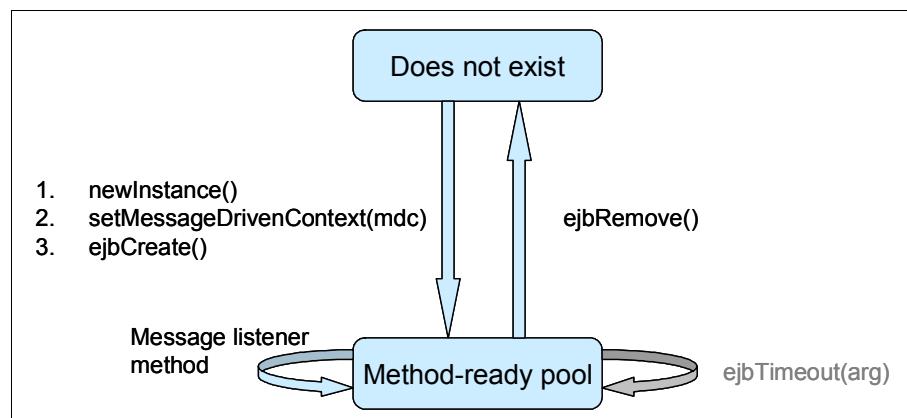


Figure 10-14 Message-driven bean life cycle

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:

- a. The EJB container invokes the `Class.newInstance()` method on the bean implementation class.
- b. The EJB container provides the new instance with its `MessageDrivenContext` reference by invoking the `setMessageDrivenContext` method.
- c. 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.

10.4.5 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 10-17.

Example 10-17 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 10-18.

Example 10-18 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 rollback transactions. The code required to use the `UserTransaction` interface within a message listener method is shown in Example 10-19.

Example 10-19 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)
        {
            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.

10.4.6 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 10.3.4, “JMS ActivationSpec JavaBean” on page 491. 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 10-20 on page 508, shows the relevant entry for the BankListener message-driven bean that is packaged as part of the WebSphereBank sample in WebSphere Application Server V6. The elements of the deployment descriptor

that are specific to messaging are shown in bold. Table 10-8 shows activation configuration properties are defined within the deployment descriptor:

Table 10-8 Activation configuration properties for the BankListener message-driven bean

Property name	Property value
destinationType	javax.jms.Queue
acknowledgeMode	Auto-acknowledge
messageSelector	JMSType = 'transfer'

Example 10-20 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>
  <messaging-type>javax.jms.MessageListener</messaging-type>
  <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-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>
```

10.4.7 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 10.3.5, “Message endpoint deployment” on page 494, the responsibility of associating a message-driven bean with a destination lies with the application deployer.

Within WebSphere Application Server V6, 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 within WebSphere Application Server V6 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 V6, 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 10-15.

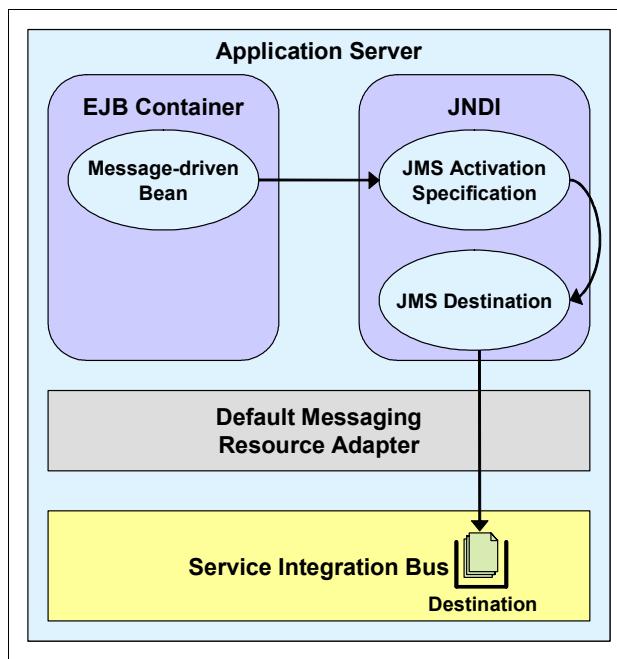


Figure 10-15 *Associating an MDB with a destination using a JMS activation specification*

The steps required to create a JMS activation specification for the default messaging JMS provider are described in “JMS activation specification configuration” on page 549.

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 10-16 on page 511. WebSphere Application Server V6 continues to use listener ports for those JMS providers that are not accessed using a resource adapter.

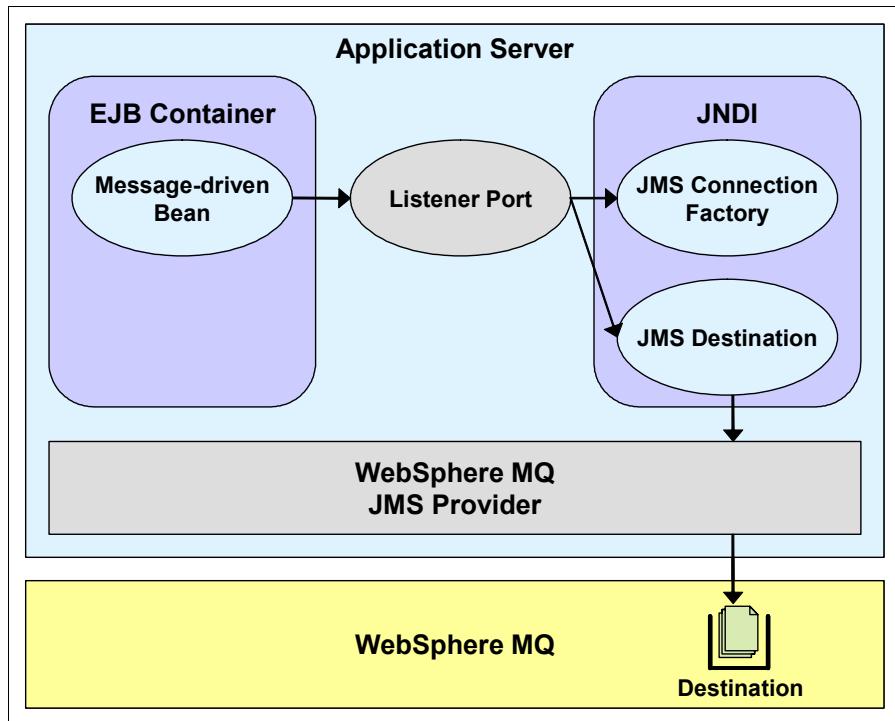


Figure 10-16 *Associating an MDB with a destination using a listener port*

The steps required to create a listener are described in 10.6.4, “Configuring listener ports” on page 568.

10.4.8 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 10-17.

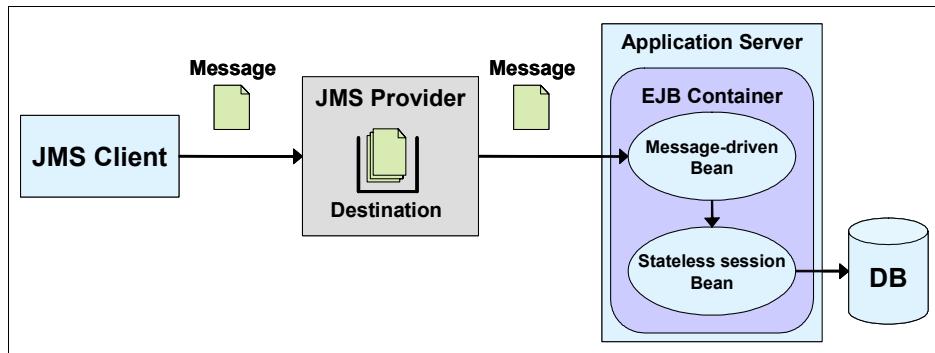


Figure 10-17 Delegating business logic to a stateless session bean

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 10-18.

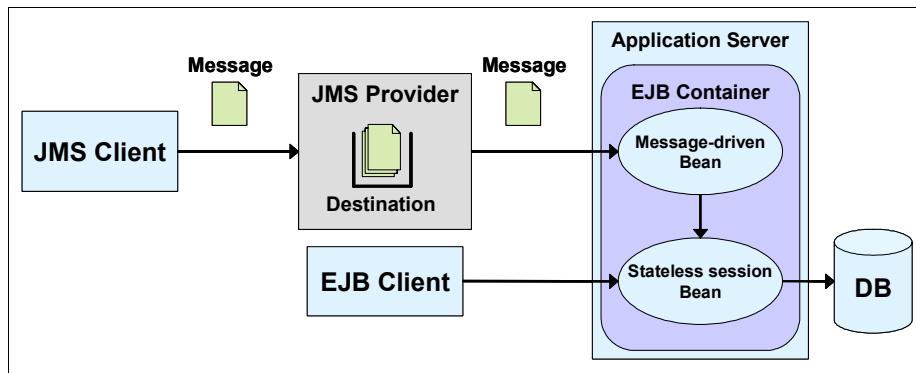


Figure 10-18 Business logic reuse

- ▶ 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 instance, 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 10.4.4, “Message-driven bean life cycle” on page 501, 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 MDB’s 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 10.4.5, “Message-driven beans and transactions” on page 502. The EJB container n

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.

10.5 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 book. For information about the V5 default messaging provider, see *IBM WebSphere Application Server V5.1 System Management and Configuration*, SG24-6195.

10.5.1 Managing the default messaging JMS provider

WebSphere Application Server V6 supplies a preconfigured J2EE Connector Architecture resource adapter implementation that can be used to communicate with a service integration bus. This resource adapter is installed as a fully-integrated component of WebSphere Application Server V6, at all levels of the WebSphere 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 panels 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 panels 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: WebSphere Application Server V6 does not require that the underlying service integration bus resources are configured, before configuring the corresponding JMS resources. However, certain fields within the default messaging JMS provider administration panels are populated with relevant service integration bus resources, if they exist. Therefore, to simplify the process of creating JMS resources for the default messaging JMS provider, it is recommended 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 panels. 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 Providers**.
2. Click **Default messaging**.
3. The properties for the Default messaging JMS provider are displayed in the main content pane of the WebSphere administrative console, as shown in Figure 10-19 on page 516.

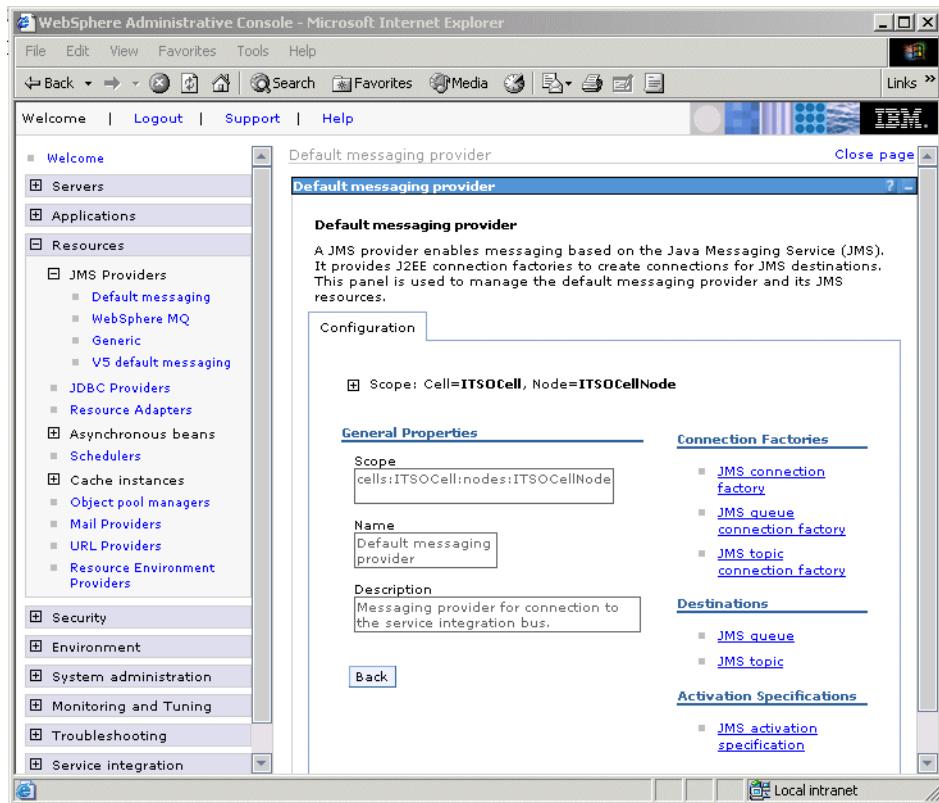


Figure 10-19 Default messaging provider configuration properties

It is worth noting, however, that the resource adapter can also be configured as a generic J2EE Connector Architecture resource adapter. However, the administration panels 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 panels. 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. 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 WebSphere Application Server V6 cell. The list of resource adapters is shown in Figure 10-20 on page 517.

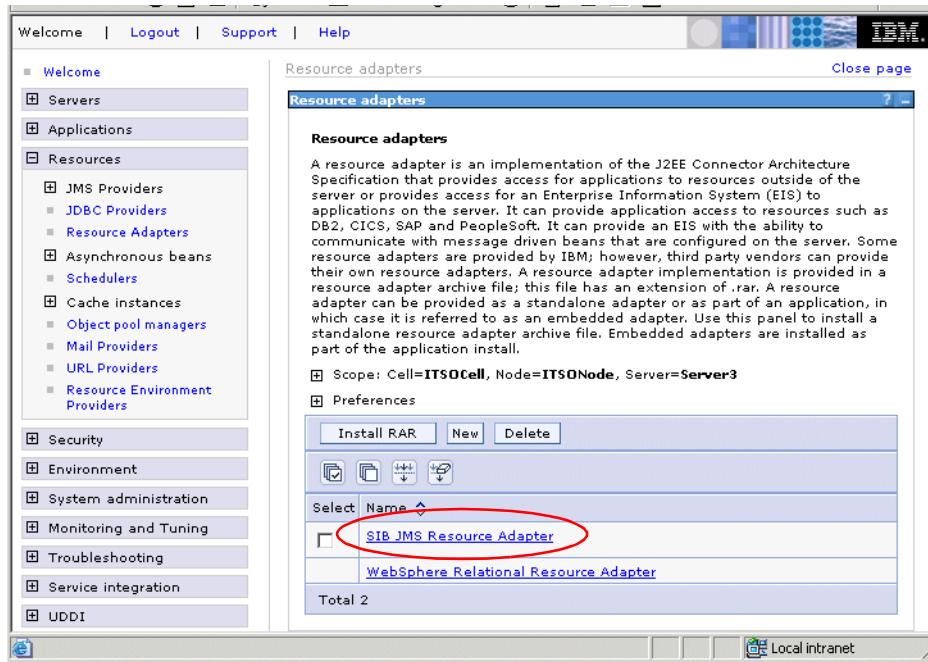


Figure 10-20 Resource adapters

4. Click **SIB JMS Resource Adapter**.
5. The properties for the resource adapter are displayed. These are shown in Figure 10-21 on page 518.

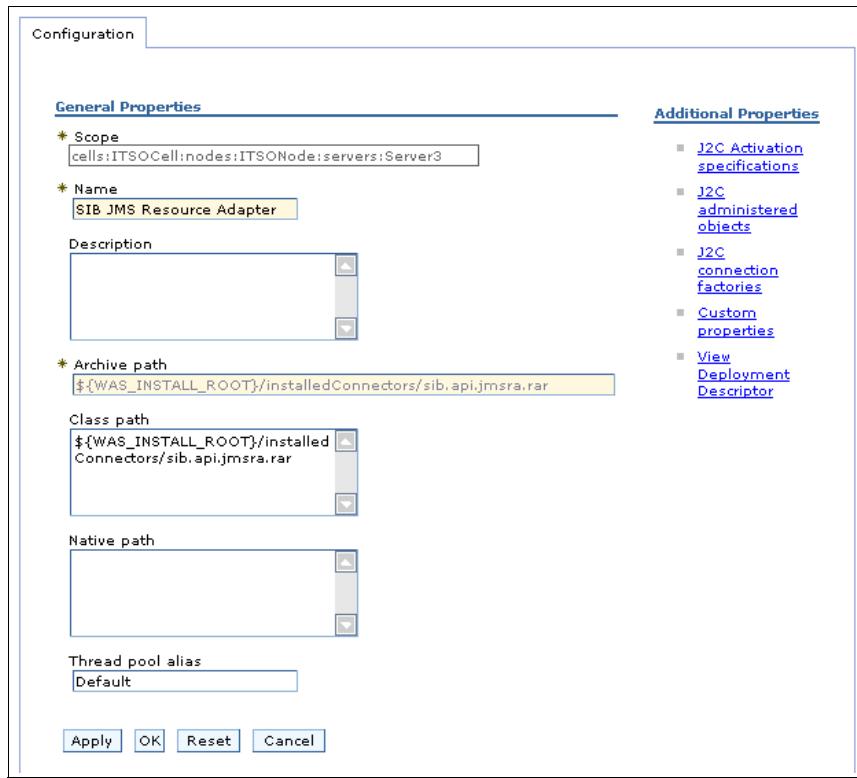


Figure 10-21 SIB JMS Resource Adapter properties

The links under the **Additional Properties** section of the configuration panel shown in Figure 10-21, 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 panels 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 panels:

- ▶ readAhead
- ▶ shareDataSourceWithCMP
- ▶ targetTransportChain

10.5.2 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 Providers**.
2. Click **WebSphere MQ**, as shown in Figure 10-22 on page 520.

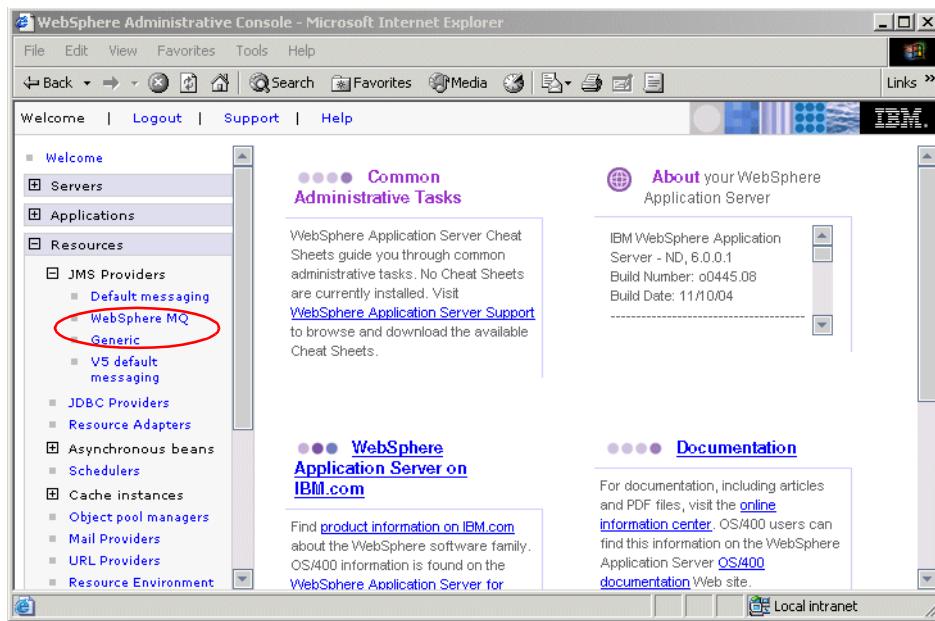


Figure 10-22 Finding the WebSphere MQ JMS provider in the navigation tree

3. The properties for the WebSphere MQ JMS provider are displayed in the main content pane of the WebSphere administrative console, as shown in Figure 10-23 on page 521.

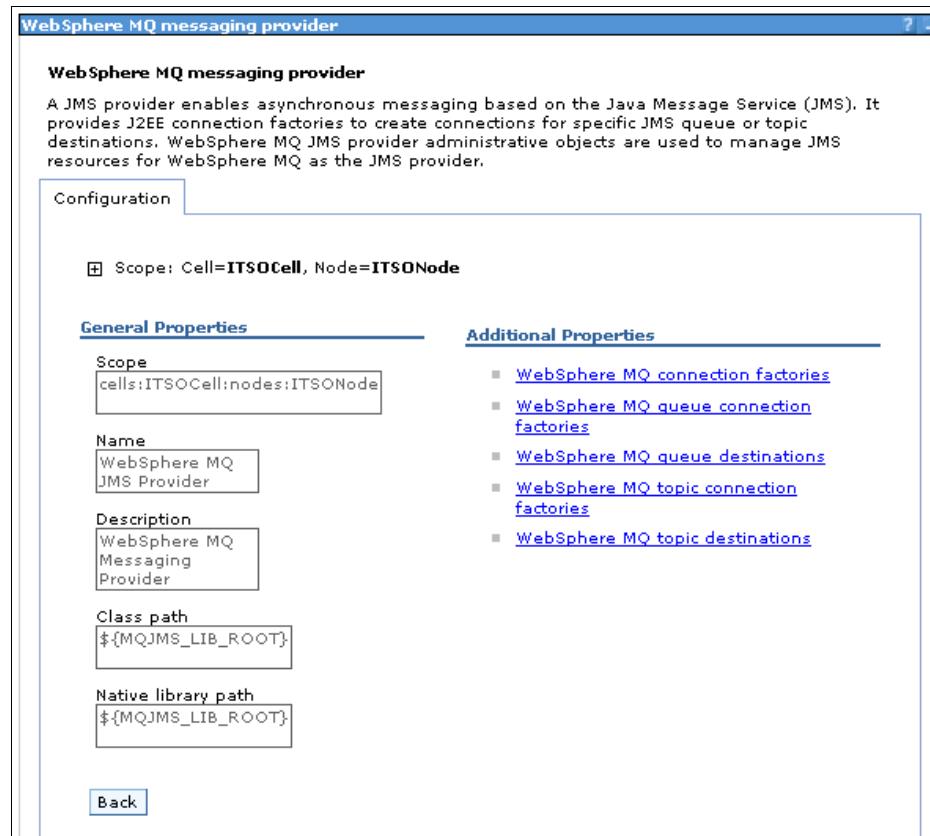


Figure 10-23 WebSphere MQ JMS provider configuration properties

Each of these properties is described in Table 10-9.

Table 10-9 WebSphere MQ JMS provider properties

Property	Description
Scope	The scope of the configured resource.
Name	The name by which the WebSphere MQ JMS provider is known for administrative purposes
Description	A description of the JMS provider, for administrative purposes within IBM WebSphere Application Server
Class path	<ul style="list-style-type: none"> ▶ The list of paths or JAR file names that form the location of the JMS provider classes ▶ Change the classpath by modifying the value of the MQJMS_LIB_ROOT WebSphere variable.

Property	Description
Native library path	<ul style="list-style-type: none"> ▶ An optional path to any native libraries (.dll's, .so's) required by the JMS provider ▶ The native path can be changed by modifying the value of the MQJMS_LIB_ROOT WebSphere variable.

Note: The MQJMS_LIB_ROOT WebSphere variable points to the WebSphere MQ JMS provider implementation libraries installed with WebSphere Application Server V6. It is recommended that these libraries are used to communicate with WebSphere MQ when using the WebSphere MQ JMS provider and that the value of the MQJMS_LIB_ROOT WebSphere variable is not modified.

10.5.3 Managing a generic JMS provider

WebSphere Application Server V6 supports the use of third party JMS providers within its runtime environment through the use 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 runtime.

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 V6 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 panels.

If the third party JMS provider is not implemented as a J2EE Connector Architecture resource adapter, it is recommended that it supports the JMS Application Server Facilities described in 10.2.12, "Application Server Facilities" on page 484.

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 V6. 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 10-24.

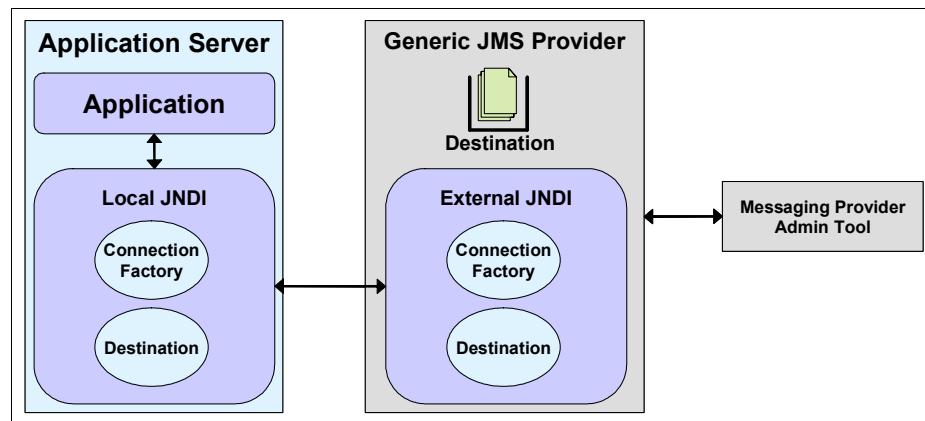


Figure 10-24 Generic JMS provider components

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 runtime 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 within WebSphere Application Server V6, 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 to WebSphere Application Server V6, use the administrative console to complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS Providers**.
2. Click **Generic**, as shown in Figure 10-25 on page 524.

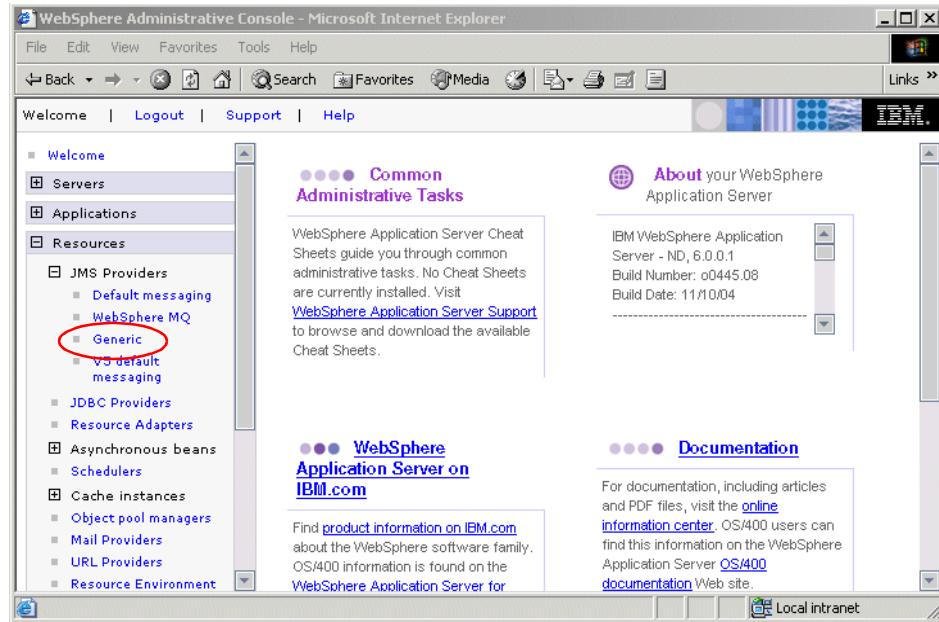


Figure 10-25 Finding the generic JMS provider in the navigation tree

3. 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.
4. Click **New** in the content pane.
5. Define the JMS provider by specifying the appropriate values in the **General Properties** section of the content pane, shown in Figure 10-26 on page 525. The properties are described in Table 10-10 on page 525.

Generic messaging provider > New

A JMS provider enables asynchronous messaging based on the Java Message Service (JMS). It provides J2EE connection factories to create connections for specific JMS queue or topic destinations. JMS provider administrative objects are used to manage JMS resources for the associated JMS provider.

Configuration

General Properties

* Scope
cells:ITSOCell:nodes:ITSOCellNode

* Name

Description

Class path

Native library path

* External initial context factory

* External provider URL

The additional properties will not be available until the general properties for this item are saved.

Additional Properties

- Custom properties
- JMS connection factories
- JMS destinations

Apply OK Reset Cancel

Figure 10-26 Generic JMS provider general properties

Table 10-10 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

Property	Description
External initial context factory	This property is the Java classname of the generic JMS providers initial context factory. For example, this would be com.swiftmq.jndi.InitialContextFactoryImpl 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 <protocol>://<host name>:<port number>. Continuing with the example above, the provider URL smqp://localhost:4001 indicates that the initial context factory connects to the SwiftMQ naming service using port 4001 on the local machine and using the sqmq protocol.

6. Click **OK**.
7. 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 10.6.5, “Configuring the generic JMS provider” on page 572.

10.6 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. Within WebSphere Application Server V6, 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.

10.6.1 Common administration properties

All of the JMS administered objects that can be configured within WebSphere Application Server V6 expose a subset of properties that are common. These properties are used by WebSphere for administrative purposes. For instance, the name and description properties are used for display purposes within the WebSphere administrative console. These common administration properties are shown in Table 10-11 on page 527.

Table 10-11 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.
Name	This property is the name by which the JMS administered object is known for administrative purposes.
JNDI name	The JNDI name os 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.

10.6.2 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 section 10.2.3, “JMS domains” on page 471 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.

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 section 10.7, “Connecting to a service integration bus” on page 576.

A brief description of the connection properties for a default messaging JMS provider connection factory are shown in Table 10-12.

Table 10-12 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 InboundBasicMessaging transport chain.

Property	Description
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 V6 environment, or if the target bus is defined within another WebSphere cell. For more information, see 10.7, “Connecting to a service integration bus” on page 576.
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 10-13.

Table 10-13 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 programmatically 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 604. 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 10-14.

Table 10-14 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 10-15 on page 531.

Table 10-15 JMS connection factory advanced messaging properties

Property	Description
Read ahead	<p>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.</p> <p>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.</p> <p>Valid values for this property are:</p> <ul style="list-style-type: none"> ▶ Default ▶ Enabled ▶ Disabled <p>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.</p>
Temporary queue name prefix	<p>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.</p>

Property	Description
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.
Share durable subscriptions	<p>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.</p> <p>Valid values for this property are:</p> <ul style="list-style-type: none"> ▶ 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 10-16.

Table 10-16 JMS connection factory advanced administrative properties

Property	Description
Component-managed authentication alias	<p>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.</p> <p>A component-managed authentication alias is only required if global security has been enabled for WebSphere Application Server. 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.</p>

Property	Description
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.
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	<p>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).</p> <p>Please refer to the WebSphere Information Center for a full description of this performance optimization.</p>
XA recovery authentication alias	<p>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.</p> <p>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.</p>

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 Providers**.
2. Click **Default messaging**.
3. Set the **Scope** for the JMS connection factory.
4. Click **JMS connection factory** in the **Connection Factories** section. A list of any existing JMS connection factories defined at this scope will be displayed. This is shown in Figure 10-27.

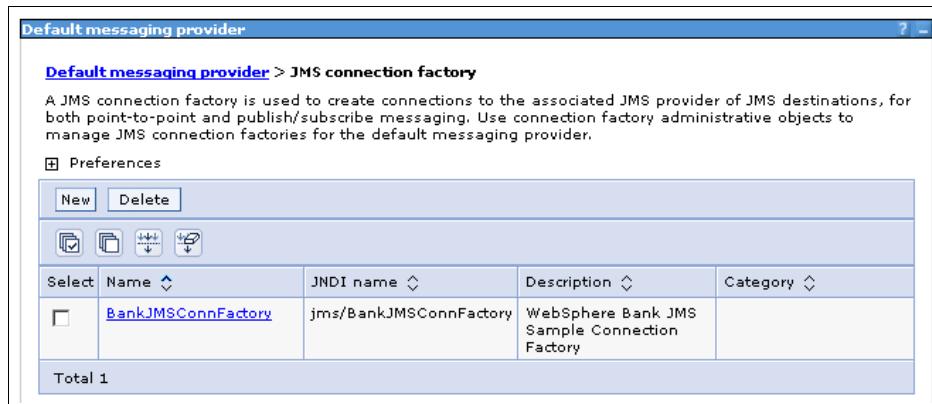


Figure 10-27 Default messaging JMS connection factory administered objects

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.

5. To create a new JMS connection factory object, click **New**. Alternatively, to change the properties of an existing JMS connection factory, click one of the connection factories displayed. Figure 10-28 on page 535 shows the top portion of the configuration page for 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 10-28, 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.

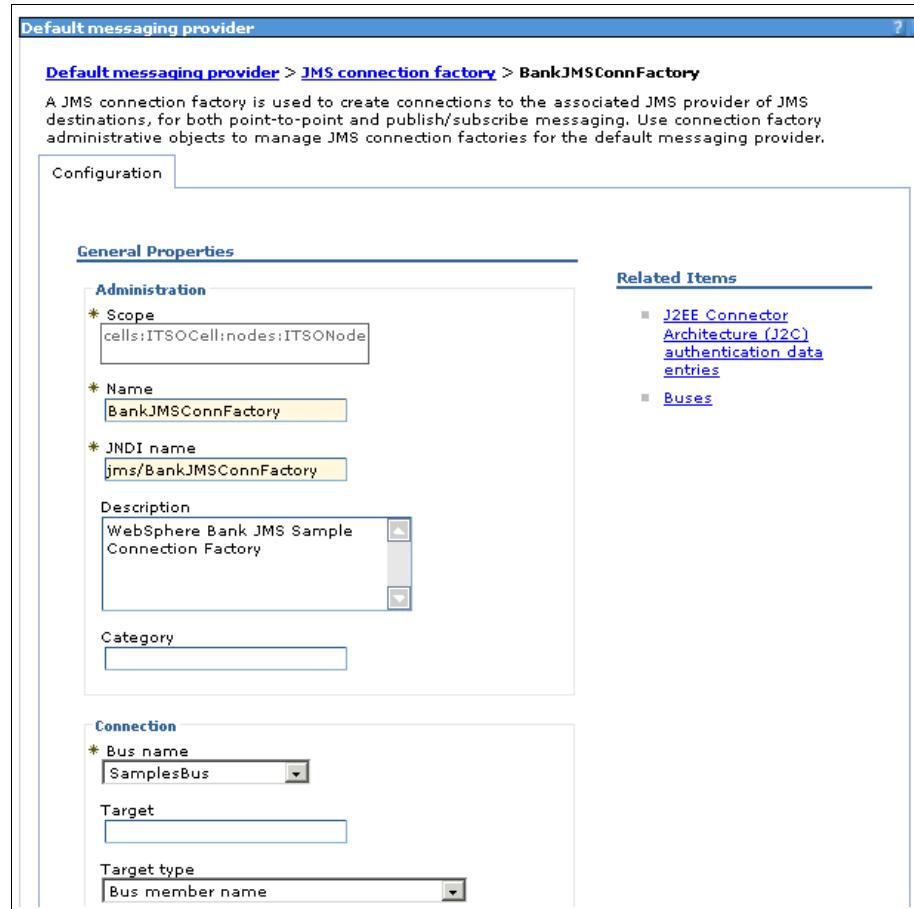


Figure 10-28 Default messaging JMS connection factory properties

6. Enter the required configuration properties for the JMS connection factory.
7. Click **OK**.
8. Save the changes and synchronize them with the nodes.
9. 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 10-17.

Table 10-17 JMS destination connection properties

Property	Description
Bus name	<p>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.</p> <p>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 section 11.1.7, “Foreign buses” on page 606 for more information.</p>
Delivery mode	<p>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.</p> <p>Valid values for this property are:</p> <ul style="list-style-type: none"> ▶ 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.

Property	Description
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 10-18 on page 537.

Table 10-18 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 633 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 wildcards characters.

Figure 10-19 on page 538, describes the wildcard characters that can be used when specifying the topic name.

Table 10-19 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 descendants of A
A//.	Selects A and all descendants of A
//*	Selects everything
A/.B	Equivalent to A/B
A/*/B	Selects all B grandchildren of A
A//B	Selects all B descendants 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 10-20.

Table 10-20 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.

Property	Description
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 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 10-21.

Table 10-21 JMS destination advanced properties

Property	Description
Read ahead	<p>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.</p> <p>Valid values for this property are:</p> <ul style="list-style-type: none"> ▶ 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. ▶ As connection factory The value of the read ahead property specified on the JMS connection factory should be used. <p>For information about the read ahead property, refer to Table 10-15 on page 531.</p>

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 Providers**.
2. Click **Default messaging**.
3. Set the **Scope** for the queue destination.

- Click **JMS queue** in the **Destinations** section. A list of any existing queue destinations defined at this scope will be displayed. This is shown in Figure 10-29.

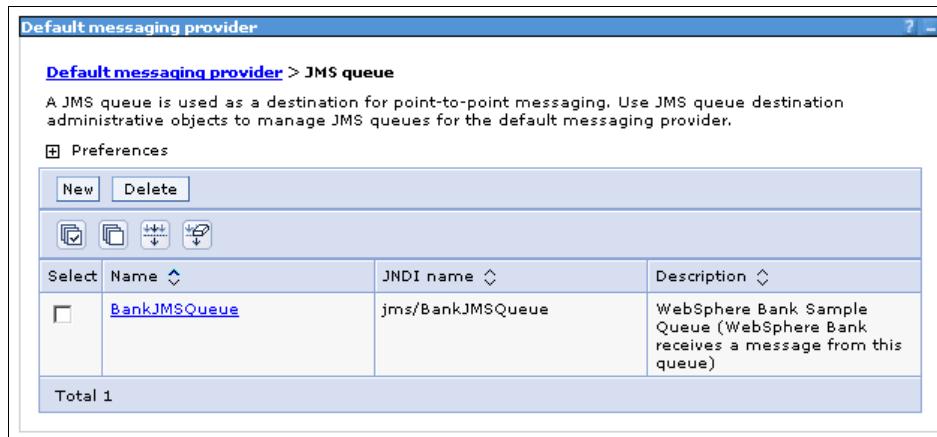


Figure 10-29 Default messaging queue destination administered objects

In this example, we already have one JMS queue destination object defined, called BankJMSQueue.

- 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 10-30 on page 541 shows 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 10-30 on page 541, 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.

General Properties

* Scope
cells:ITSOCell:nodes:ITSONode

* Name
BankJMSQueue

* JNDI name
jms/BankJMSQueue

Description
WebSphere Bank Sample Queue
(WebSphere Bank receives a message from this queue)

Bus name
Select...

* Queue name
BankJSQueue

Delivery mode
Application

Time to live

Priority

Read ahead
Enabled

Apply OK Reset Cancel

Figure 10-30 Default messaging queue destination properties

6. Enter the required configuration properties for the JMS queue destination.
7. Click **OK**.
8. Save the changes and synchronize them with the nodes.
9. 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 Providers**.
2. Click **Default messaging**.
3. Set the **Scope** for the queue destination.
4. Click **JMS topic** in the **Destinations** section. A list of any existing topic destinations defined at this scope will be displayed. This is shown in Figure 10-31 on page 542.

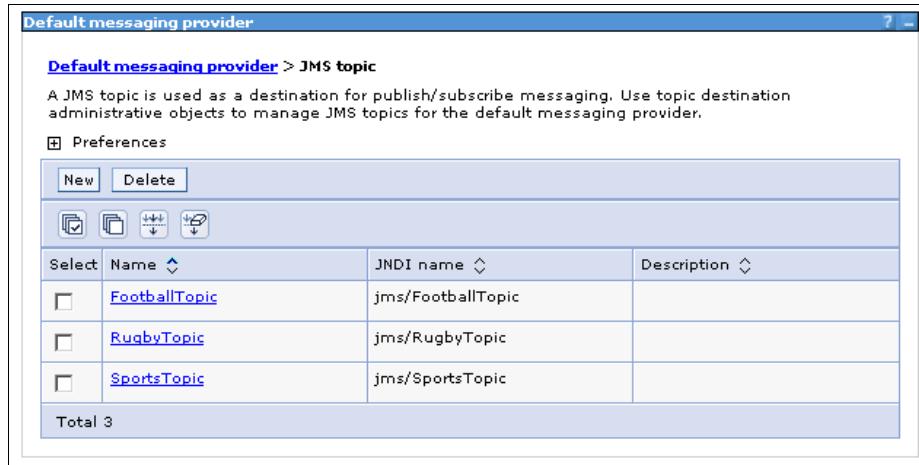


Figure 10-31 Default messaging topic destination administered objects

In this example, we already have three JMS topic destination objects defined, FootballTopic, RugbyTopic and SportsTopic.

5. 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 10-32 on page 543 shows 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 10-32 on page 543, 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.

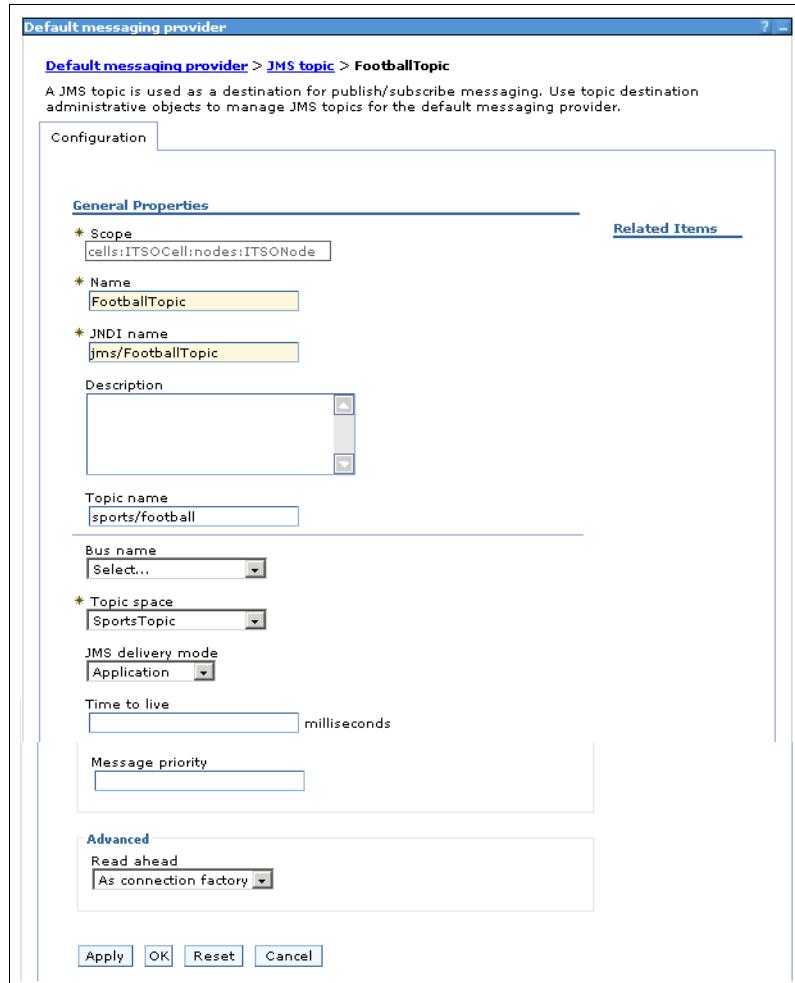


Figure 10-32 Default messaging topic destination properties

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 10-32 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 10-31 on page 542, all refer to the SportsTopic

destination on the underlying service integration bus. However, they all specify different topic names, as shown in Table 10-22.

Table 10-22 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 10-33.

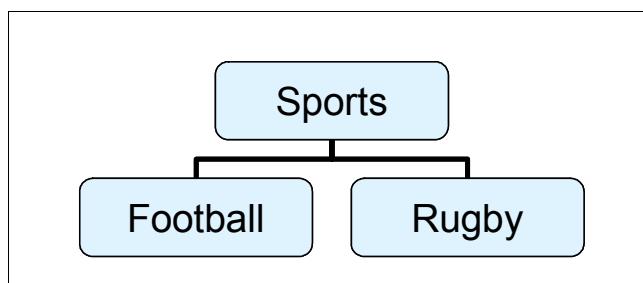


Figure 10-33 Sample sports topic hierarchy

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 537 for more information about using wild cards.

6. Enter the required configuration properties for the JMS topic destination.
7. Click **OK**.
8. Save the changes and synchronize them with the nodes.
9. For the changes to become effective, restart any application servers within the scope of the resources.

JMS activation specification properties

As we discussed in section 10.4.7, “Associating a message-driven bean with a destination” on page 509, 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 10.3.4, “JMS ActivationSpec JavaBean” on page 491. 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
- ▶ Name
- ▶ JNDI name

A description for these properties can be found in section 10.6.1, “Common administration properties” on page 526.

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 10-23 on page 546.

Table 10-23 JMS activation specification destination properties

Property	Description
Destination type	<p>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:</p> <ul style="list-style-type: none"> ▶ 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	<p>You must specify a JNDI name for the target destination.</p>
Message selector	<p>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 478 for more information.</p>
Bus name	<p>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.</p>
Acknowledge mode	<p>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:</p> <ul style="list-style-type: none"> ▶ 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.

Additional properties

The JMS activation specification for the default messaging JMS provider also exposes a group of additional properties, as described in Table 10-24 on page 547.

Table 10-24 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. An authentication alias is only required if global security has been enabled for WebSphere Application Server.
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.
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 programmatically 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 10-25 on page 548.

Table 10-25 JMS activation specification subscription durability properties

Property	Description
Subscription durability	<p>Use this property to specify whether a JMS topic subscription is durable or nondurable.</p> <p>Valid values for this property are:</p> <ul style="list-style-type: none"> ▶ Durable <p>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.</p> <ul style="list-style-type: none"> ▶ Nondurable <p>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.</p>
Subscription name	<p>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).</p> <p>A message-driven bean is not able to programmatically 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.</p>
Client identifier	<p>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.</p> <p>A message-driven bean is not able to programmatically 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.</p>

Property	Description
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 10-26 on page 549.

Table 10-26 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 10-15 on page 531 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 Providers**.
2. Click **Default messaging**.
3. Set the **Scope** for the queue destination.
4. Click **JMS activation specifications** in the **Activation specifications** section. A list of any existing activation specifications defined at this scope will be displayed. This is shown in Figure 10-34 on page 550.

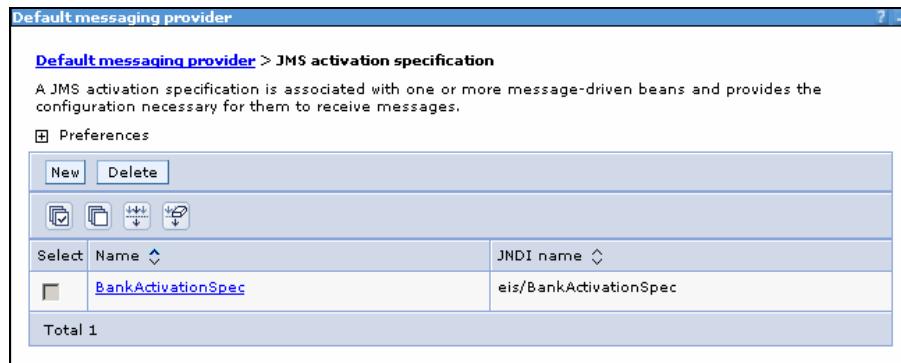


Figure 10-34 Default messaging JMS activation specifications

In this example, we already have one JMS activation specification object defined, called BankActivationSpec.

5. 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 10-32 on page 543 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**, **name** and **JNDI name**. 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 10-13 on page 490, 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 10-27.

Table 10-27 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 through, using the JMS queue defined in “JMS queue configuration” on page 539, 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 BankJSQueue destination on the SamplesBus. See Figure 10-35.

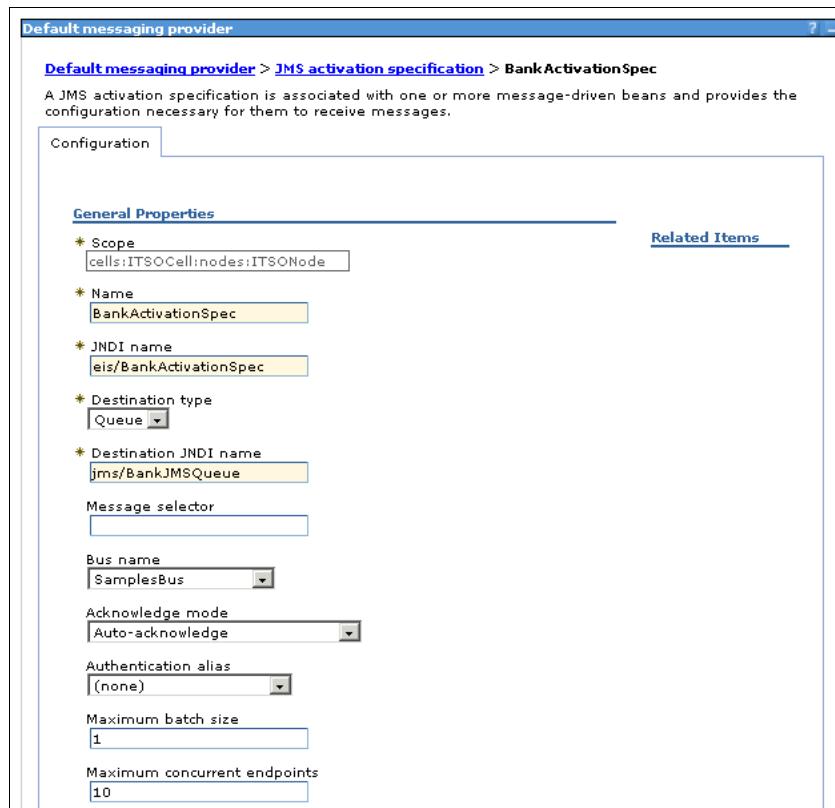


Figure 10-35 Default messaging JMS activation specification properties

6. Enter the required configuration properties for the JMS activation specification.
7. Click **OK**.
8. Save the changes and synchronize them with the nodes.
9. For the changes to become effective, restart any application servers within the scope of the resources.

10.6.3 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 V6 must be installed on the same machine.

- ▶ Client connection

If it is not possible to collocate WebSphere Application Server V6 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 instance:

- 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 section 10.5.2, “Managing the WebSphere MQ provider” on page 519, 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 section 10.2.3, “JMS domains” on page 471 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.

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 section 10.8, “References and resources” on page 590.

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 10-28.

Table 10-28 WebSphere MQ connection factory bindings connection properties

Property	Description
Transport type	Use this property to specify whether 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. 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 10-29.

Table 10-29 WebSphere MQ connection factory client connection properties

Property	Description
Transport type	Use this property to specify whether 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.
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 10-30 on page 555 describes these properties.

Table 10-30 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 as 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 10-31 describes some of the more important topic connection specific properties.

Table 10-31 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 SYSTEM.CONTROL.BROKER.QUEUE.

Property	Description
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 SYSTEM.BROKER.DEFAULT.STREAM.
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 SYSTEM.JMS.ND.SUBSCRIBER.QUEUE.
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 10-32 describes the properties of a WebSphere MQ connection factory that relate to security.

Table 10-32 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. 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.

Property	Description
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 which 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 10-33.

Table 10-33 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 (<code>session.commit</code> and <code>session.rollback</code>) 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.

Property	Description
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 WebSphere MQ browse cursor. The rescan interval property defaults to 5000.
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 Providers**.
2. Click **WebSphere MQ**.
3. Set the **Scope** for the connection factory.
4. Click **WebSphere MQ connection factories** in the **Additional Properties** section. A list of any existing connection factories defined at this scope will be displayed. This is shown in Figure 10-36.



Figure 10-36 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.

- To create a new connection factory object, click **New**. Alternatively, to change the properties of an existing connection factory, click one of the connection factories displayed. Figure 10-37 on page 560 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 kaa5070.

- **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.SVRCNN.

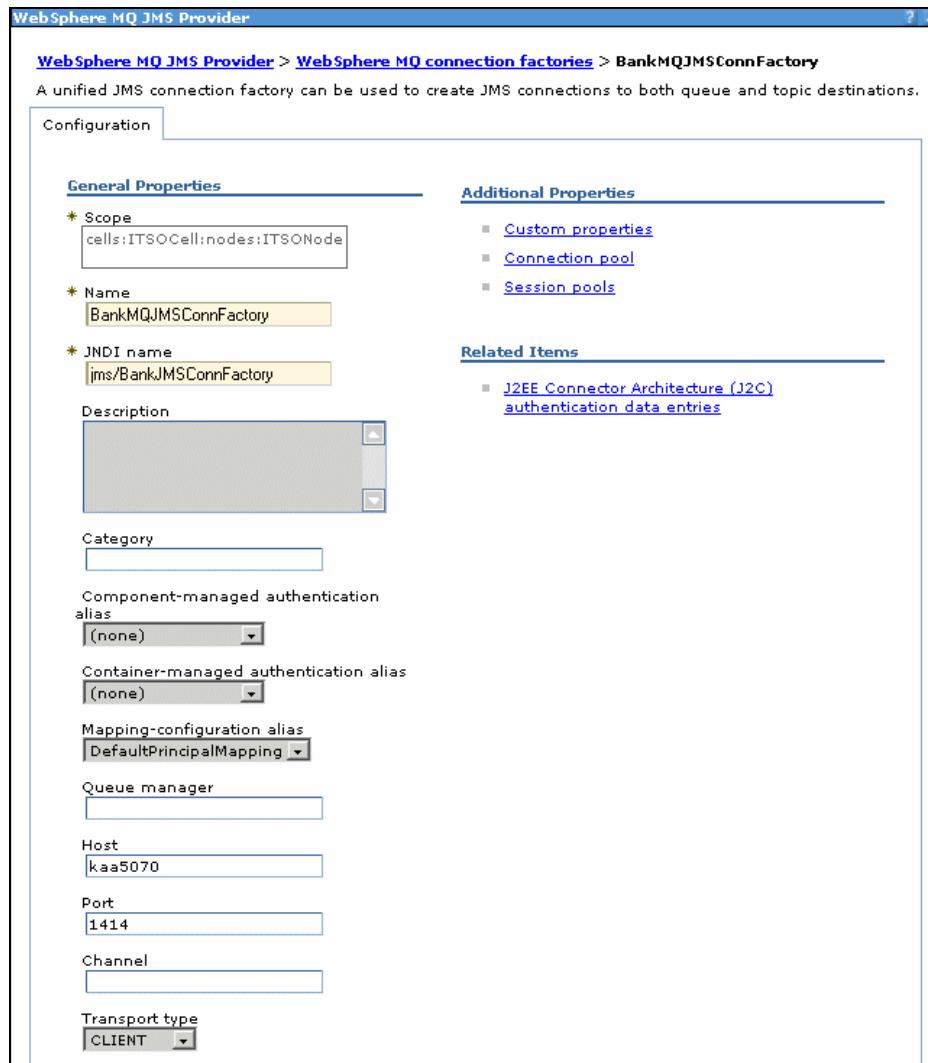


Figure 10-37 WebSphere MQ connection factory properties

6. Enter the required configuration properties for the WebSphere MQ connection factory.
7. Click **OK**.
8. Save the changes and synchronize them with the nodes.
9. 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 10-34.

Table 10-34 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).
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.

Property	Description
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 10-35.

Table 10-35 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.

Topic specific destination properties

The properties specific to WebSphere MQ topic destination objects are shown in Table 10-36.

Table 10-36 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 10-37.

Table 10-37 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, 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 Providers**.
2. Click **WebSphere MQ**.
3. Set the **Scope** for the queue destination.
4. Click **WebSphere MQ queue destinations** in the **Additional Properties** section. A list of any existing queue destinations defined at this scope will be displayed. This is shown in Figure 10-38 on page 564.

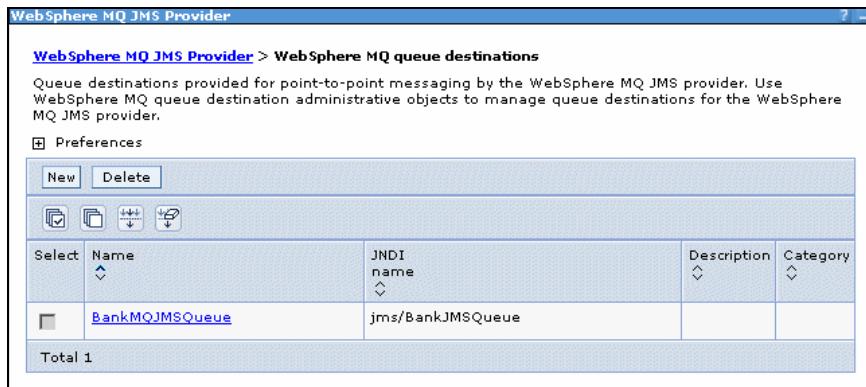


Figure 10-38 WebSphere MQ queue destination administered objects

In this example, we already have one WebSphere MQ queue destination object defined, called BankMQJMSQueue.

5. 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 10-39 on page 565 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**. In Figure 10-39, the value specified for the Base queue name property is BankJSQueue. This must match the name of the queue defined on the WebSphere MQ queue manager to which we are connecting.

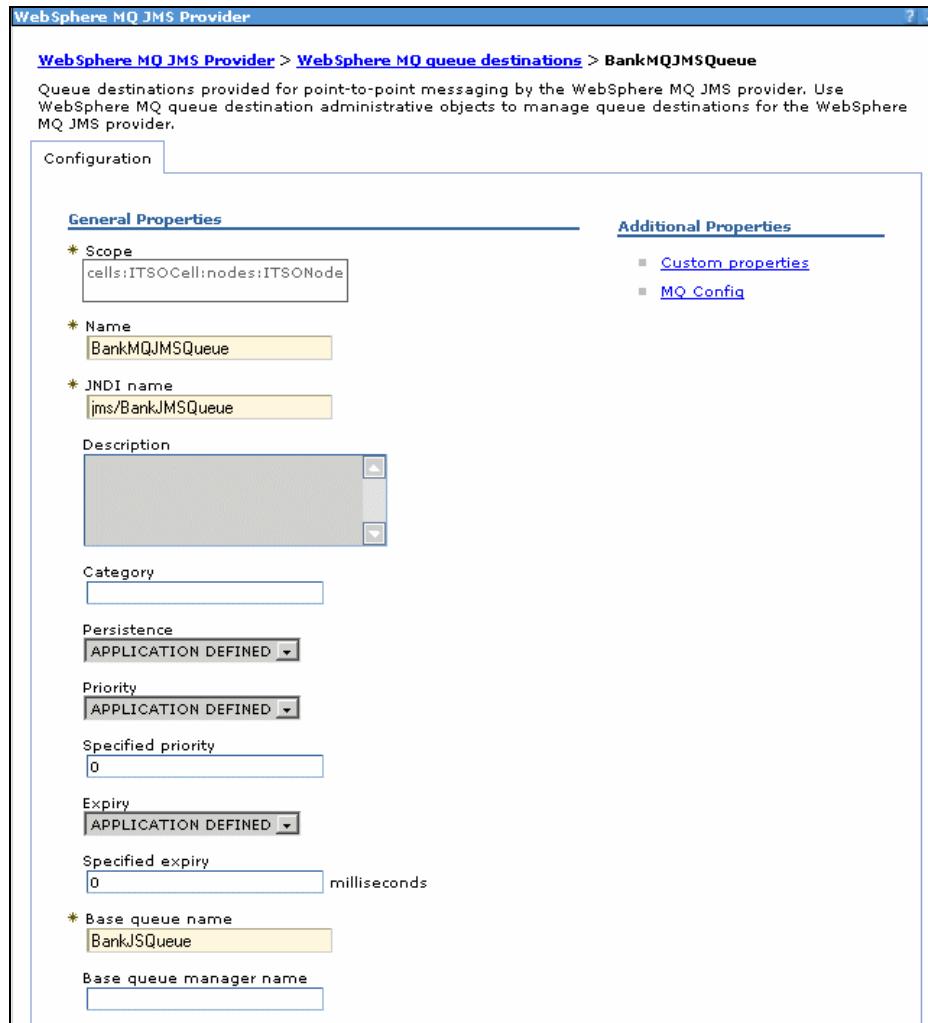


Figure 10-39 WebSphere MQ queue destination properties

6. Enter the required configuration properties for the WebSphere MQ queue destination.
7. Click **OK**.
8. Save the changes and synchronize them with the nodes.
9. 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 Providers**.
2. Click **WebSphere MQ**.
3. Set the **Scope** for the queue destination.
4. Click **WebSphere MQ topic destinations** in the **Additional Properties** section. A list of any existing topic destinations defined at this scope will be displayed. This is shown in Figure 10-40.

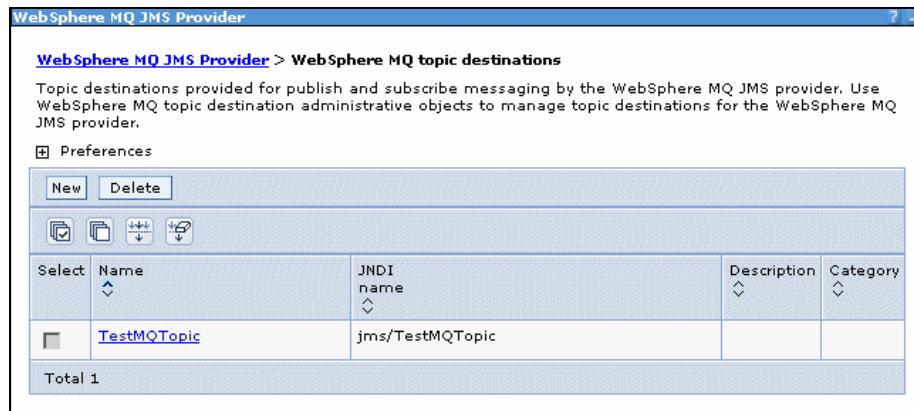


Figure 10-40 WebSphere MQ topic destination administered objects

In this example, we already have one WebSphere MQ topic destination object defined, called TestMQTopic.

5. 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 10-41 on page 567 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 10-41, the value specified for the Base topic name property is **TestTopic**. This must match the name of the topic defined on the broker.

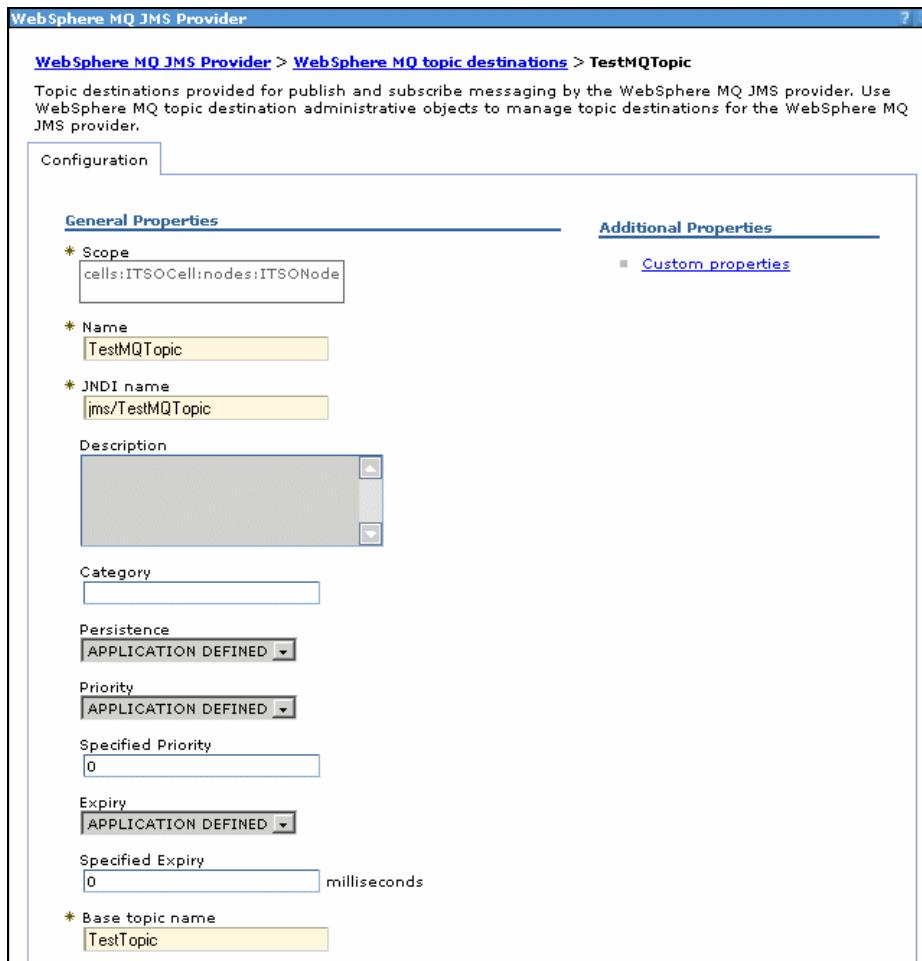


Figure 10-41 WebSphere MQ topic destination properties

6. Enter the required configuration properties for the WebSphere MQ topic destination.
7. Click **OK**.
8. Save the changes and synchronize them with the nodes.
9. For the changes to become effective, restart any application servers within the scope of the resources.

10.6.4 Configuring listener ports

As discussed in section 10.4.7, “Associating a message-driven bean with a destination” on page 509, 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 10-42.

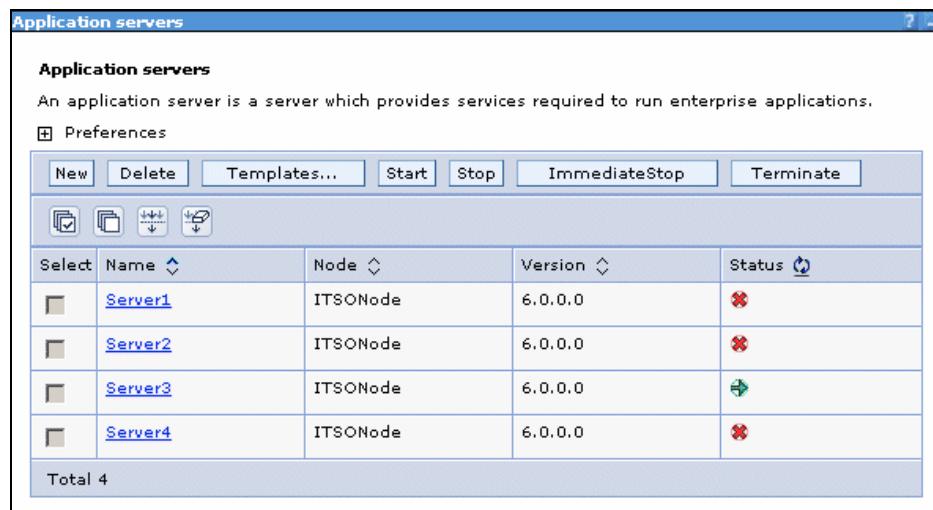


Figure 10-42 Application servers defined within the cell

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 10-43 on page 569.

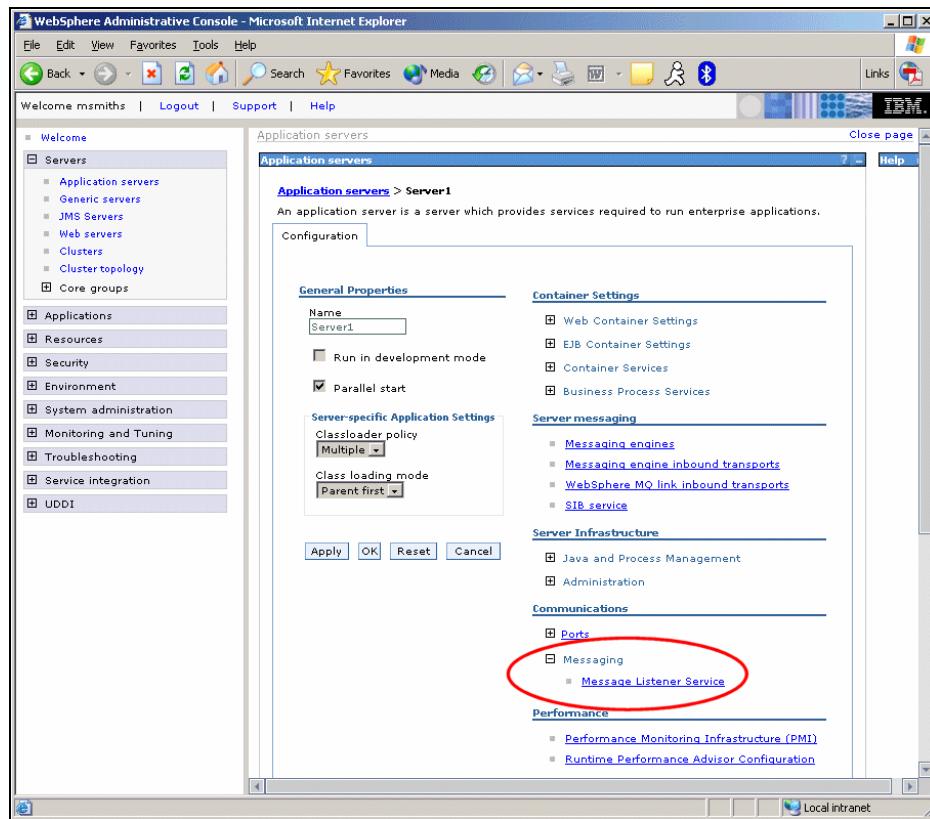


Figure 10-43 Message listener service link

7. The configuration properties for the message listener service will be displayed.
8. Click **Listener Ports**. A list of listener ports that are currently defined for this application server will be displayed. This is shown in Figure 10-44 on page 570.

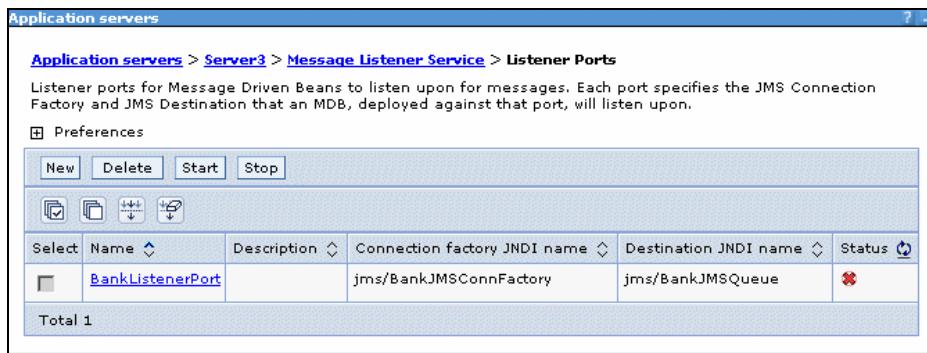


Figure 10-44 Listener ports

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 10-45 on page 571 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.

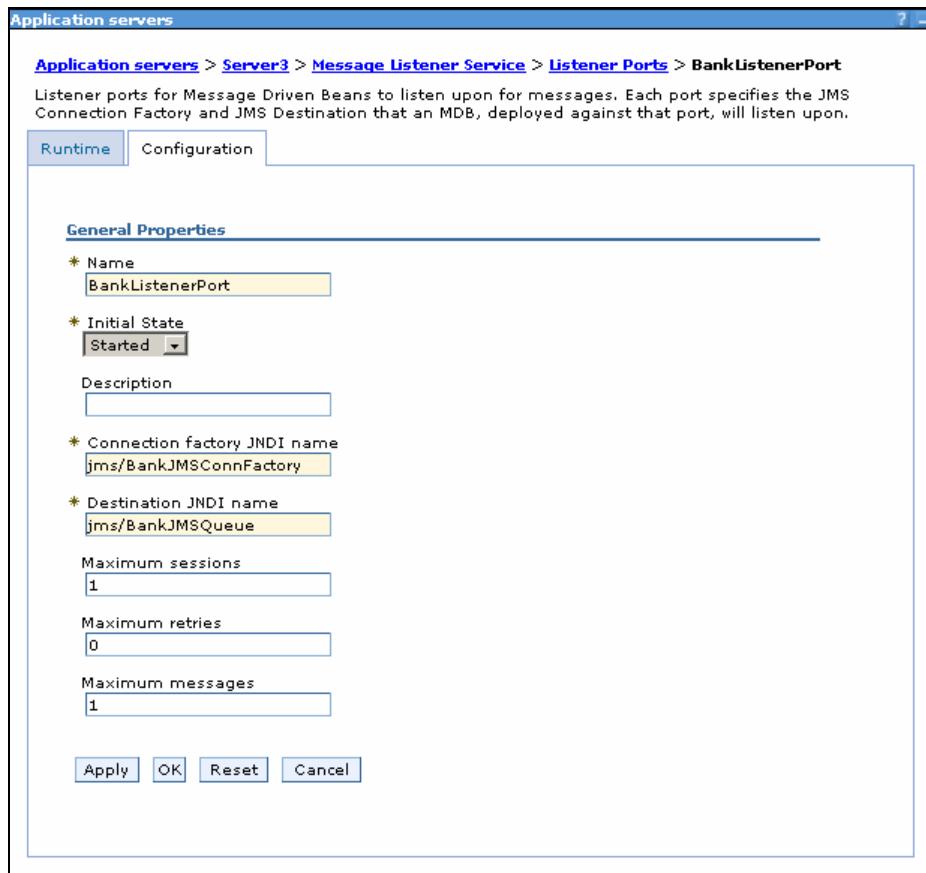


Figure 10-45 Listener port properties

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 through, using the WebSphere MQ connection factory defined in “WebSphere MQ connection factory configuration” on page 558 and the WebSphere MQ queue defined in “WebSphere MQ queue destination configuration” on page 563, 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 kaa5070 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 kaa5070.

10.6.5 Configuring the 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 configure JMS connection factories and JMS destinations for a generic JMS provider.

JMS connection factory configuration

To configure a JMS connection factory for a generic JMS provider, complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS Providers**.
2. Click **Generic**.
3. Set the scope to the level at which the generic JMS provider has been configured. The generic JMS provider should appear in the list providers defined at this level.
4. Select the generic JMS provider for which you want to configure the JMS connection factory.
5. Click **JMS connection factories** in the **Additional Properties** section to display a list of the connection factories that have been defined for the generic JMS provider.
6. To create a new connection factory object, click **New**. Alternatively, to change the properties of an existing connection factory, click one of the connection factories displayed. Figure 10-46 on page 573 shows the configuration page for a connection factory object.

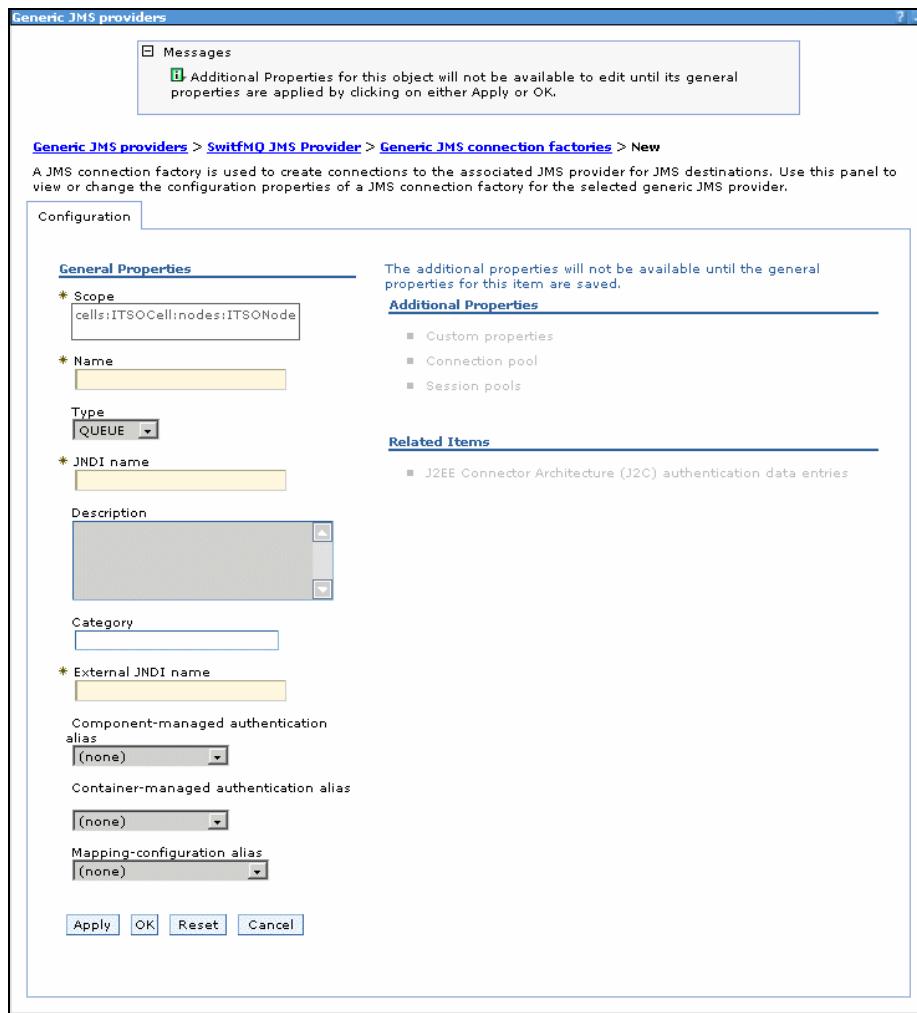


Figure 10-46 Generic JMS provider connection factory configuration panel

- Enter the required configuration properties for the JMS connection factory. The common properties are described in 10.6.1, “Common administration properties” on page 526. The properties specific to the generic JMS connection factory object are shown in Table 10-38 on page 573.

Table 10-38 Generic JMS provider connection factory properties

Property	Description
Type	Use this property to specify whether the connection factory is for JMS queues (QUEUE) or JMS topics (TOPIC).

Property	Description
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.

8. Click **OK**.
9. Save the changes and synchronize them with the nodes.
10. 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

To configure a JMS destination for a generic JMS provider, complete the following steps:

1. In the navigation tree, expand **Resources** → **JMS Providers**.
2. Click **Generic**.
3. Set the scope to the level at which the generic JMS provider has been configured. The generic JMS provider should appear in the list providers defined at this level.
4. Select the generic JMS provider for which you want to configure the JMS destination.
5. Click **JMS destinations** in the **Additional Properties** section to display a list of the destinations that have been defined for the generic JMS provider.
6. To create a new destination, click **New**. Alternatively, to change the properties of an existing destination, click one of the destinations displayed.
Figure 10-47 on page 575 shows the configuration page for a destination object.

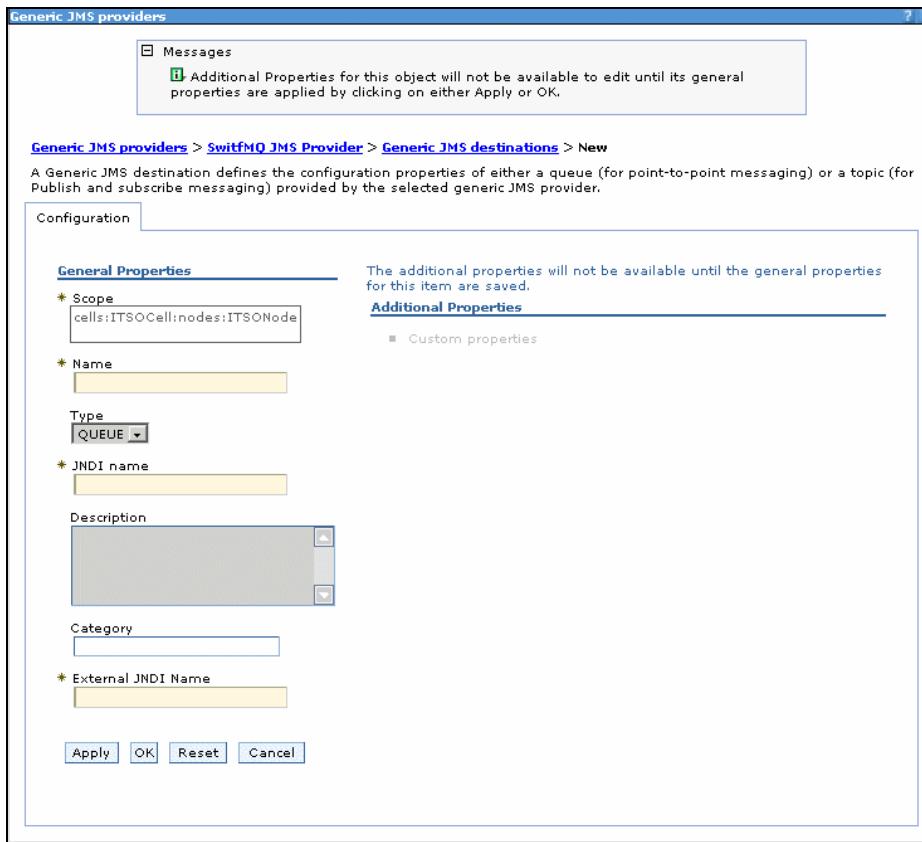


Figure 10-47 Generic JMS provider destination configuration panel

- Enter the required configuration properties for the JMS destination. The common properties are described in 10.6.1, “Common administration properties” on page 526. The properties specific to the generic JMS destination object are shown in Table 10-39.

Table 10-39 Generic JMS provider destination properties

Property	Description
Type	Use this property to specify whether the connection factory is for JMS queues (QUEUE) or JMS topics (TOPIC).
External JNDI name	Define the JNDI name used to bind the JMS connection factory into the name space of the messaging provider.

- Click **OK**.

9. Save the changes and synchronize them with the nodes.
10. For the new destination to be bound into the JNDI name space at the correct scope, restart the relevant application servers.

10.7 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 effect 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.

10.7.1 JMS client runtime environment

Regardless of the environment in 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 Application Server V6 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 in which it is executing.

- ▶ Clients running inside of WebSphere Application Server V6

When the connection factory is executing within the WebSphere Application Server V6 environment, it is able to communicate with components of the

WebSphere runtime 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 V6

When the connection factory is executing outside of the WebSphere Application Server V6 environment, or in a WebSphere Application Server V6 environment in 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 10-48 on page 578.

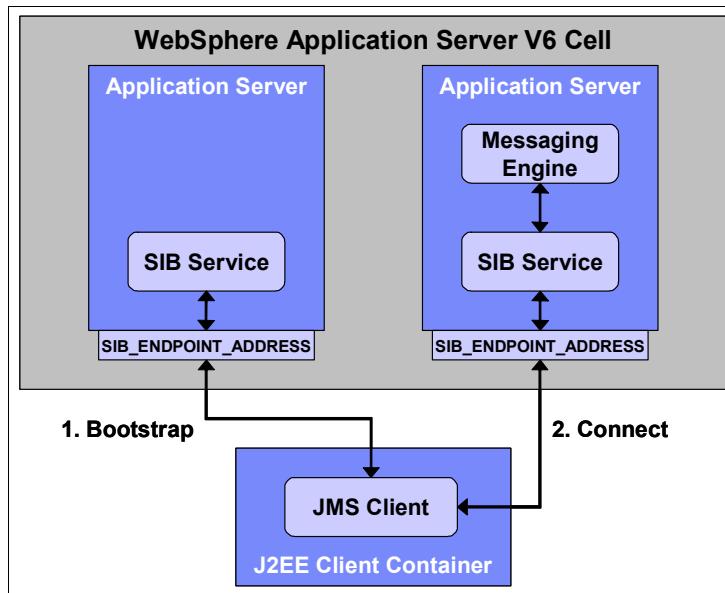


Figure 10-48 Using a bootstrap server with a messaging engine

Use the **provider endpoints** property to configure the bootstrap servers to which a connection factory can connect. See Figure 10-28 on page 535.

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 `hostname` 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.

10.7.2 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 10-49.

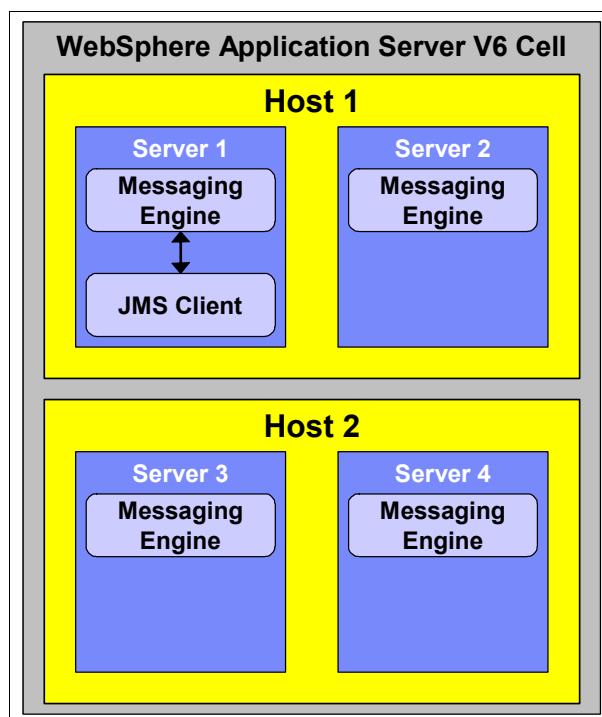


Figure 10-49 In-process connection for a JMS client and a messaging engine

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 10-50.

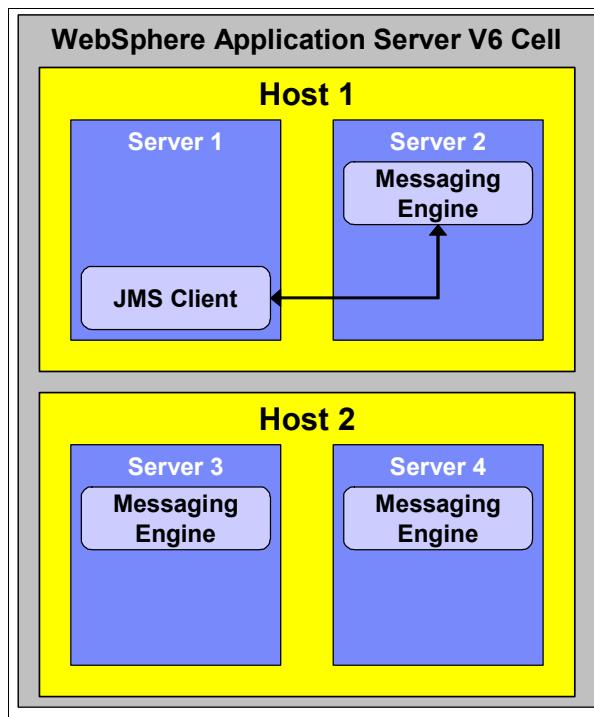


Figure 10-50 Remote connection on the same host

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 10-51.

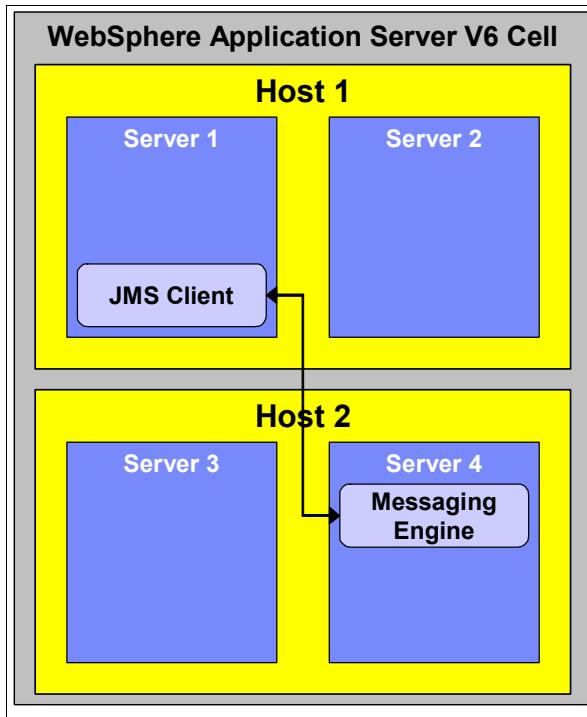


Figure 10-51 Remote connection on a different host

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 10-21.

Example 10-21 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 which cannot be contacted using the target transport chain specified, so a final selection is made only from those messaging engines which 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 580, is used.

- ▶ **Cluster**

The set of available messaging engines for the target service integration bus I 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 on the within the same application server process as the JMS client or bootstrap server.

To see how the value of the connection proximity property effects the messaging engine selection process, consider the configuration shown in Figure 10-52. All of the messaging engines shown in Figure 10-52 exist within the same service integration bus.

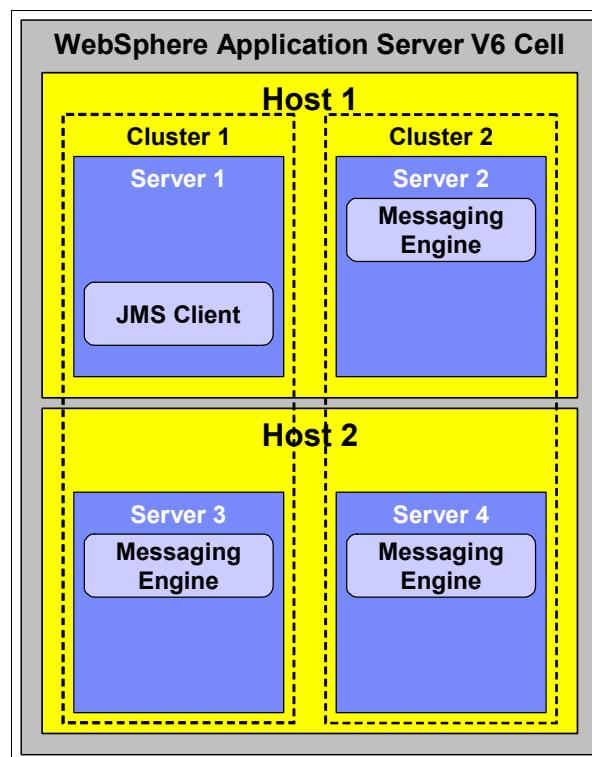


Figure 10-52 Sample topology for a service integration bus

The effect of the value of the connection proximity property on messaging engine selection is described in Table 10-40 on page 585.

Table 10-40 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 580.
Cluster	The JMS client connects 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 which 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 10-53 on page 587. All of the messaging engines shown in Figure 10-53 exist the same service integration bus.

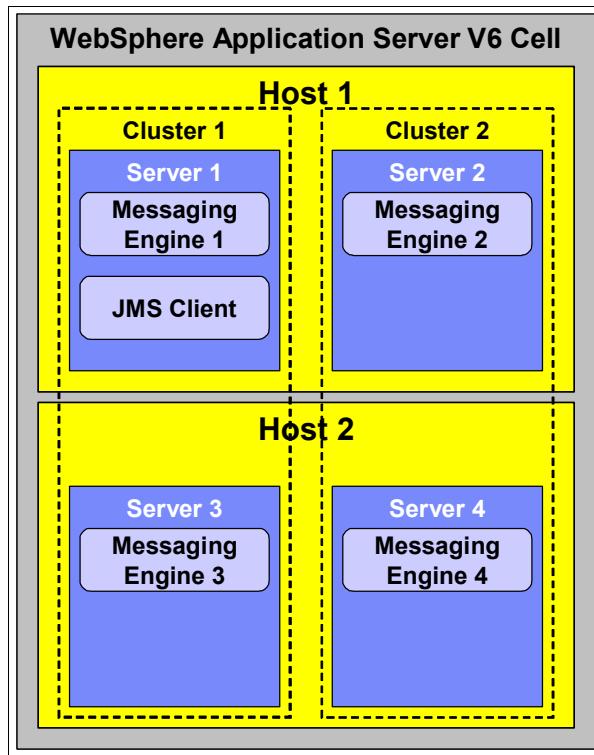


Figure 10-53 Sample topology for a service integration bus

The effect of the value of the connection proximity property on messaging engine selection is described in Table 10-41.

Table 10-41 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	

10.7.3 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 10.7.2, “Controlling messaging engine selection” on page 579.

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 10-54 on page 589.

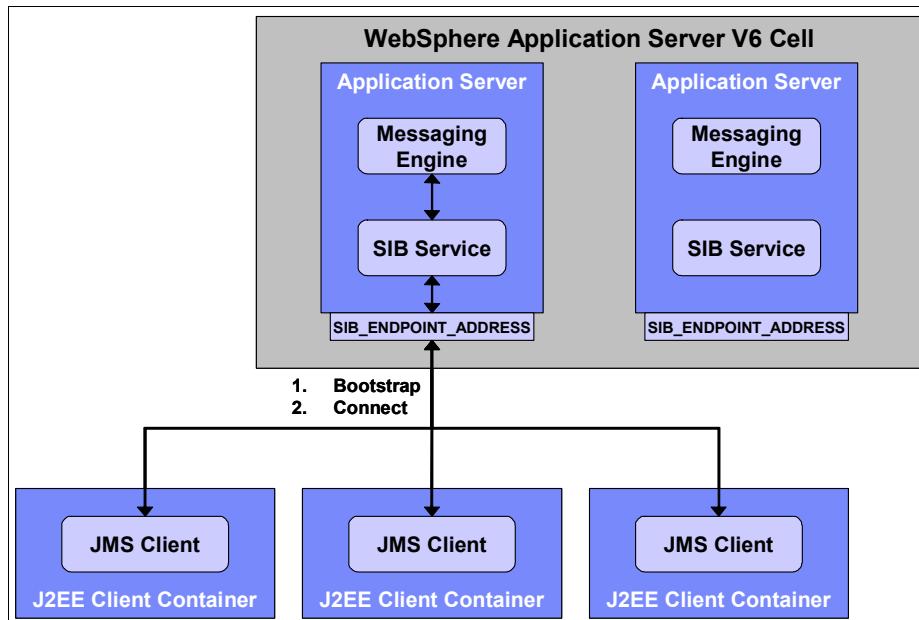


Figure 10-54 Bootstrapped JMS clients connecting to a single messaging engine

A solution to this problem is to make use of a dedicated bootstrap server that is not a 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 10-55 on page 590.

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.

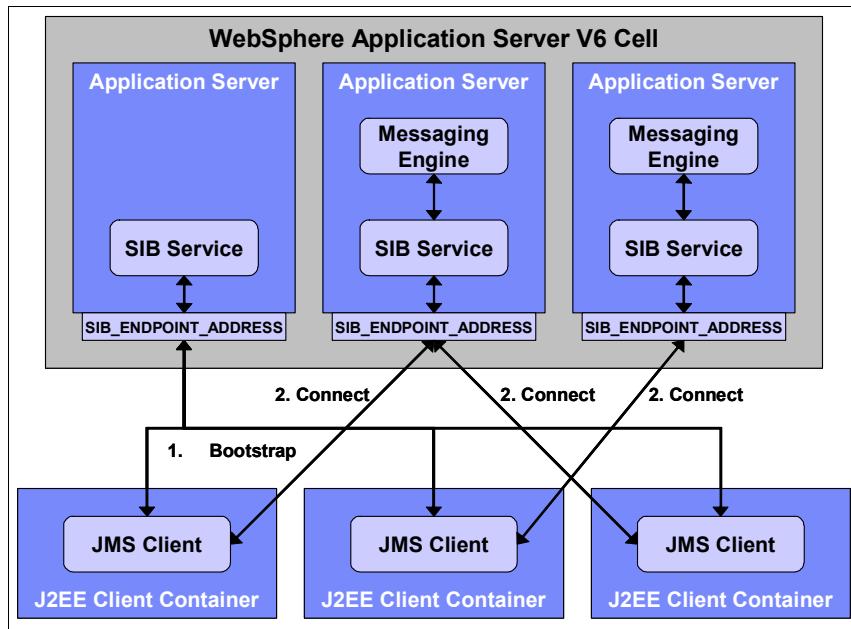


Figure 10-55 Load balancing of connections for bootstrapped JMS clients

10.8 References and resources

These documents and web sites are also relevant as further information sources:

- ▶ 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/manuals/crosslatest.html>
- ▶ Java Message Service (JMS)
<http://java.sun.com/products/jms>

- ▶ *Enterprise Messaging Using JMS and WebSphere* (Kareem Yusuf), Prentice Hall, ISBN: 0-13-146863-4
- ▶ *Java Message Service* (Monson-Haefel, Chappell), O'Reilly, ISBN: 0-596-00068-5
- ▶ *Professional JMS* (Grant, Kovacs, et al), Wrox Press Inc., ISBN: 1861004931
- ▶ *Enterprise JavaBeans, Fourth Edition* (Monson-Haefel, Burke, Labourey), O'Reilly, ISBN: 0-596-00530-X
- ▶ *EJB Design Patterns* (Marinescu), Wiley, ISBN: 0471208310



Default messaging provider

WebSphere Application Server V6 introduces 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 V6. We cover:

- ▶ Concepts and architecture
- ▶ Runtime 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

11.1 Concepts and architecture

The service integration bus provides a managed communications framework which 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 V6 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.

11.1.1 Buses

A *service integration bus*, or *bus*, within WebSphere Application Server V6 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 runtime, 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 within WebSphere Application Server V6. It is anticipated that, in a standard configuration, no more than one service integration bus will be required within a WebSphere Application Server V6 cell. However, a WebSphere 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 WebSphere cell has no runtime impact on any of the components running within a cell. It is not until members are added to a bus that any of the runtime components within an application server are affected.

Figure 11-1 on page 595 shows a service integration bus defined within a WebSphere Application Server V6 cell.

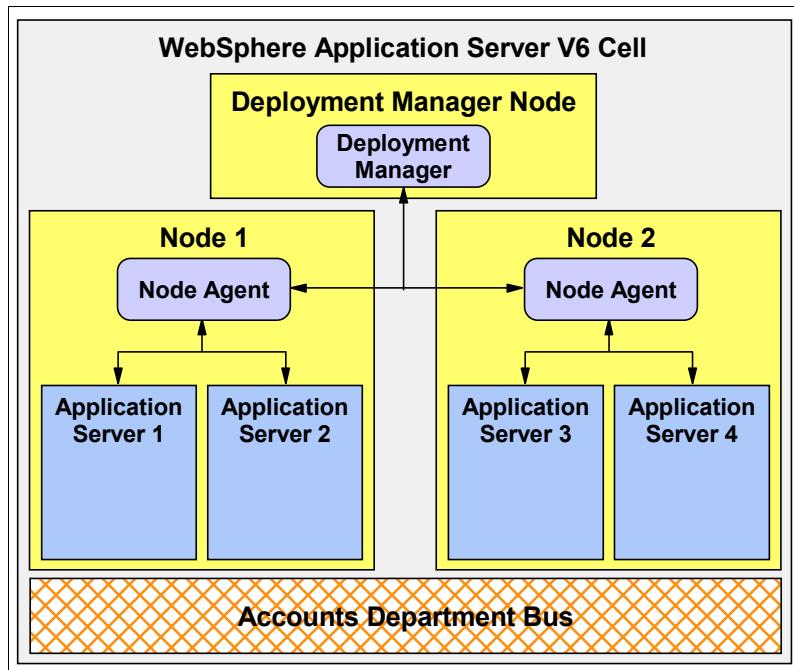


Figure 11-1 Service integration buses within a WebSphere Application Server V6 cell

11.1.2 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 service integration bus, the most important of the resources that are automatically defined is a messaging engine.

11.1.3 Messaging engines

A *messaging engine* is the component within an application server that provides the core messaging functionality of a service integration bus. At runtime, 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 11-2 on page 596 shows a WebSphere Application Server V6 cell that contains two buses, each of which has two application servers defined as bus members.

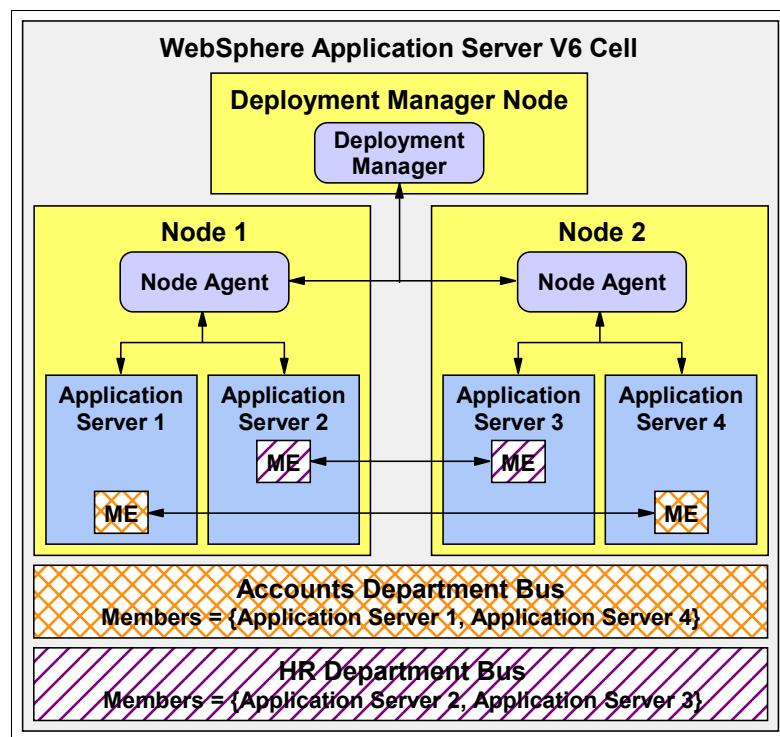


Figure 11-2 *Messaging engines within service integration bus members*

A messaging engine is a relatively lightweight runtime 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 11-3 on page 597.

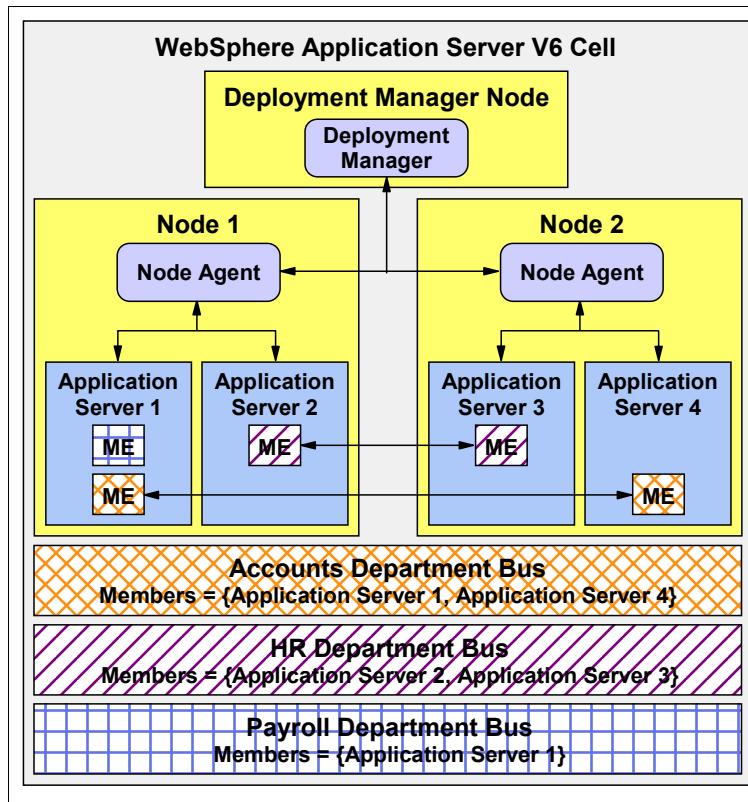


Figure 11-3 Multiple messaging engines within a single application server

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 runtime, 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 11-4.

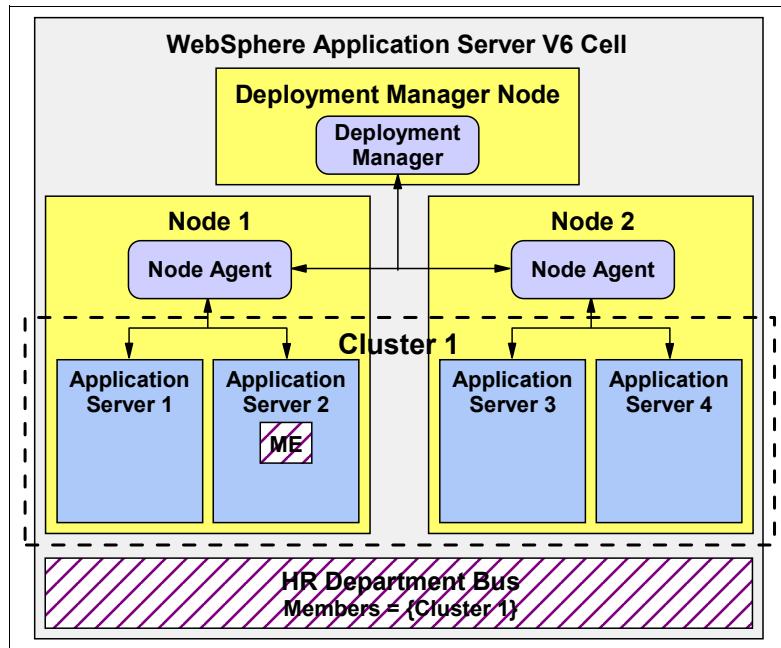


Figure 11-4 An application server cluster as a bus member

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 11-5 on page 599.

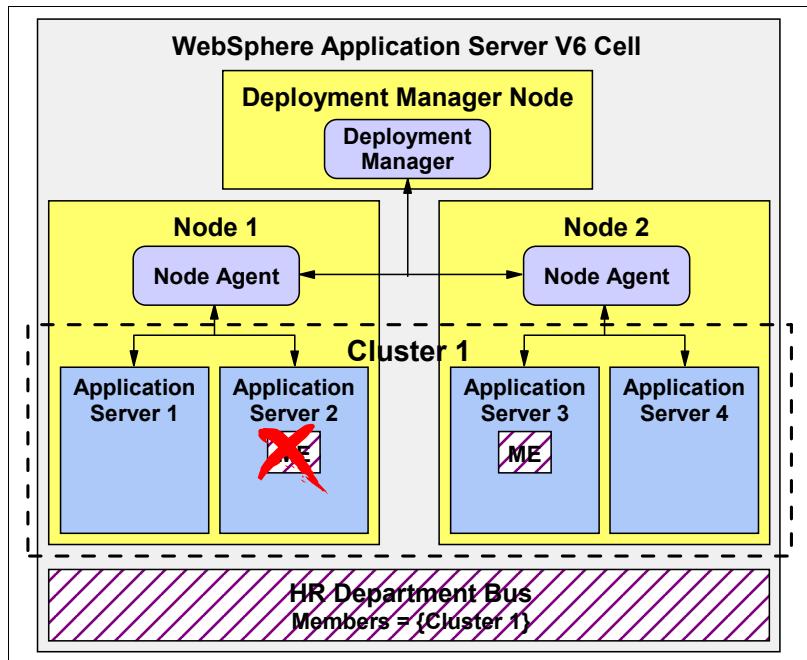


Figure 11-5 Messaging engine fail-over within an application server cluster

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 11-6 on page 600.

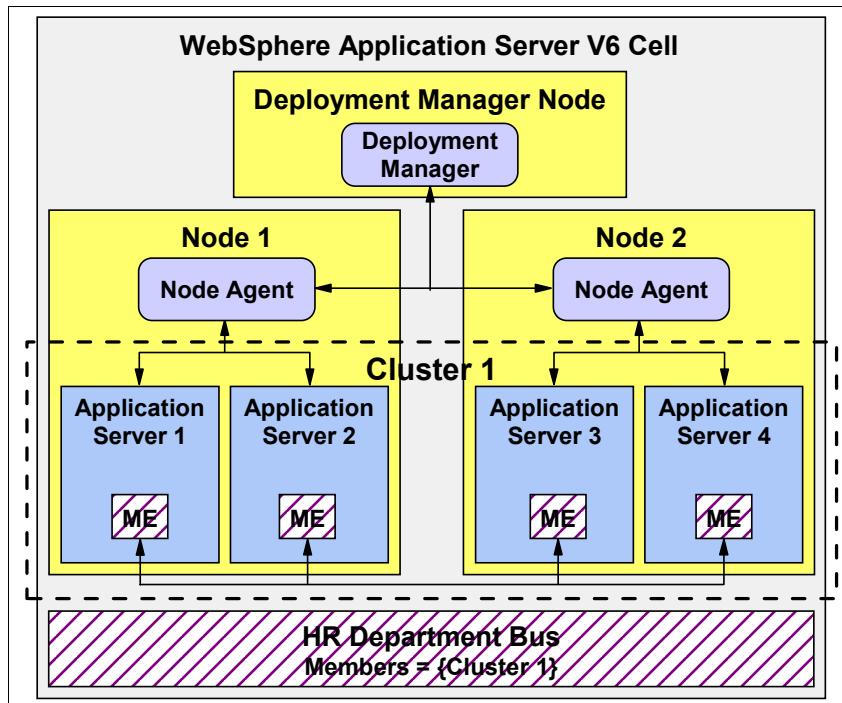


Figure 11-6 Messaging engine scalability within an application server cluster

For more information about fail-over and scalability within the service integration bus, refer to 11.3, “High availability and workload management” on page 638.

Messaging engine naming

As discussed previously, when a member is added to a service integration 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 Name>.<Server Name>-<Bus Name>

The elements are defined as:

- <Node Name> is the name of the node on which the new bus member is defined.
- <Server Name> is the name of the new application server bus member.

- <Bus Name> is the name of the service integration 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 Name>.XXX-<Bus Name>

The elements of this format are:

- <Cluster Name> is the name of the new application server cluster bus member.
- XXX is a number that is used to uniquely identify the messaging engine within the cluster. This value is incremented each time a new messaging engine is added to the cluster.
- <Bus Name> is the name of the service integration bus to which the new bus member has been added.

We can use it in an example, such as:

ITSOCluster.000-ITSOBus

11.1.4 Data stores

Every messaging engine defined within a bus has a *data store* associated with it. A messaging engine uses this data store to persist durable data, such as persistent messages and transaction states. Durable data written to the data store survives the orderly shutdown, or failure, of a messaging engine, regardless of the reason for the failure.

It can also use the data store to reduce runtime resource consumption. For example, the messaging engine can write non-persistent messages to the data store in order to reduce the size of the Java heap when handling high message volumes. This is known as *spilling*.

Figure 11-7 on page 602, shows messaging engines associated with data stores. Two of the messaging engines shown in Figure 11-7 are associated with data stores that exist within the same database. There are certain considerations you must take into account when deciding the data store topology. These considerations are discussed in more detail in 11.2.3, “Data stores” on page 620, as part of the description of the runtime components of the service integration bus.

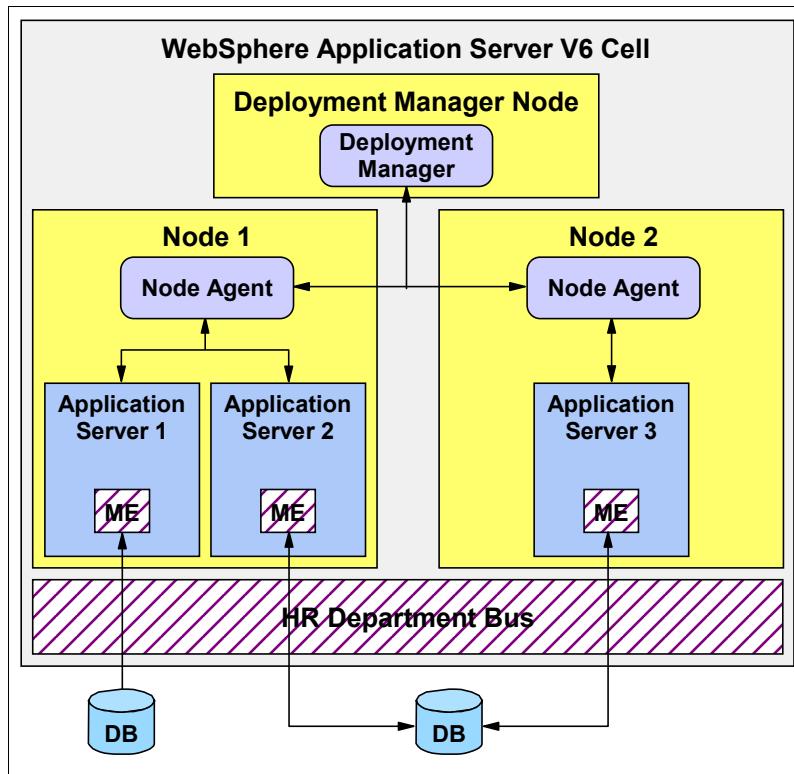


Figure 11-7 Messaging engine data stores

11.1.5 Destinations

A *destination* within a service integration 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 section 11.1.7, “Foreign buses” on page 606.

- ▶ Foreign destinations

Foreign destinations are not destinations within a service integration 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 section 11.1.7, “Foreign buses” on page 606.

Message points

Recall that a service integration bus is simply an architectural concept within a WebSphere Application Server V6 cell. Similarly, when a destination is configured on a service integration 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 data 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 which 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 service integration bus, please refer to 11.3, “High availability and workload management” on page 638.

- ▶ 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 11-8 on page 604 shows a queue destination and a topic space destination and their associated queue and publication points.

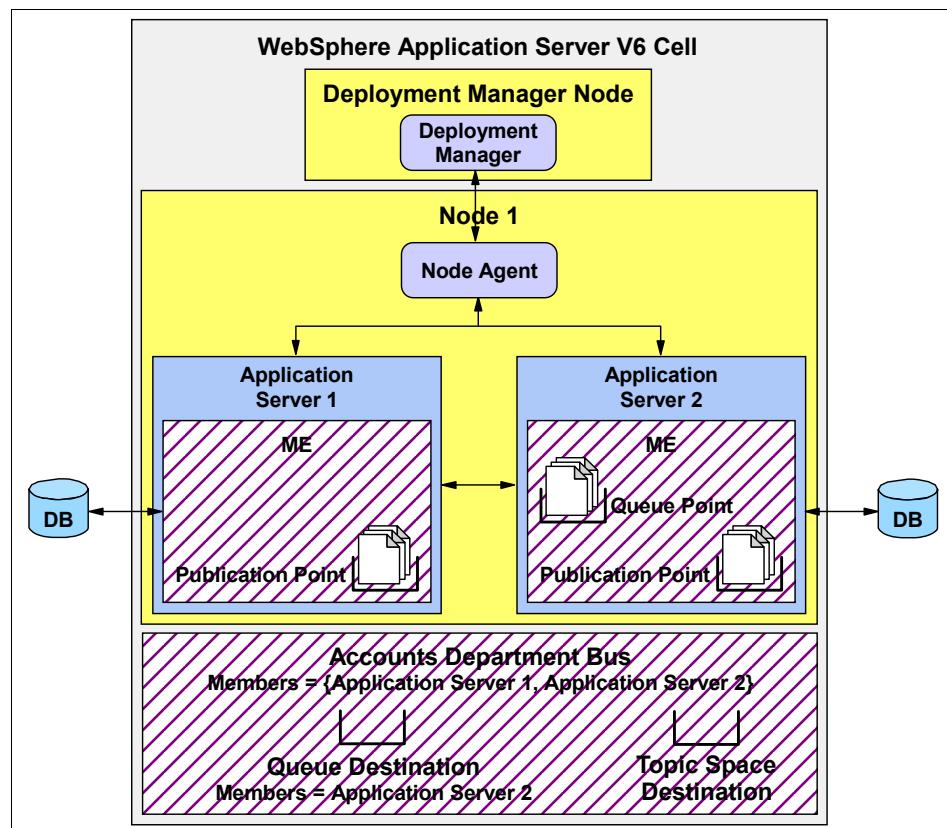


Figure 11-8 Queue and publication points in the service integration bus

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 service integration bus destination.

There are five levels of reliability that can be specified for these properties. These are described in Table 11-1.

Table 11-1 Service integration bus destination reliabilities

Reliability	Description
Least reliable	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.
Most reliable	Assured persistent Messages that are sent to this destination are never discarded.

Note: Reliability 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 service integration bus reliabilities. This is discussed in more detail in “Quality of service properties” on page 530.

Note: The reliability specified by a message producer can never exceed the maximum reliability specified on a service integration 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.

11.1.6 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 service integration 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 service integration 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*.

11.1.7 Foreign buses

A service integration 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 service integration bus in the same WebSphere Application Server V6 cell as the local bus
- ▶ A service integration bus in a different WebSphere Application Server V6 cell from the local bus
- ▶ A WebSphere MQ network

The ability of a service integration 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 which 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 service integration 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 runtime 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 service integration bus. This is shown in Figure 11-9, where the Accounts Department Bus is linked to the HR Department Bus within its own WebSphere Application Server V6 cell and the Payroll Department Bus within another WebSphere Application Server V6 cell.

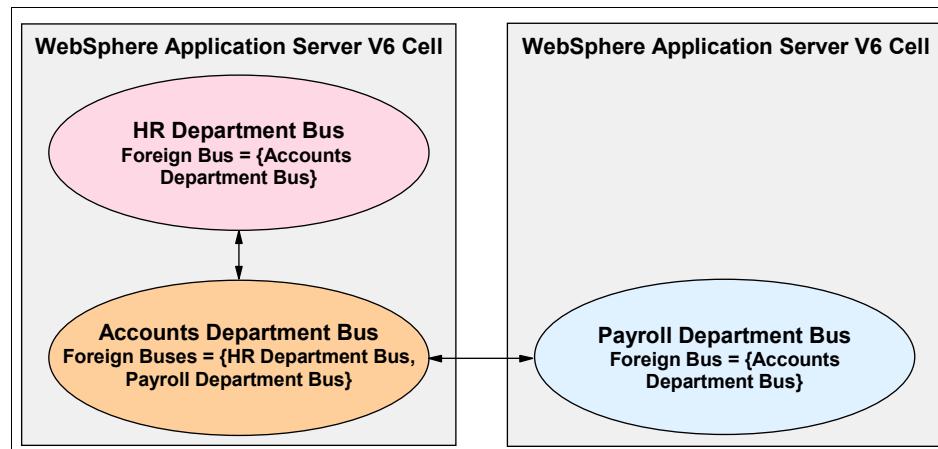


Figure 11-9 Direct, service integration bus links

- 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 11-10.

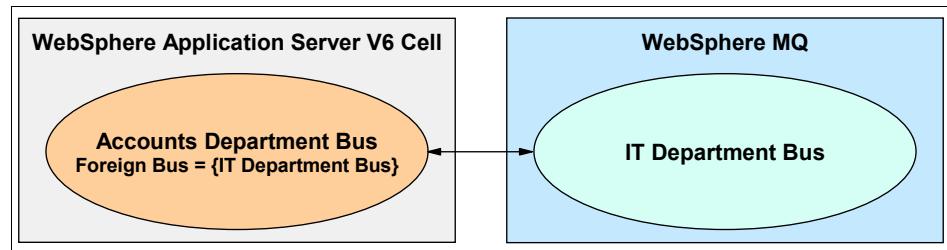


Figure 11-10 Direct, WebSphere MQ link

- 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 service integration 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 11-11, where the Accounts Department Bus is linked indirectly to the Payroll Department Bus via the HR Department Bus.

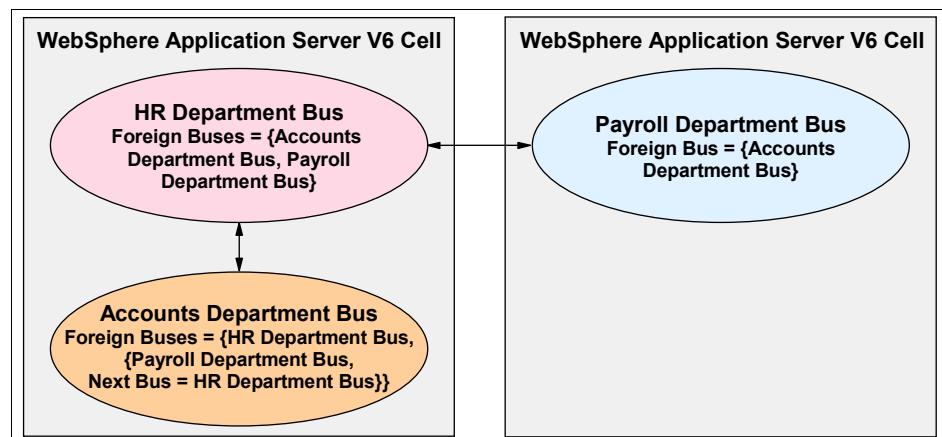


Figure 11-11 Indirect foreign bus link

Foreign bus links

Recall that a service integration bus is simply an architectural concept within a WebSphere Application Server V6 cell. Similarly, when a foreign bus is configured on a service integration 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 runtime, 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 runtime, messages that are routed to a foreign bus will flow across the corresponding link. This is shown in Figure 11-12.

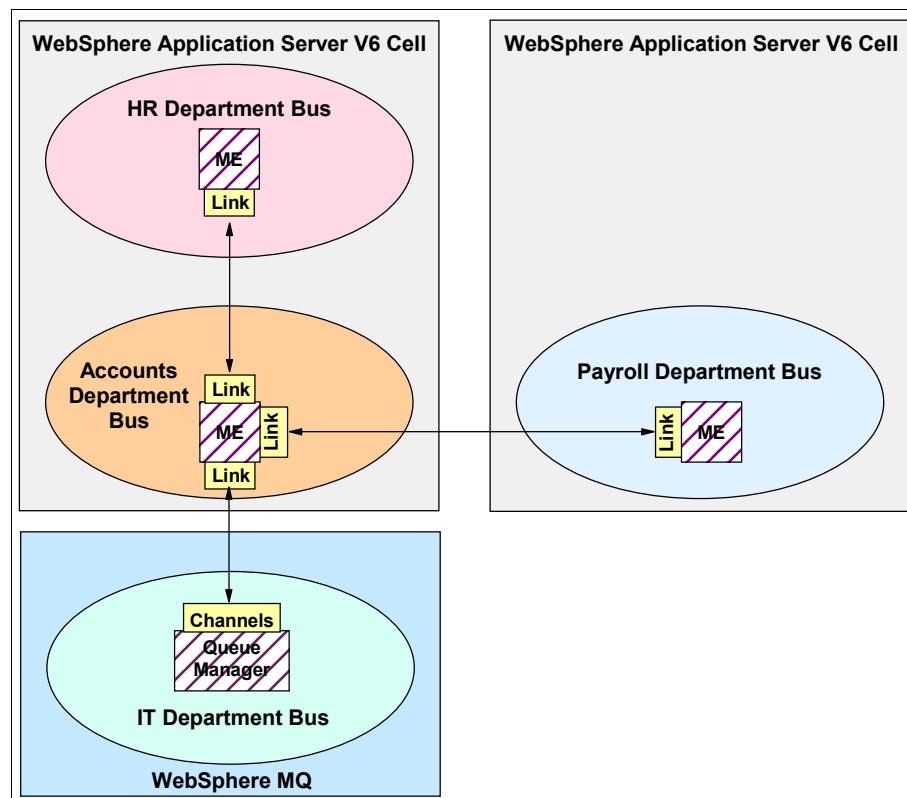


Figure 11-12 Runtime view of foreign buses

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 11-13.

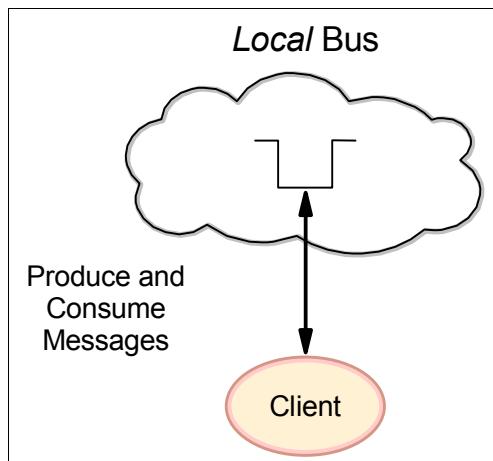


Figure 11-13 Point-to-point messaging on the local bus

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 11-14.

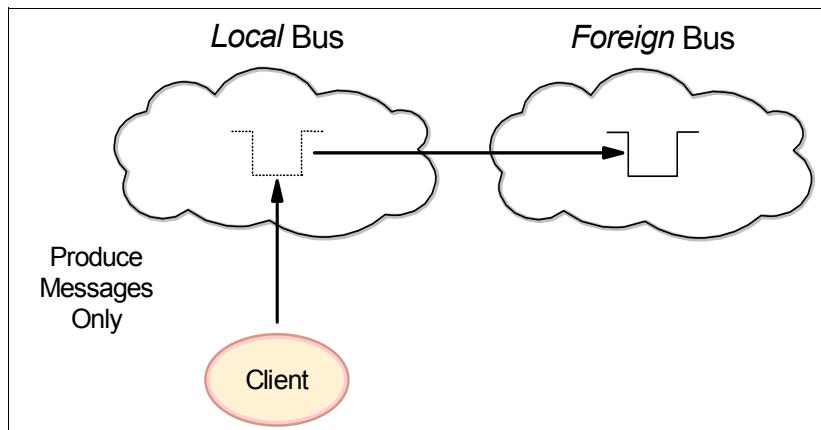


Figure 11-14 Point-to-point message producer for a foreign bus

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 11-15 on page 611.

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.

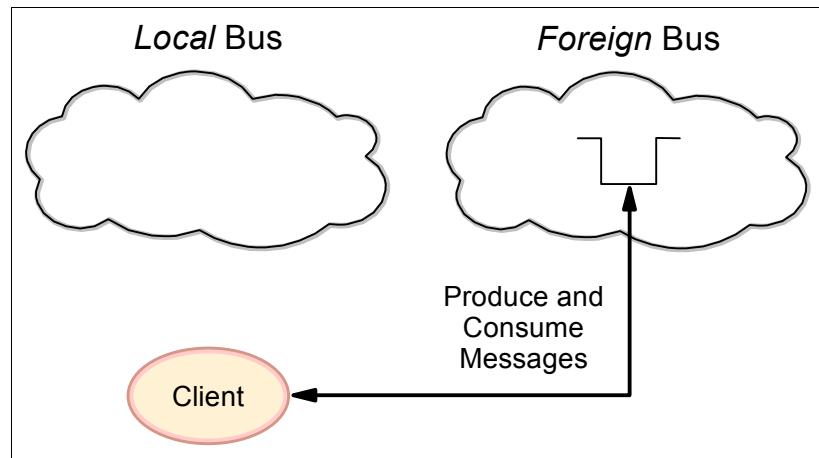


Figure 11-15 Point-to-point messaging on a foreign bus

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 11-16.

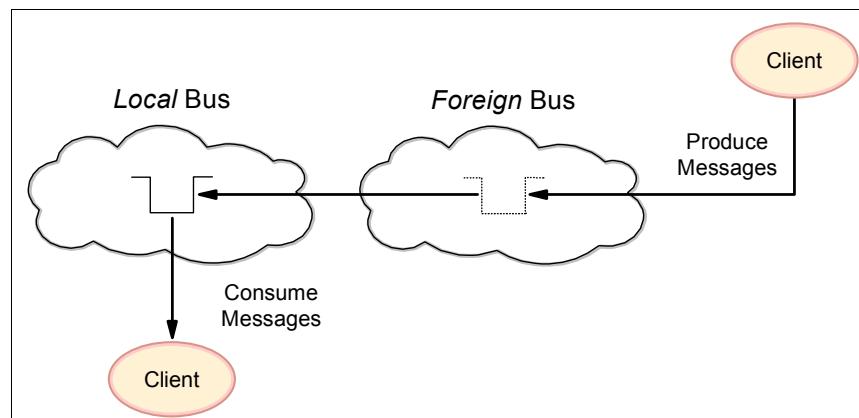


Figure 11-16 Forwarding messages for consumption from the local bus

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.

11.2 Runtime components

At runtime, a service integration bus is comprised of a collection of cooperating messaging resources. The sections that follow describe the runtime aspects of these messaging resources in more detail.

11.2.1 SIB service

The *SIB service* is a WebSphere Application Server V6 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 service integration 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 11-17 shows a SIB service within an application server environment.

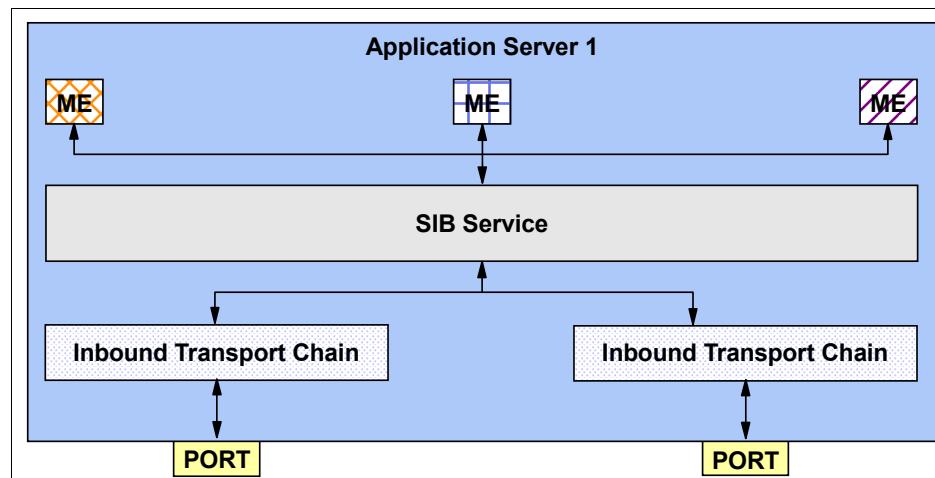


Figure 11-17 SIB service

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 service integration 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 service integration 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 V6 environment, or for messaging engines that are running in a different WebSphere Application Server V6 cell. Refer to section 10.7, “Connecting to a service integration bus” on page 576 for more information regarding bootstrap servers.

Configuration reload

The SIB service also allows certain configuration changes to be applied to a service integration 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 service integration 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 service integration bus
- ▶ Creation of a new messaging engine
- ▶ Creation of a service integration bus link
- ▶ Creation of a WebSphere MQ link

Note: Each service integration bus that requires this functionality must also be configured to support configuration reload. By default, each service integration bus has configuration reload support enabled. See Section 11.8.1, “SIB service configuration” on page 655 for more information.

11.2.2 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 service integration 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 11-2 describes these transport chains.

Table 11-2 Messaging engine inbound transport chains

Transport chain & 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
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.

Transport chain & associated port	Default port	Client Types	Description
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.

Note: Table 11-2 describes the default port numbers that are associated with the ports:

- ▶ SIB_ENDPOINT_ADDRESS
- ▶ SIB_ENDPOINT_SECURE_ADDRESS
- ▶ SIB_MQ_ENDPOINT_ADDRESS
- ▶ SIB_MQ_ENDPOINT_SECURE_ADDRESS

However, on nodes that are hosting multiple application servers, these ports might have been automatically configured to use different port numbers to avoid conflicts with existing application servers.

As discussed in section 11.2.1, “SIB service” on page 612, 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 11-3 describes the circumstances under which the inbound transport chains are started by the SIB service.

Table 11-3 Default transport chain initialization during application server startup

Application server configuration	Transport chains	
	InboundBasicMessaging InboundSecureMessaging	InboundBasicMQLink InboundSecureMQLink
SIB service disabled	Not started	Not started

Application server configuration	Transport chains	
	InboundBasicMessaging InboundSecureMessaging	InboundBasicMQLink InboundSecureMQLink
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 11-18 shows the InboundBasicMessaging and InboundSecureMessaging transport chains, and the corresponding ports that they are bound to, within an application server.

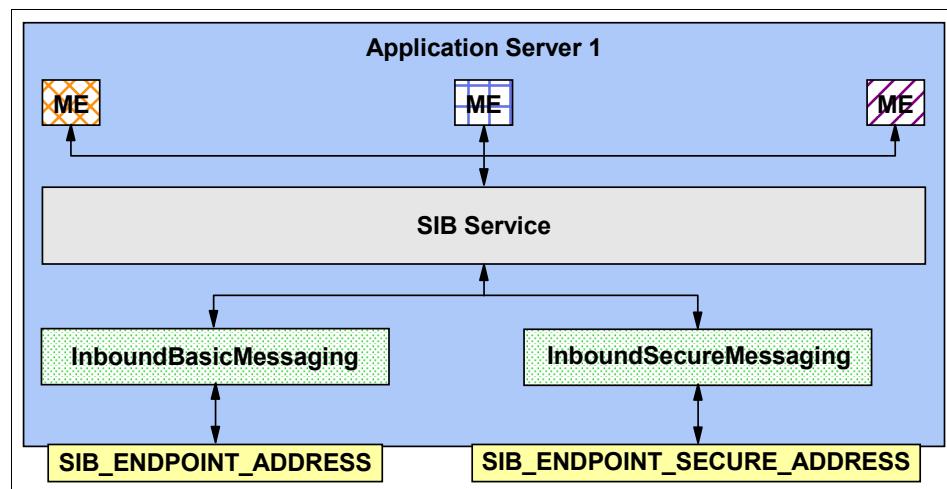


Figure 11-18 Messaging engine inbound transport chains

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 11-4 on page 617 describes these transport chains.

Table 11-4 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.

Transport chain	Description
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.
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 manager 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 instance, 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 section 10.7, “Connecting to a service integration bus” on page 576 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 WebSphere Application Server V6 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 Application Server V6 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.

11.2.3 Data stores

A messaging engine uses a JDBC data source to access the data store with which it is associated. The process of adding an application server as a member of a service integration bus automatically creates a messaging engine on that application server. By default, a data store will also be created for this messaging engine, hosted within an embedded Cloudscape database. A JDBC data source for 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 instance, 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 service integration bus components. For instance, 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 11-19, 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.

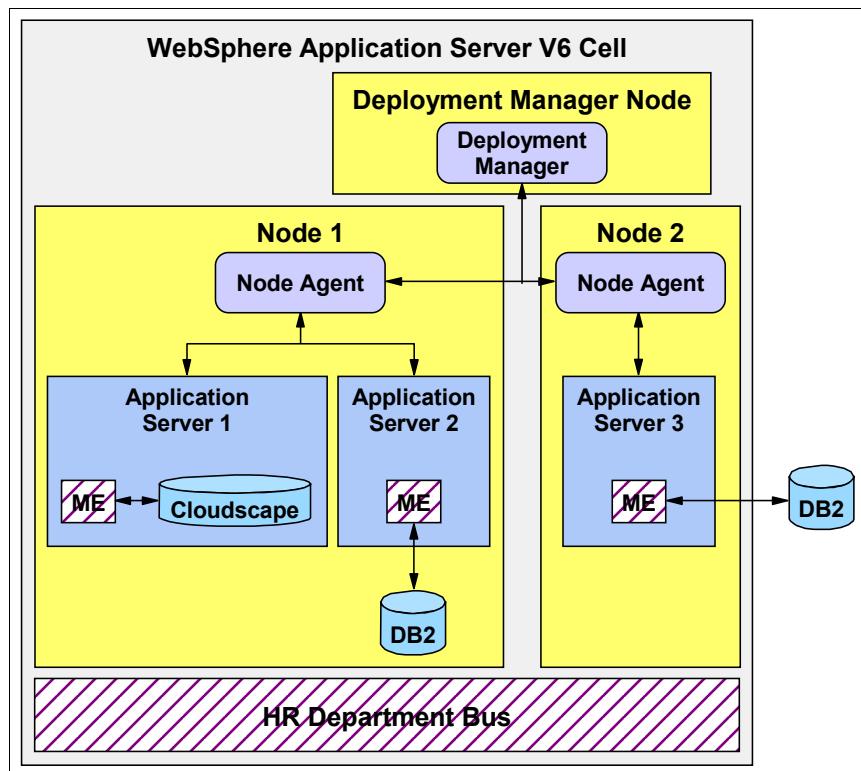


Figure 11-19 Data store locations relative to the associated messaging engine

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 instance, 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.

Figure 11-20 on page 623 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.

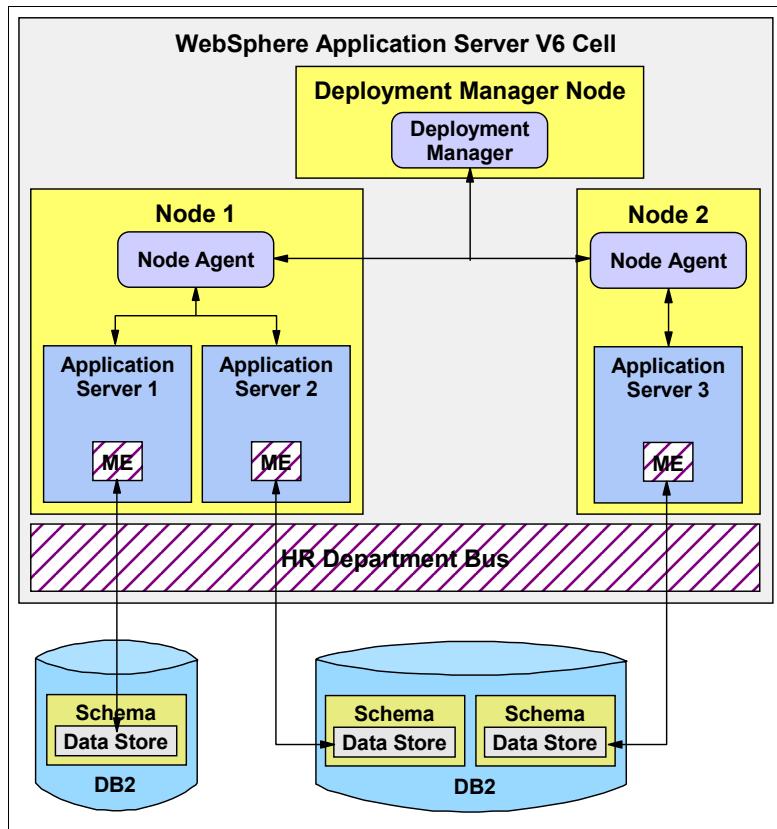


Figure 11-20 Exclusive and shared access to data stores

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 11-5 describes the tables defined within the data store for a messaging engine.

Table 11-5 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
SIBnnn, where nnn is a number	<p>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:</p> <ul style="list-style-type: none"> ▶ SIB000 Use this name for the table which contains information about the structure of the data in the other two tables ▶ SIB001 Use this name for the table which contains persistent objects ▶ SIB002 Use this name for the table which 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 V6 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.

Embedded Cloudscape considerations

As we discussed previously, each messaging engine defaults to using an embedded Cloudscape database to host its data store. However, when adding an application server cluster as a bus member, you do not have this default configuration. This is because the embedded Cloudscape database is not supported for cluster bus members. In certain failover scenarios, multiple processes might attempt to access the embedded Cloudscape database, and this is not supported.

11.2.4 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 maximum failed deliveries 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.Destinaton.<ME_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. It is also recommended 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 service integration 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.

11.2.5 Service integration bus links

As discussed in section 11.1.7, “Foreign buses” on page 606, defining a foreign bus on a service integration bus simply defines a link between the two buses at an architectural level. When the foreign bus in question represents another service integration bus, the link is implemented at runtime 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 service integration bus. This is shown in Figure 11-21.

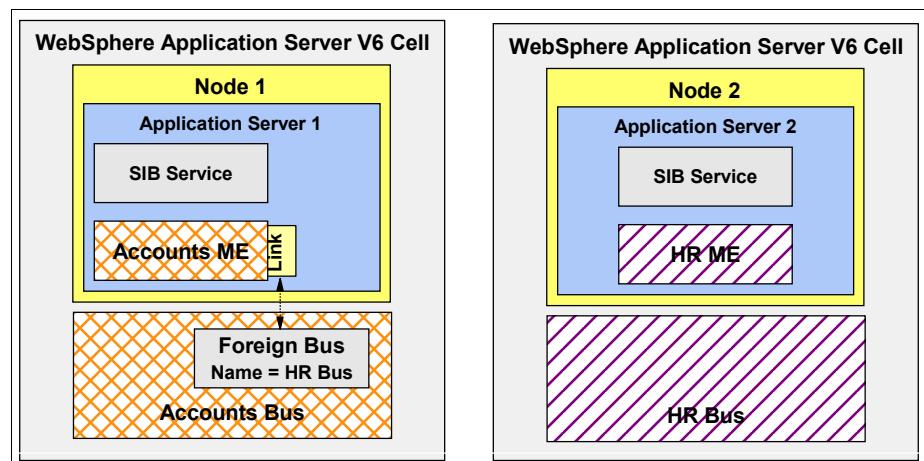


Figure 11-21 Association between a service integration bus link and a foreign bus

This requirement also determines the order in which these objects must be defined. The foreign bus must be defined within a service integration 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 service integration bus.

The names of each of the buses involved in the link must also be unique. For this reason, if two service integration buses within separate WebSphere Application Server V6 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 attempt to connect to the foreign bus as though it were a remote client, even if the foreign bus is defined within the same WebSphere Application Server V6 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 section 10.7, “Connecting to a service integration bus” on page 576.

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 11-22 shows this process.

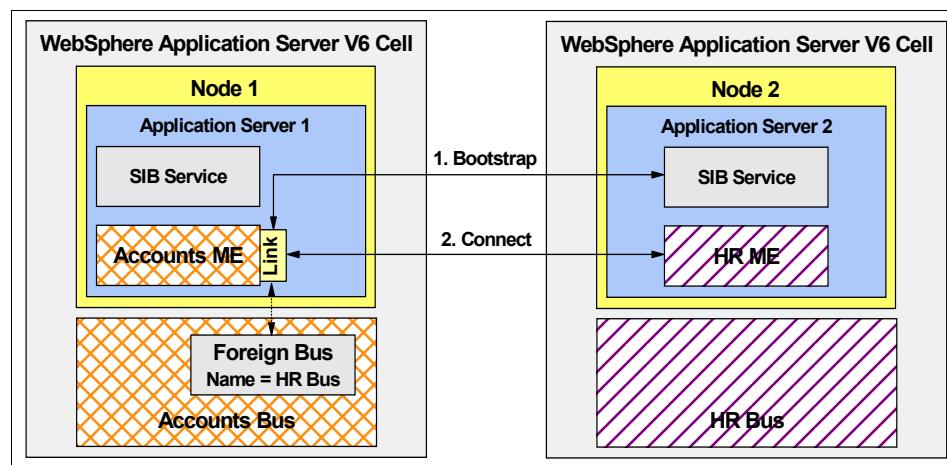


Figure 11-22 Bootstrapping during service integration bus link initialization

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 runtime. This is shown in Figure 11-23 on page 629.

Note: The name specified for the service integration bus link within both buses must be the same.

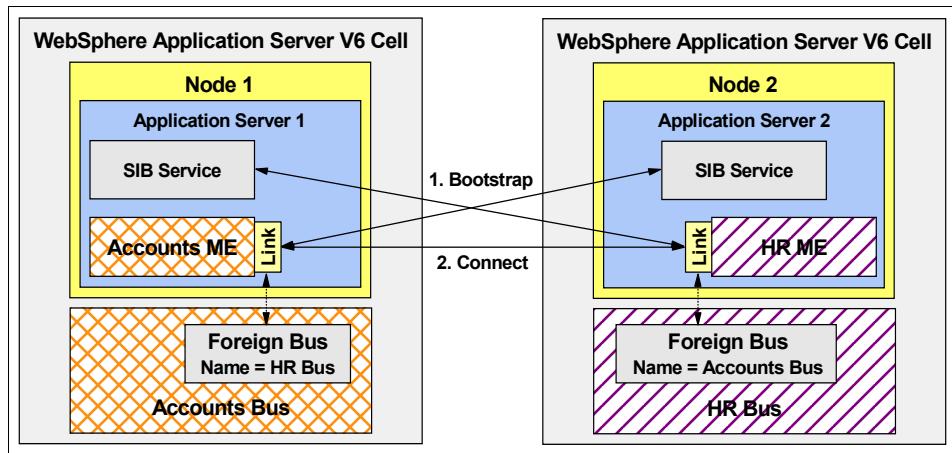


Figure 11-23 Defining a service integration bus link in both directions

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.

11.2.6 WebSphere MQ links

As discussed in section 11.1.7, “Foreign buses” on page 606, defining a foreign bus on a service integration 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 runtime 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 11-24 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.

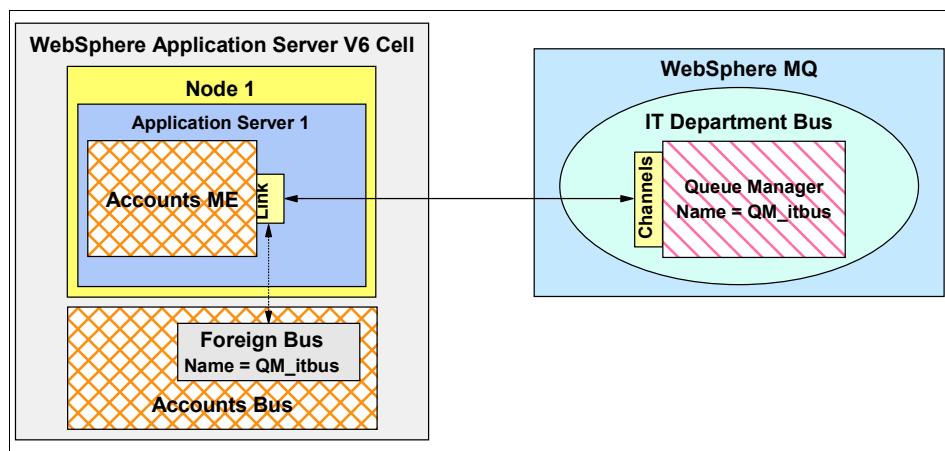


Figure 11-24 Overview of a WebSphere MQ link

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 service integration 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 11-25.

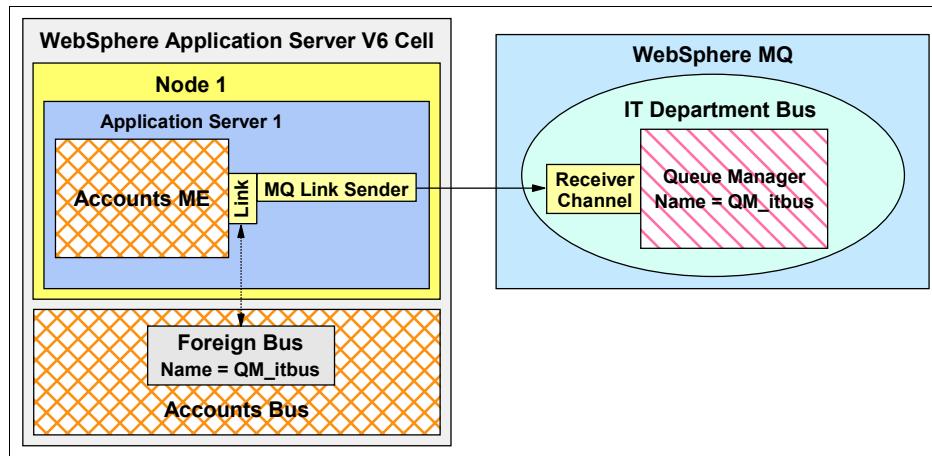


Figure 11-25 WebSphere MQ link sender channel

Note: It is only necessary to define a WebSphere MQ link sender channel if messages are required to be sent from the service integration 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 service integration bus. It converts messages from the format used within WebSphere MQ, to the format used by the service integration 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 11-26.

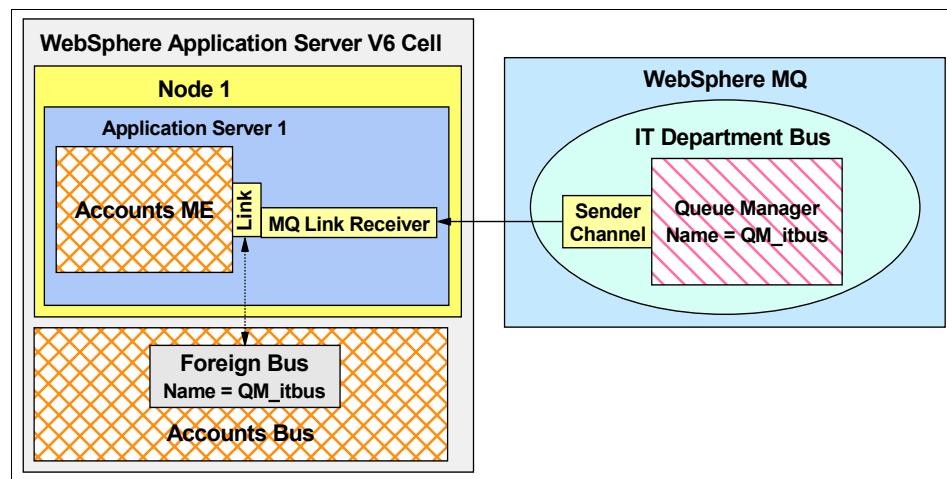


Figure 11-26 WebSphere MQ link receiver channel

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 service integration 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 614, 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 service integration 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 service integration bus, a destination can be uniquely identified using the following elements:

- ▶ Service integration bus name
- ▶ Destination name

The service integration 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 service integration bus.

The service integration bus uses an alias destination to map between the shorter name used by WebSphere MQ, and the longer name used by the service integration bus. A WebSphere MQ client application can address a message to an alias destination within a service integration 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 11-27 on page 635.

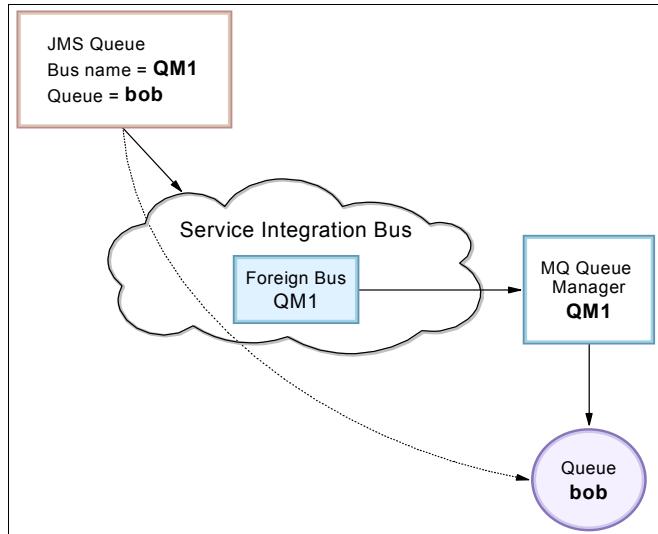


Figure 11-27 Simple WebSphere MQ addressing

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 11-28.

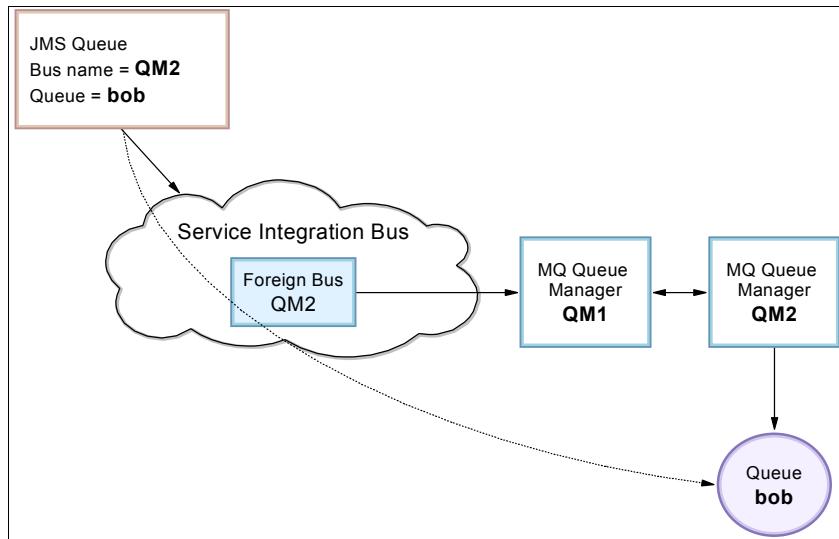


Figure 11-28 Simple WebSphere MQ addressing

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 service integration bus. This is shown in Figure 11-29.

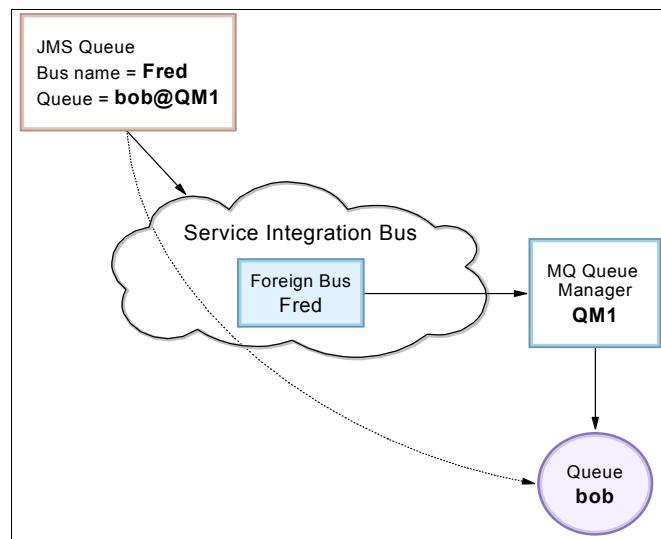


Figure 11-29 Advanced WebSphere MQ addressing

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 11-30 on page 637.

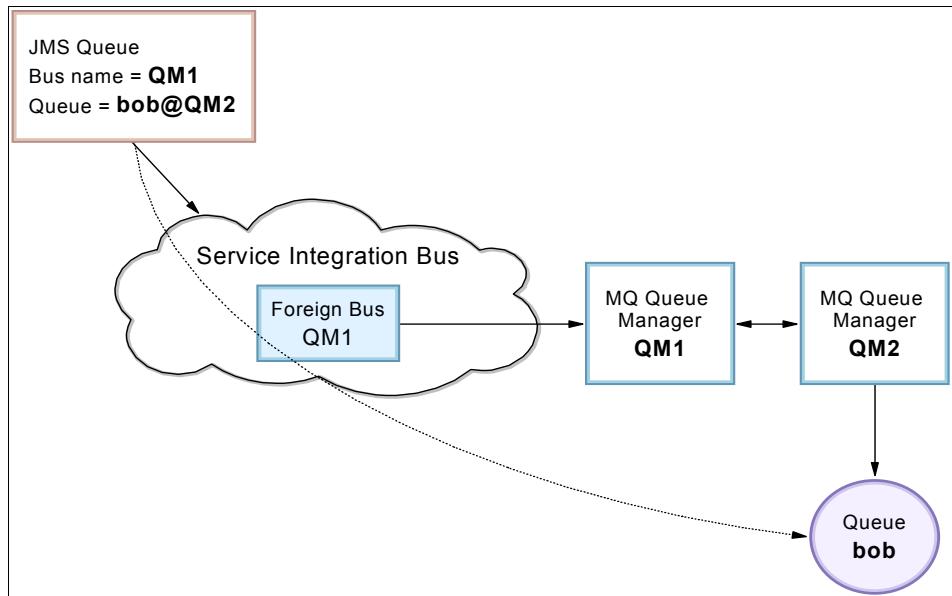


Figure 11-30 Advanced WebSphere MQ addressing

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 service integration 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 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 service integration bus with a WebSphere MQ client link. Queues previously defined on the V5.x embedded JMS server will be created automatically on the service integration 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: It is recommended 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.

11.3 High availability and workload management

Note: This section introduces you to the high availability and workload management capabilities when using the service integration 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 service integration bus members. It is worth noting however, that service integration bus messaging engines do not follow the same clustering model that J2EE applications do in clusters.

11.3.1 Cluster bus members for high availability

When You add a cluster to a service integration 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 11.8.9, “Setting up preferred servers” on page 672 for details on configuring preferred servers.

11.3.2 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.

11.3.3 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 service integration bus.

- ▶ Message producers must be configured to ensure that their messages will be workload-managed onto the different partitions of a partitioned queue. Following are examples of workload management:
 - Message producers, JMS clients, connect directly to the cluster. This has some restrictions in version 6.0. See 11.3.4, “JMS clients connecting into a cluster of messaging engines” on page 640. 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.

11.3.4 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 10.7, “Connecting to a service integration bus” on page 576.

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 585.

- ▶ Provide different clients with differently configured connection factories, each of which has a different provider endpoint in the first position in the list.

11.3.5 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 service integration bus 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 no

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 11-31 on page 644.

In Figure 11-31 on page 644, the shading indicates the preference order of the servers.

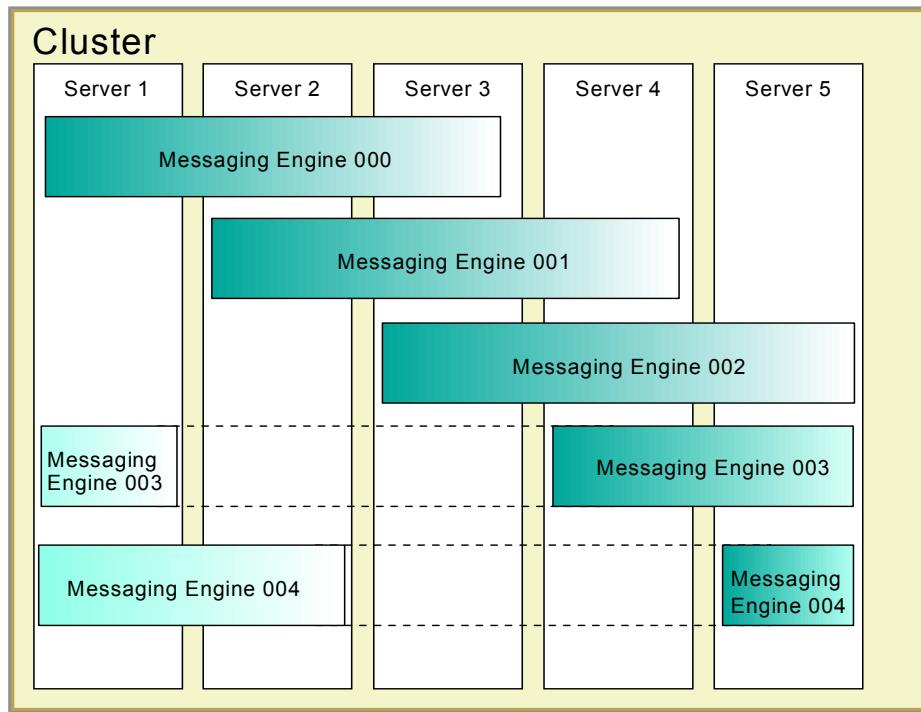


Figure 11-31 Configuring large clusters of messaging engines

11.3.6 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 service integration bus can then workload manage the messages across the cluster.
4. Install an MDB application on the cluster to consume the queue messages.

11.4 Service integration bus topologies

This section discusses briefly some messaging topologies, working up from the simplest to more complex configurations.

11.4.1 One server in the cell is a member of one bus

In this topology, there is only one service integration 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 10.7, “Connecting to a service integration bus” on page 576 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.

11.4.2 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 service integration 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 upgradable to support high availability or workload management. See the list of cons in 11.4.1, “One server in the cell is a member of one bus” on page 645.

11.4.3 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 service integration 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 638 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 672.

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 is available, that the messaging engine will not have another server on which to start even if other servers are available in the cluster. See “High availability and workload management” on page 638.

11.4.4 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. See “High availability and workload management” on page 638 for more details.

The drawback is:

- ▶ There are some restrictions on the workload management of client connections directly into a cluster. See 11.3.4, “JMS clients connecting into a cluster of messaging engines” on page 640 for details.

11.4.5 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 11.3.4, “JMS clients connecting into a cluster of messaging engines” on page 640), 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:

- ▶ These more complex messaging topologies take a little more planning and configuration than simpler topologies.

11.4.6 Multiple buses in a cell

It is possible to have many service integration 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 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.

11.5 Service integration bus and message-driven beans

Message-driven beans (MDBs) attached to destinations in the service integration 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 service integration 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.

11.5.1 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 10.7, “Connecting to a service integration bus” on page 576.

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 which 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 following warning message will appear in SystemOut.log:

Example 11-1 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 11-2 is displayed when the MDB is connected to its destination:

Example 11-2 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 Example 11-3 is displayed:

Example 11-3 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 in Example 11-3.

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 11-4 appears on the first failed attempt to connect to a messaging engine. Subsequent failures are silent:

Example 11-4 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 11-5 is displayed with a connection is made:

Example 11-5 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.

11.5.2 MDBs and clusters

The behavior of message-driven beans installed on clusters that use the service integration bus is directly related to the service integration 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 fail or be 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 service integration 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 be 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.

11.6 Service integration bus security

When security is enabled on WebSphere Application Server V6, certain steps must be taken for JMS applications using the service integration 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, as 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 user name and password on the call to create 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.

Any user that authenticates as a valid user to WebSphere has, by default, full access to the bus and all destinations on it. It is possible to configure destination and even topic-specific authentication requirement, if you want.

Every bus has an optional inter-engine authentication alias which can be specified. If this property is left unset, then it will be ignored. However, if an alias is specified and security enabled, then the ID will be checked when each messaging engine starts communications to other messaging engines in the bus. This provides additional security to prevent hackers pretending to be another messaging engine in the bus.

Details on WebSphere security can be found in *WebSphere Application Security V6 Security Handbook*, SG24-6316

11.7 Problem determination

The following information is presented to help you become familiar with successful messaging engine startup, and some common problems.

No problems

Example 11-6 shows an example of what you can expect to see in systemOut.log on server start up for a messaging engine that starts sucessfully:

Example 11-6 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 11-7.

Example 11-7 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 11-8, 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 data store as another messaging engine.

Example 11-8 Messaging engine unique id doesn't match

```
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-ITSOBUS cannot be restarted because a serious error has been reported.
CWSID0016I: Messaging engine Node1.server1-ITSOBUS is in state Stopped.

The simplest solution is to drop the tables in the database, or delete and recreate the database 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. See , “Adding the bus member” on page 664 for more details.

CWSIT0019E: No suitable Messaging Engine...

This exception shown in Example 11-9 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.

Example 11-9 Exception on createConnection call

```
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 ITSOBus.
```

11.8 Configuration and management

This section discusses how to set up and configure a service integration bus using the administrative console.

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.

11.8.1 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 panel.

1. Select **Servers** → **Application Servers**.
2. Select any server.
3. Under **Communications**, expand the **Ports** heading. **SIB_ENDPOINT_ADDRESS** is the port used by SIB Service for that server. See Figure 11-32.

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 tunneled secure endpoints are also provided: **jfap/http/tcp** and **jfap/http/ssl/tcp**. Refer to the Information Center for more details.

Communications		
Ports		
Port Name	Port	details
BOOTSTRAP_ADDRESS	9810	
SOAP_CONNECTOR_ADDRESS	8880	
SAS_SSL_SERVERAUTH_LISTENER_ADDRESS	9404	
CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS	9405	
CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS	9406	
WC_adminhost	9061	
WC_defaulthost	9080	
DCS_UNICAST_ADDRESS	9353	
WC_adminhost_secure	9044	
WC_defaulthost_secure	9443	
SIB_ENDPOINT_ADDRESS	7276	
SIB_ENDPOINT_SECURE_ADDRESS	7286	
SIB_MQ_ENDPOINT_ADDRESS	5558	
SIB_MQ_ENDPOINT_SECURE_ADDRESS	5578	
ORB_LISTENER_ADDRESS	9101	

Figure 11-32 Port numbers used by a server

The settings for the SIB service of an application server can be found on the administrative console:

1. Select **Servers** → **Application Servers**.
2. Select any server.
3. Under **Server messaging** select **SIB service**. See Figure 11-33 on page 657.

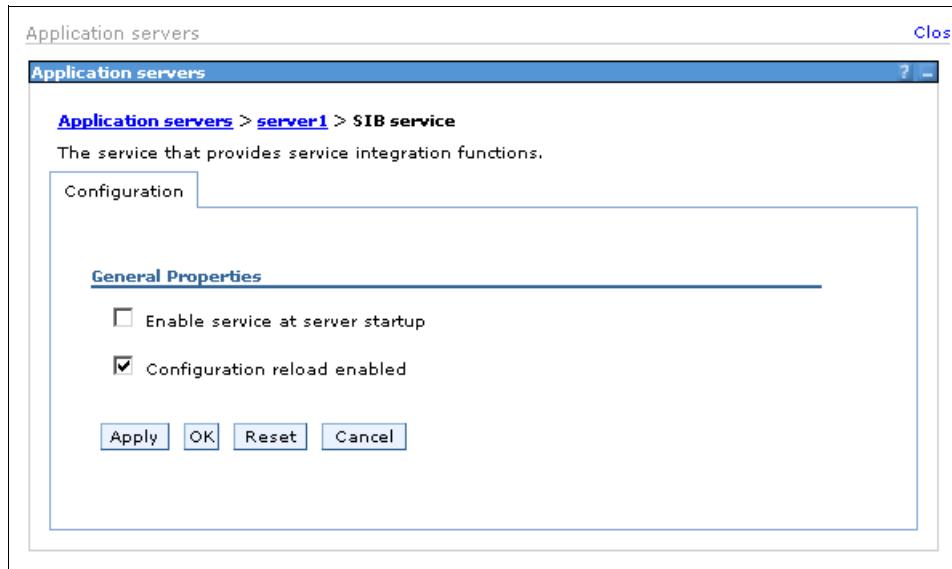


Figure 11-33 SIB Service panel

The panel 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 select changes to a bus configuration during runtime. 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.

11.8.2 Creating a bus

No service integration buses are defined by default. To create a bus, do the following:

1. Select **Service integration** → **Buses**.
2. Click **New**. See Figure 11-34.

General Properties

Name	ITSONodeSamplesBus
UUID	A9A7473F53245F9F
Description	(empty)

Additional Properties

- [Bus members](#)
- [Messaging engines](#)
- [Destinations](#)
- [Mediations](#)
- [Foreign buses](#)
- [Custom properties](#)
- [Inbound Services](#)
- [Outbound Services](#)

Security

<input checked="" type="checkbox"/> Secure
Inter-engine authentication alias (none)
Mediations authentication alias (none)

Related Items

- [J2EE Connector Architecture \(J2C\) authentication data entries](#)

Buttons

Inter-engine transport chain

Discard messages

Configuration reload enabled

High message threshold

Figure 11-34 Bus creation panel

The only required property is **Name**. 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.

The following properties can also be set:

– **Description**

This field is an optional description for the bus, for administrative purposes.

– **Secure**

Select this option to inherit the Global Security setting of the cell. Deselect this option if you always want to disable bus security. When the Secure property is selected and global security for the cell is also selected, access to the bus itself and to all destinations on the bus must be authorized.

– **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.

– **Mediations authentication alias**

Enter the name of the authentication alias used to authorize mediations to access the bus.

– **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 11.8.1, “SIB service configuration” on page 655.

– **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.

11.8.3 Adding a bus member using a default data store

Every messaging engine has a data 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 which automatically creates a messaging engine and uses the default Cloudscape database, do the following.

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 panel click **Add**. See Figure 11-35 on page 661.

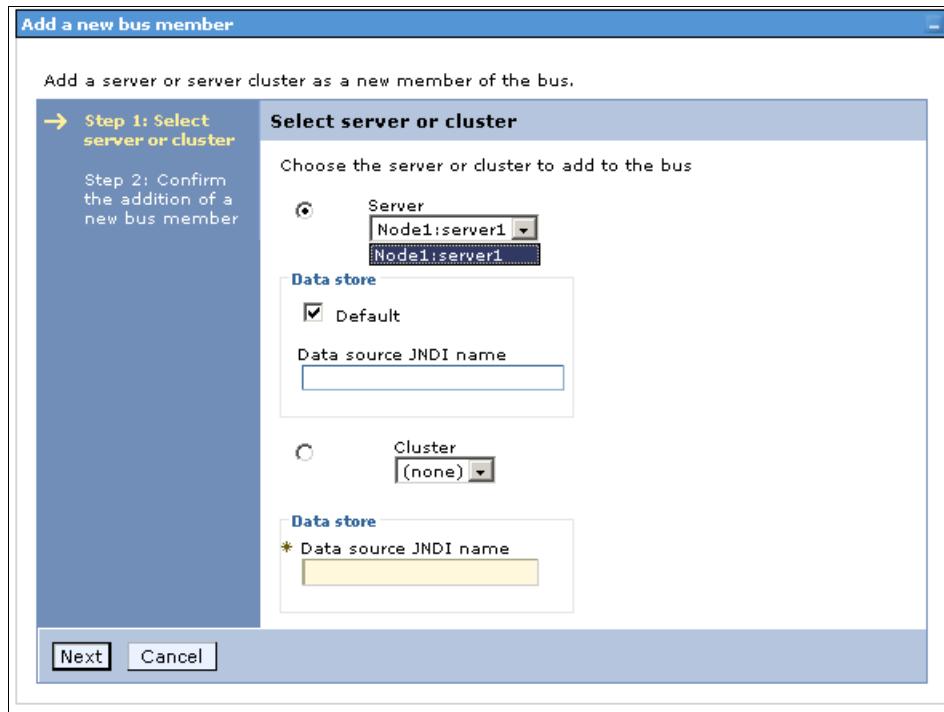


Figure 11-35 Adding a bus member

- Click the appropriate radio button to add a server or cluster, then select the server or cluster from the list.
 - Check the **Default** box to use the default Cloudscape data store. You do not need to enter the data source JNDI name. It will default to the proper value when the server or cluster is added to the bus.
5. Click **Next** and then **Finish** and save your changes.

11.8.4 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 661.

1. Select **Security** → **Global security**.
2. Under **Authentication** expand the **JAAS Configuration** 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. Information about defining JDBC providers and data sources can be found in 7.2, “JDBC resources” on page 321.

1. Select **Resources** → **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** and then click **Apply**.
5. 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:
 - 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.

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.

6. Create a data source for the bus member. Select **Resources → JDBC Providers** and then select your DB2 JDBC Provider.
7. Select **Data sources**.
8. Click **New** to create a new data source.
 - a. 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.
 - b. 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.

- c. Click **Apply** or **OK** 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**. See Figure 11-35.
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.
 - c. Provide the data source JNDI name for the data store.
- 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.
 - c. Enter the data source JNDI name for the data store.

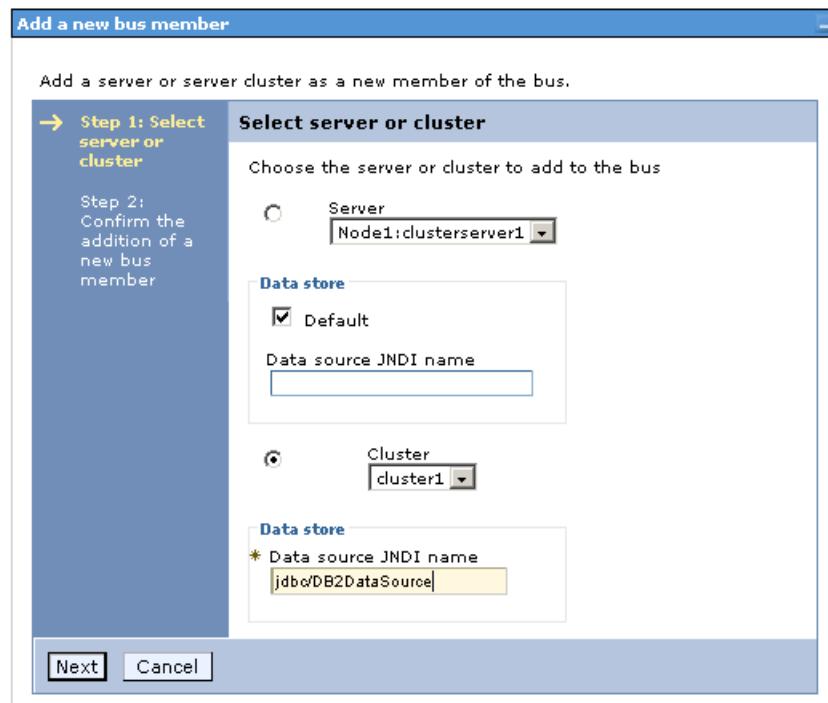


Figure 11-36 Adding a cluster to a bus.

5. Click **Next** and then click **Finish**.

6. You now have to set the J2C authentication alias on the newly created messaging engines data store. Select **Service integration** → **Buses**. Select the appropriate bus.
7. Select **Messaging engines**.
8. Select the messaging engine created when you added the bus member.
9. Select **Data store** in the Additional Properties section. See Figure 11-37.
10. From the menu, select the authentication alias, allowing the messaging engine to access the database.

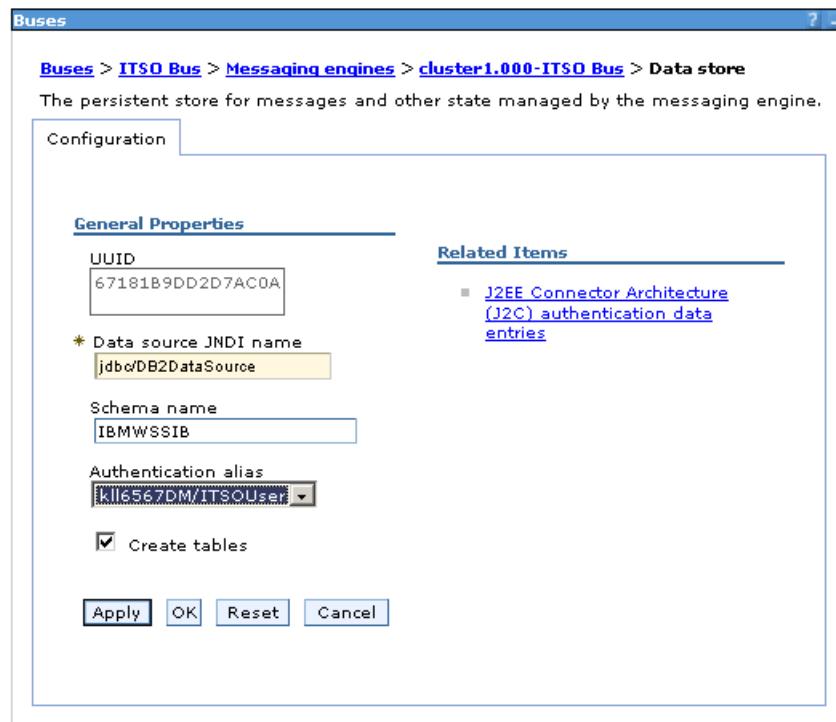


Figure 11-37 Defining the authentication alias for the data store

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.

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 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*.

11.8.5 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 Additional Properties section. See Figure 11-38.

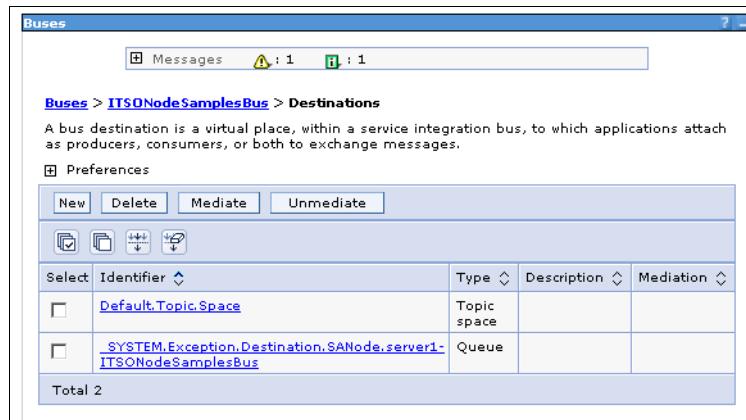


Figure 11-38 Default destinations

The Destinations panel shows two destinations that are created automatically for you. The Default.Topic.Space is a default topic space which can be used for publish/subscribe messaging. It can be deleted. The _SYSTEM.Exception.Destination is a built in queue which cannot be deleted.

4. Click **New**. See Figure 11-39 on page 667.

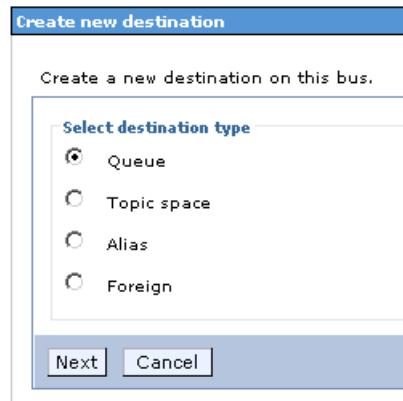


Figure 11-39 Options when creating a new destination

Select **Queue** from the radio button list and click **Next**. See Figure 11-40.

5. Provide an identifier and optional description for the queue.

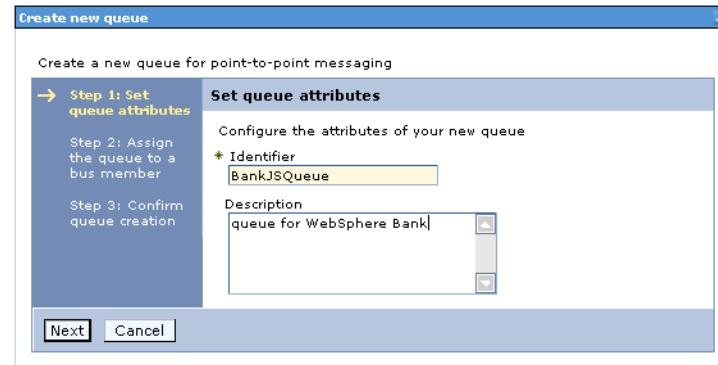


Figure 11-40 Provide an identifier for your destination

If your application uses the JMS interface, it is not sufficient to create a destination on the service integration bus. A JMS destination referencing the service integration 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 539).

Click **Next**. See Figure 11-41 on page 668.

6. Select a bus member for the queue point for this queue from the list for the queue. Click **Next**.

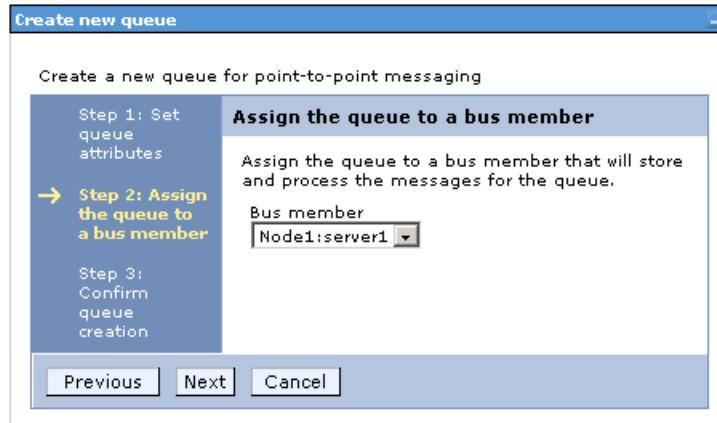


Figure 11-41 Select a bus member for the queue

7. Click **Finish** and then save your changes.

11.8.6 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 Additional Properties 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.

11.8.7 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 11.1.7, “Foreign buses” on page 606.

1. Select **Service integration** → **Buses**.
2. Select the bus on which you want to create a topic space.
3. Select **Destinations**. in the Additional Properties section.

4. Click **New**.

5. Select **Alias** from the list and click **Next**. See Figure 11-42.

The dialog box is titled "Configure the attributes of your new alias destination". It contains the following fields:

- * Identifier: Text box containing "SANodeSamplesBus".
- description: Text area.
- Bus: Drop-down menu set to "SANodeSamplesBus".
- * Target identifier: Text box.
- Target bus: Drop-down menu set to "Select...".
- Default reliability: Drop-down menu set to "Inherit".
- Maximum reliability: Drop-down menu set to "Inherit".
- Enable producers to override default reliability: Drop-down menu set to "Inherit".
- Send allowed: Drop-down menu set to "Inherit".
- Receive allowed: Drop-down menu set to "Inherit".
- Reply destination: Text box.
- Reply destination bus: Text box.
- Default forward routing path: Text area.
- Delegate authorization check to target destination

Figure 11-42 Alias destination properties

The properties to note are:

– **Identifier**

This field is the destination name as known by 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 service integration 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 panel that you want to override for the destination.

6. Click **Next**.
7. Click **Finished**.
8. Save your changes.

11.8.8 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 662.
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 11-43 on page 671.

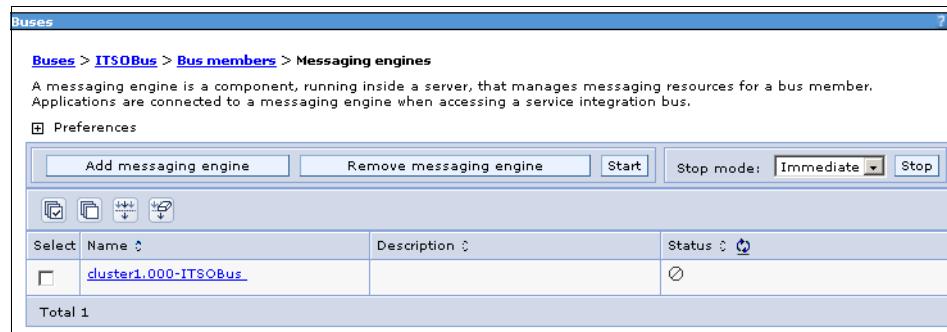


Figure 11-43 Messaging engines as part of a cluster bus member

- Click **Add messaging engine**. Enter data store information for the new messaging engine as in Figure 11-44.

Buses > ITSOBus > Bus members > Messaging engines > Data store

The persistent store for messages and other state managed by the messaging engine.

Configuration

General Properties	
UUID	D5925451AB539A32
* Data source JNDI name	jdbc/ME001
Schema name	ME001
Authentication alias	kll6567DM/DB2
<input checked="" type="checkbox"/> Create tables	
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>	
Related Items	
<ul style="list-style-type: none"> ■ J2EE Connector Architecture (J2C) ■ authentication data entries 	

Figure 11-44 Provide information for the messaging engines data store.

- Enter the required information for the database. For information about using multiple data stores, see 11.2.3, "Data stores" on page 620.
- Click **Apply** or **OK** and save your changes.

11.8.9 Setting up preferred servers

Configure a messaging engine to 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 service integration bus.

Note: Before attempting to configure a system for workload management and high availability, consult the following:

- ▶ 11.3, “High availability and workload management” on page 638
- ▶ 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 V6: High Availability Solutions*, REDP-3971

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 11-45 on page 673.

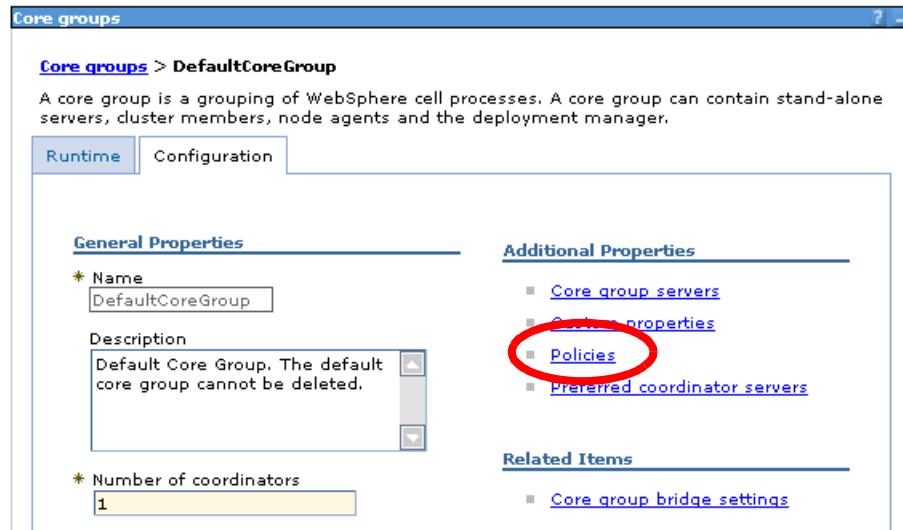


Figure 11-45 Default Core Group

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 11-46.

Core groups > DefaultCoreGroup > Policies				
This panel provides a list of the core group's policies. Policies are used by the coordinators to determine on which core group servers the group members are activated or deactivated.				
<input type="checkbox"/> Preferences <input type="button" value="New"/> <input type="button" value="Delete"/> <input type="button"/> <input type="button"/> <input type="button"/> <input type="button"/>				
Select	Name	Description	Policy type	Match criteria
<input type="checkbox"/>	Clustered TM Policy	TM One-Of-N Policy	One of N policy	type=WAS_TRANSACTIONS
<input type="checkbox"/>	Default SIBus Policy	SIBus One-Of-N Policy	One of N policy	type=WSAF_SIB
Total 2				

Figure 11-46 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 11-47 on page 674.

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**.

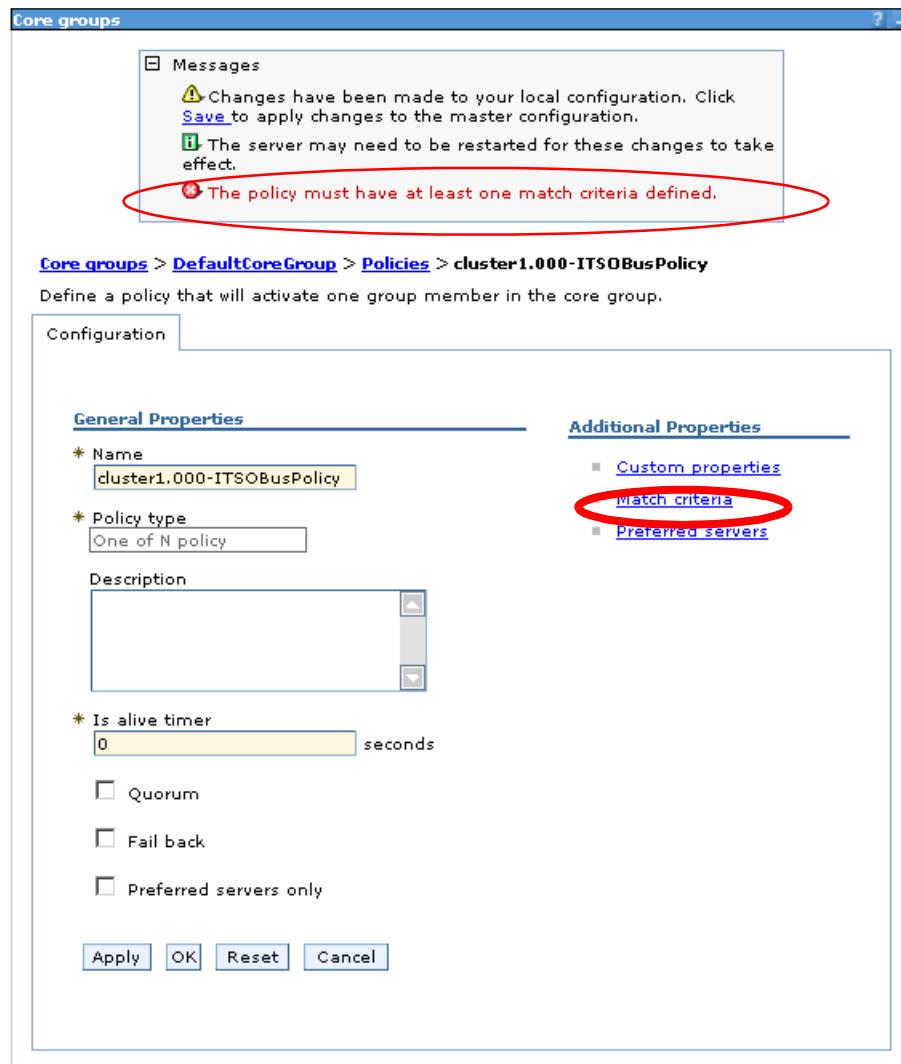


Figure 11-47 Defining a new policy

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. 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.
8. Select **Match criteria** in the Additional Properties section.
9. Click **New**. See Figure 11-48.

Enter type for the name and WSAF_SIB for the value. This match criteria will match any messaging engine.

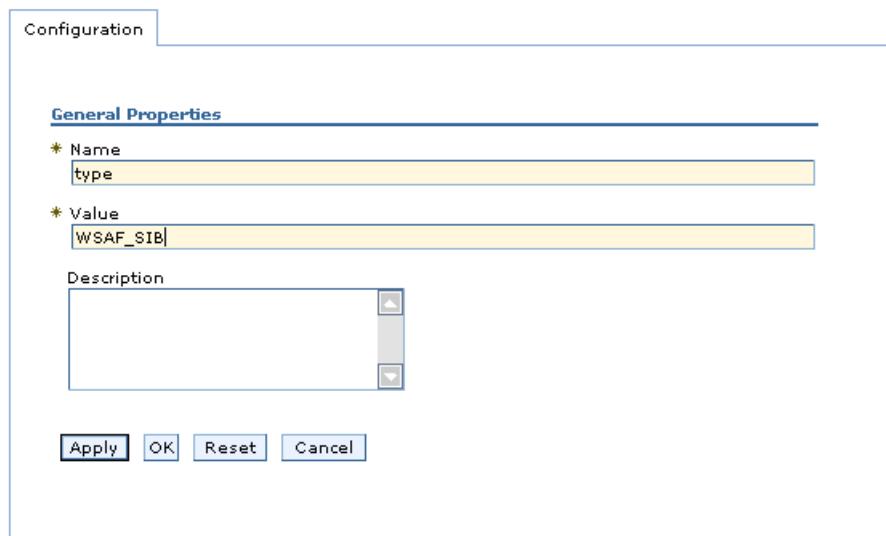


Figure 11-48 Defining match criteria for any messaging engine

Click OK.

10. Click **New** to define another set of match criteria. See Figure 11-49 on page 676.

Enter WSAF_SIB_MESSAGEING_ENGINE for the name and the messaging engine name for the value.

Configuration

General Properties

* Name
WSAF_SIB_MESSAGING_ENGINE

* Value
Cluster1.000-ITSOBus

Description

Apply OK Reset Cancel

This screenshot shows a configuration dialog box with a blue header bar containing the word 'Configuration'. Below the header is a section titled 'General Properties' with a dark blue horizontal line. Underneath this title are two required fields, both marked with an asterisk (*): 'Name' and 'Value'. The 'Name' field contains the value 'WSAF_SIB_MESSAGING_ENGINE' and has a light blue background. The 'Value' field contains the value 'Cluster1.000-ITSOBus' and has a yellowish-orange background. Below these fields is a 'Description' section with a large, empty text area and scroll bars on the right side. At the bottom of the dialog are four buttons: 'Apply', 'OK', 'Reset', and 'Cancel', arranged horizontally.

Figure 11-49 Defining a match criteria for a specific messaging engine

Click **OK**.

- 11.Return to your policy by clicking the policy name in the navigation trail. See Figure 11-50 on page 677.

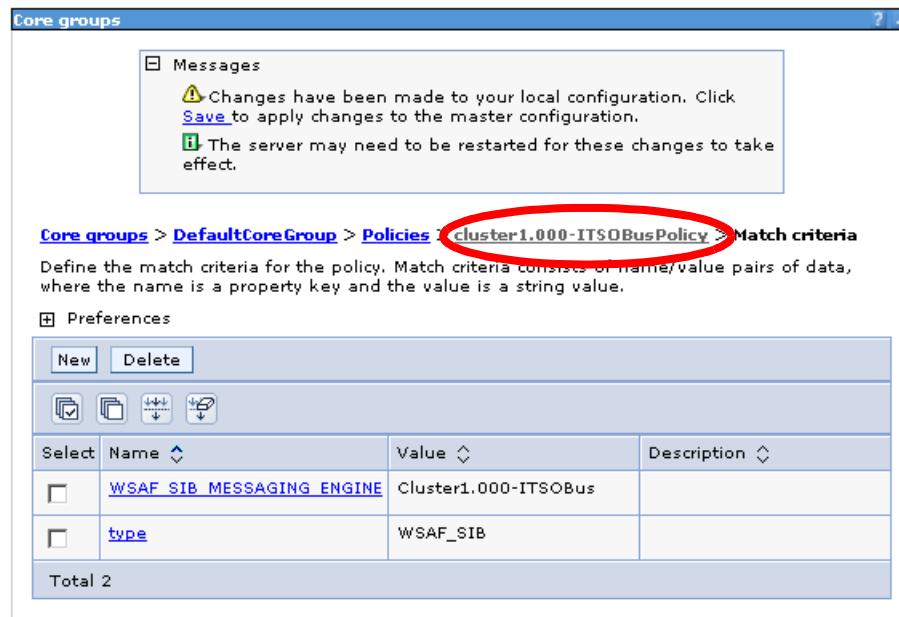


Figure 11-50 Match criteria for a messaging engine

12. Click **Preferred servers** in the Additional Properties section.

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 11-51 on page 678. 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.

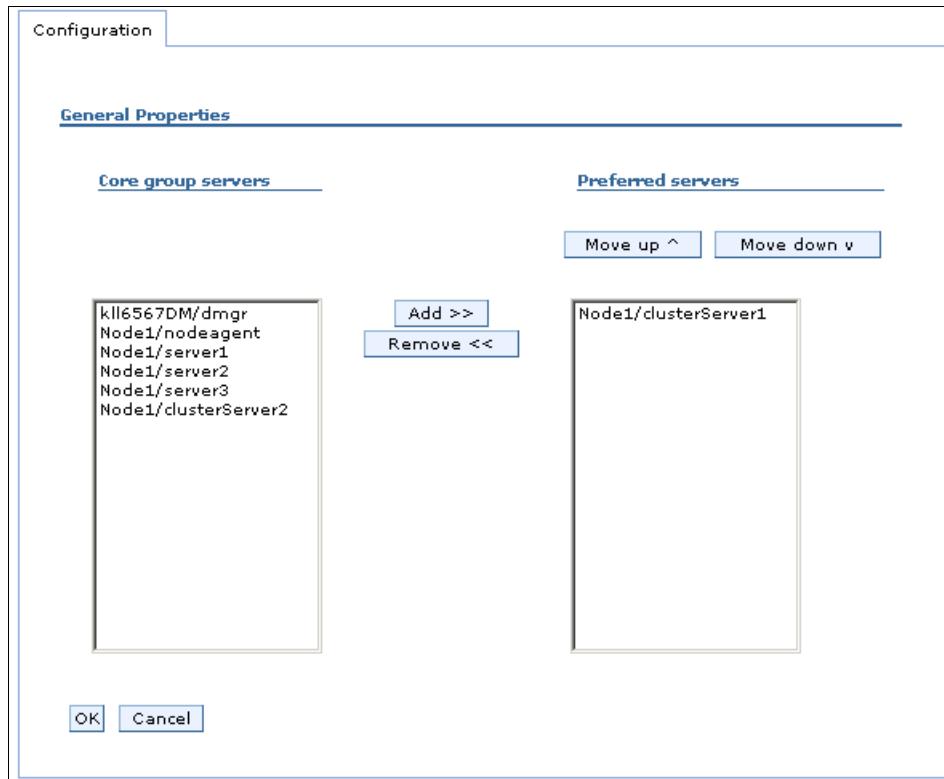


Figure 11-51 Selecting preferred servers for a core group policy

14. Click **OK** and save your changes.

11.8.10 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 Additional Properties section.
4. Click **New**. See Figure 11-52 on page 679.
5. Provide the **Name** of the foreign bus.

Important: When your foreign bus is a service integration bus, then this name must match exactly the name of that bus.

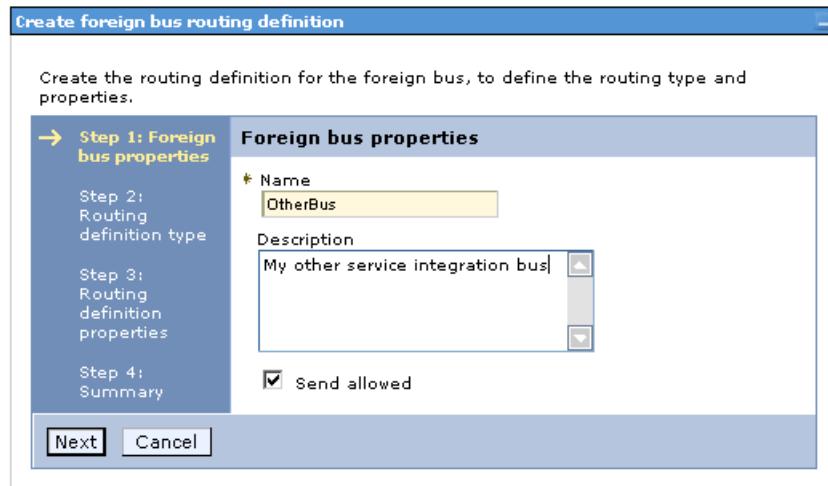


Figure 11-52 Creating a new foreign bus

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 11-53.

6. Select the appropriate value for the routing type.

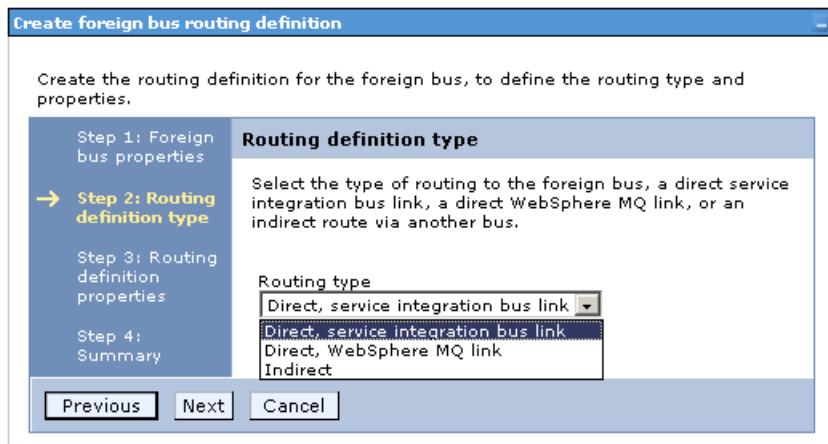


Figure 11-53 Selecting the type of foreign bus

To define another service integration bus, select **Direct, service integration bus link** from the menu. Click **Next**. See Figure 11-54.

7. Optionally, define outbound and inbound user IDs.

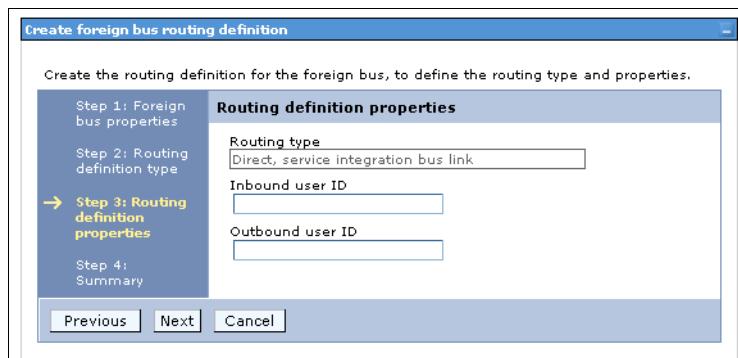


Figure 11-54 Define inbound and outbound user IDs

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 on 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 service integration 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 Additional Properties section and select the messaging engine you want to host the link.
3. Select **Service integration bus link** in the Additional Properties section
4. Click **New**. See Figure 11-55 on page 681 and fill in the following properties:

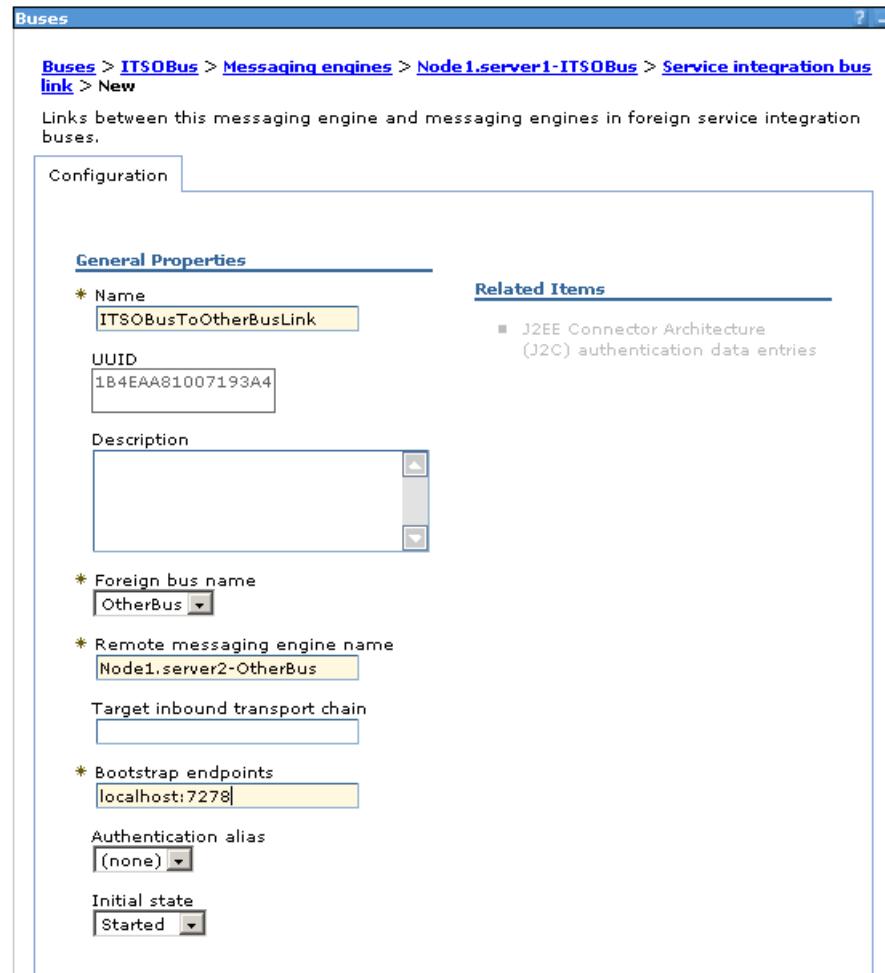


Figure 11-55 Defining a service integration bus link

– **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 10.7.1, “JMS client runtime environment” on page 576.

– **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 service integration 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 Additional Properties section.
- ▶ Select the foreign service integration bus you to which you want to create a topic mapping.
- ▶ Select **Service integration bus link routing properties** in the Additional Properties section.
- ▶ Select **Topic space mappings** 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.

11.8.11 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 11.2.6, “WebSphere MQ links” on page 629.

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 Additional Properties section.
3. Click **New**. See Figure 11-56. Enter information in the following fields.

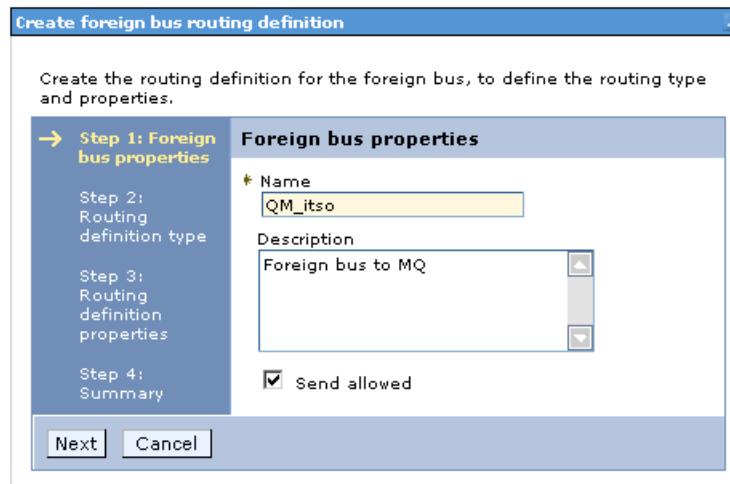


Figure 11-56 creating a new foreign bus

– **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 11-57.

4. Select **Direct, WebSphere MQ link** from the menu.

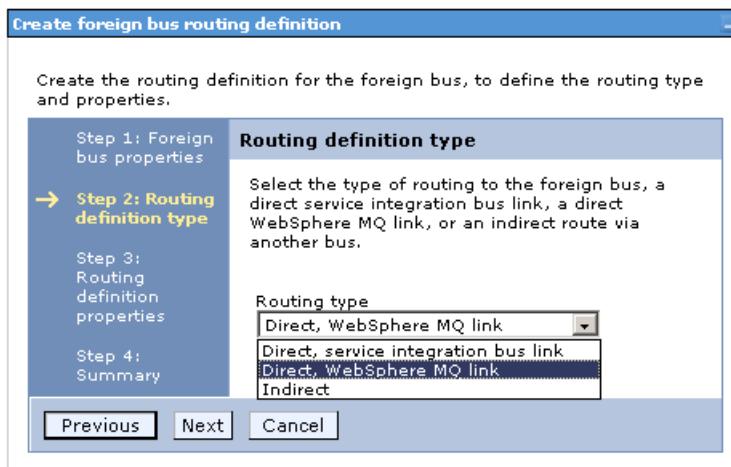


Figure 11-57 Selecting the type of foreign bus

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 11-58 on page 686 and enter information in the following fields.

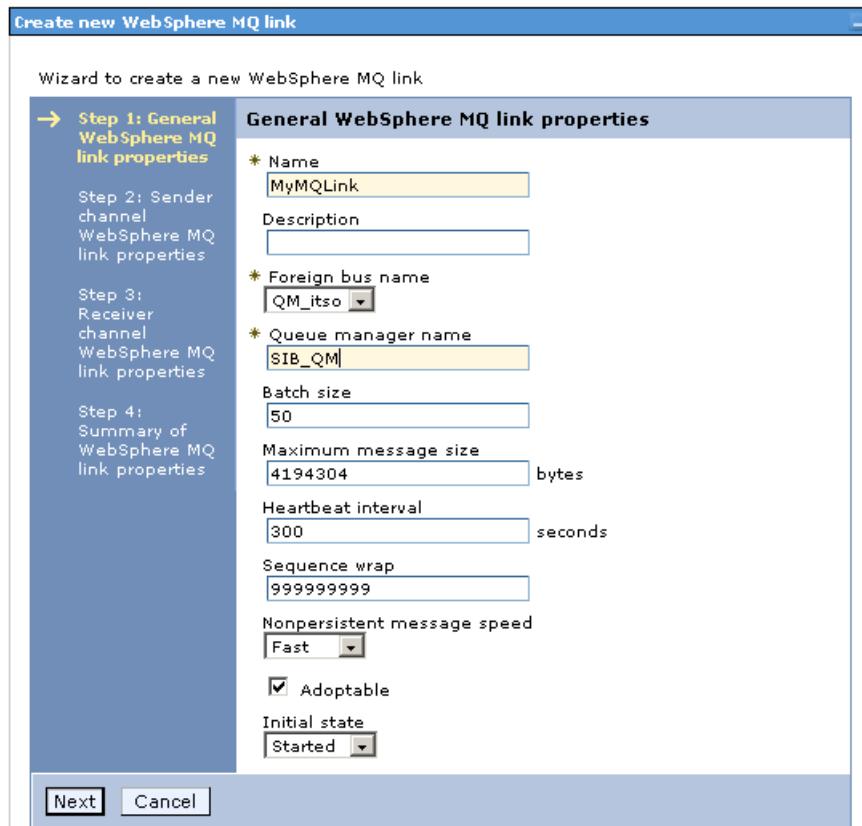


Figure 11-58 Defining properties for a new MQ link

– **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 define 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 11-59. Enter the following:

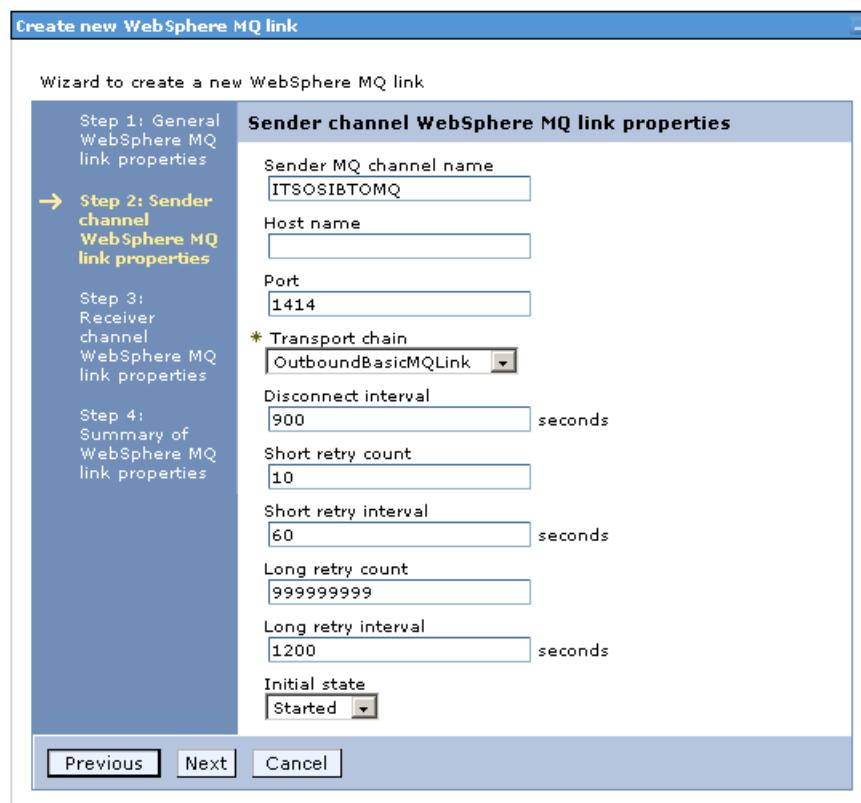


Figure 11-59 Providing the link with details on how to send messages to MQ

- **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 11.2.2, “Service integration bus transport chains” on page 614 for further information.

Click **Next**. See Figure 11-60.

7. Enter information about the virtual queue manager. Remember, this link performs as a virtual queue manager for WebSphere MQ. Enter the following information:

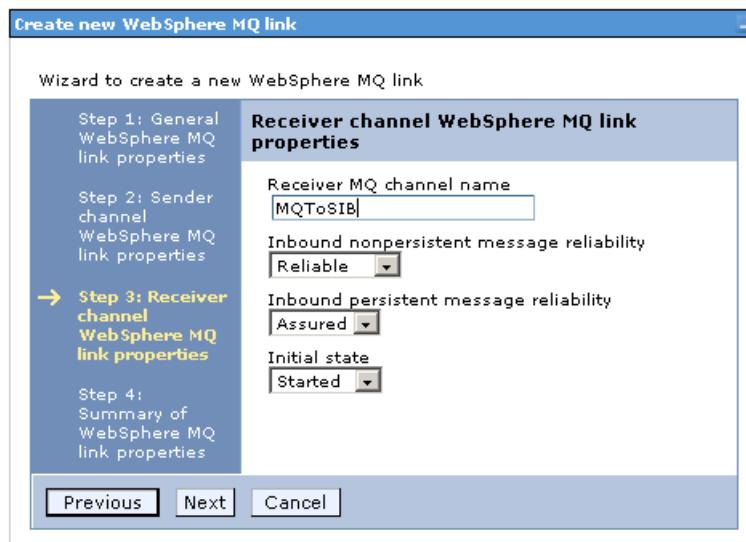


Figure 11-60 Providing the link with details on how MQ will send messages to it

– **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 604 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 11-61 and enter information in the following fields:

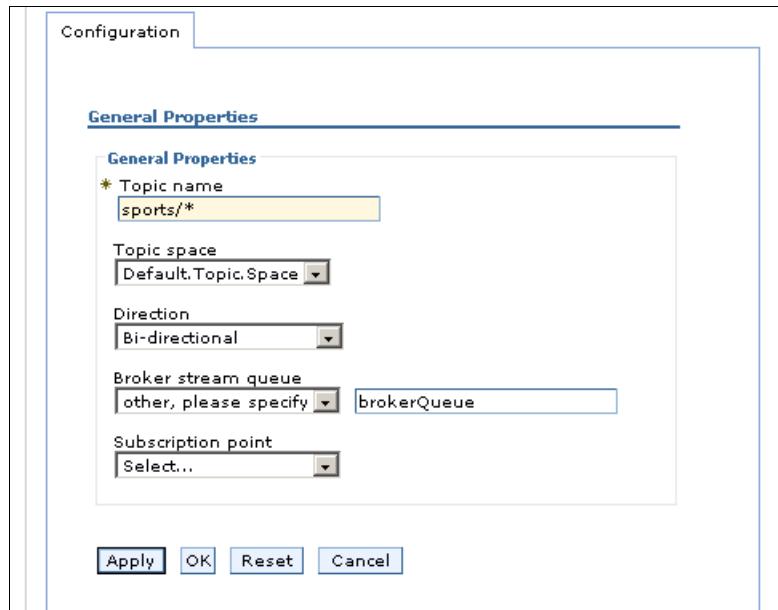


Figure 11-61 Defining a new Topic mapping

– 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 service integration bus wildcards in this topic name to map a group of topics. For more information about the use of wildcard characters when specifying topic names, refer to “Topic specific connection properties” on page 537.

– **Topic space**

Select the Topic space in which this topic will be published on the service integration bus.

– **Direction**

Select the desired direction of the mapping.

- **Bi-directional**

Messages published in either WebSphere MQ or the service integration bus will be published in both.

- **To WebSphere MQ**

Messages published in the service integration bus will be published in WebSphere MQ, but messages published in WebSphere MQ will not be published in the service integration bus.

- **From WebSphere MQ**

Messages published in WebSphere MQ will be published in the service integration bus, but messages published in the service integration 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 dropdown 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.

11.8.12 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 Additional Properties section.
4. Click **New**.
5. Select **Foreign** from the list and click **Next**. Enter information as in Figure 11-62.

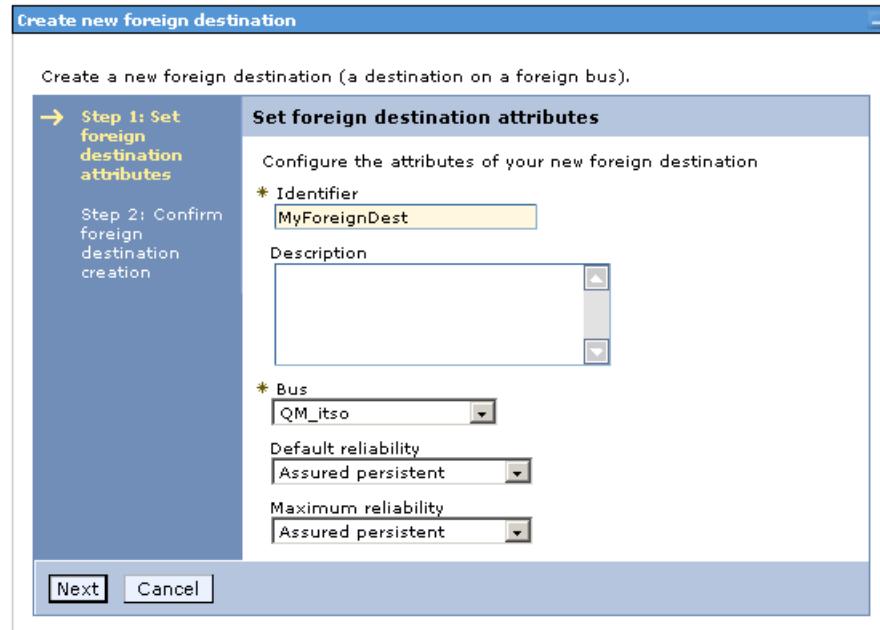


Figure 11-62 Creating a new foreign destination

– 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.

– Default reliability and Maximum reliability

Select the desired default and maximum reliabilities from the drop-down menus. Consult “Reliability” on page 604.

Click **Next**.

6. Click **Finished**.

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 runtime environment.

This part includes the following chapters:

- ▶ Chapter 12, “Session management” on page 697
- ▶ Chapter 13, “WebSphere naming implementation” on page 769
- ▶ Chapter 14, “Understanding class loaders” on page 821
- ▶ Chapter 15, “Packaging applications” on page 847
- ▶ Chapter 16, “Deploying applications” on page 907
- ▶ Chapter 17, “WebSphere Rapid Deployment” on page 957



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:

- ▶ 12.1, “What is new?” on page 698
- ▶ 12.2, “HTTP session management” on page 698
- ▶ 12.3, “Session manager configuration” on page 699
- ▶ 12.4, “Session scope” on page 700
- ▶ 12.5, “Session identifiers” on page 702
- ▶ 12.6, “Local sessions” on page 710
- ▶ 12.7, “General properties for session management” on page 711
- ▶ 12.8, “Session affinity” on page 715
- ▶ 12.9, “Persistent session management” on page 719
- ▶ 12.10, “Invalidating sessions” on page 749
- ▶ 12.11, “Session security” on page 751
- ▶ 12.12, “Session performance considerations” on page 752
- ▶ 12.13, “Stateful session bean failover” on page 760
- ▶ 12.11, “Session security” on page 751

12.1 What is new?

The following improvements have been made to session management in WebSphere Application Server V6:

- ▶ The Data Replication Service (DRS) that provides session persistence using memory-to-memory replication has the following improvements:
 - Simplified topology
 - Simplified configuration
 - Improved performance with faster underlying transport
 - Integration with workload management (WLM) to provide hot failover in peer-to-peer mode
- ▶ Stateful session bean failover support

WebSphere Application Server V6 utilizes the functions of DRS and WLM to enable stateful session bean failover.

Note: The above features are only available for Network Deployment distributed server environments.

12.2 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*.

12.3 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.

12.3.1 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* select 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.

12.3.2 Accessing session management properties

You can access all 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. Click **Servers** → **Application servers**.
2. Click the application server.
3. In the Container Settings section of the Configuration tab, click **Web Container Settings**.
4. 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 Additional 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 the administrative console, do the following:

1. Click **Applications** → **Enterprise Applications**.
2. Click the application.
3. In the Related Items section of the Configuration tab, click **Web modules**.
4. Click the Web module.
5. In the Additional Properties section, click **Session Management**.

12.4 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.xmi 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 12-1 on page 702.

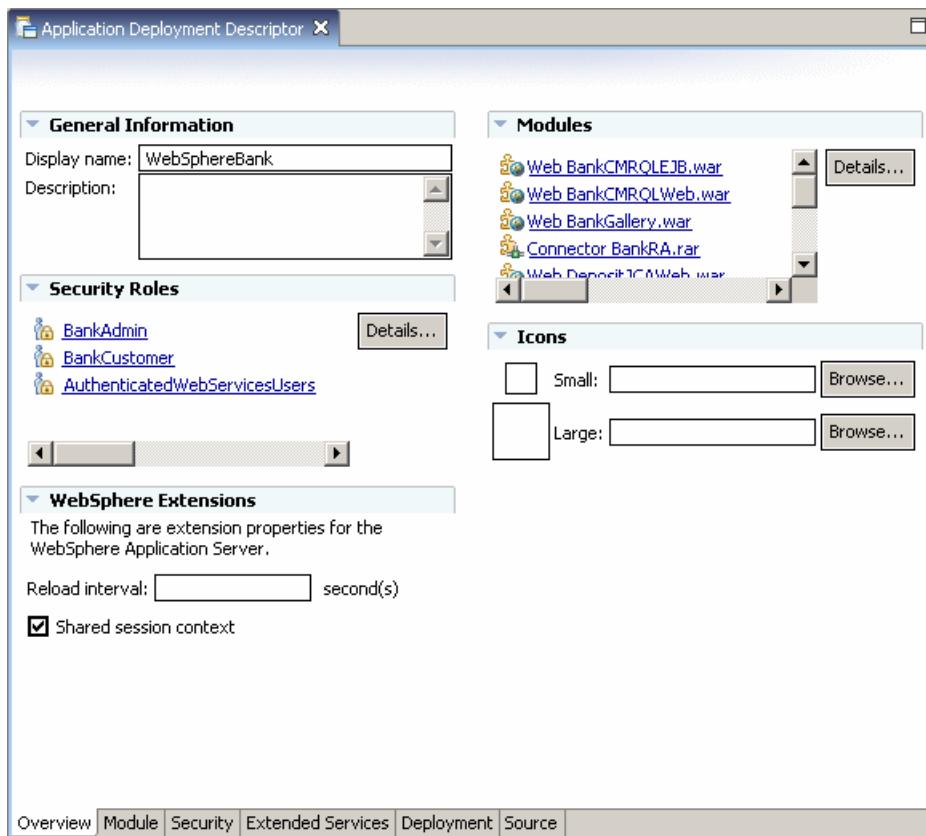


Figure 12-1 Shared HTTP session context using the Application Server Toolkit

5. Save and close the deployment descriptor.

12.5 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.

12.5.1 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 12-2 on page 704.

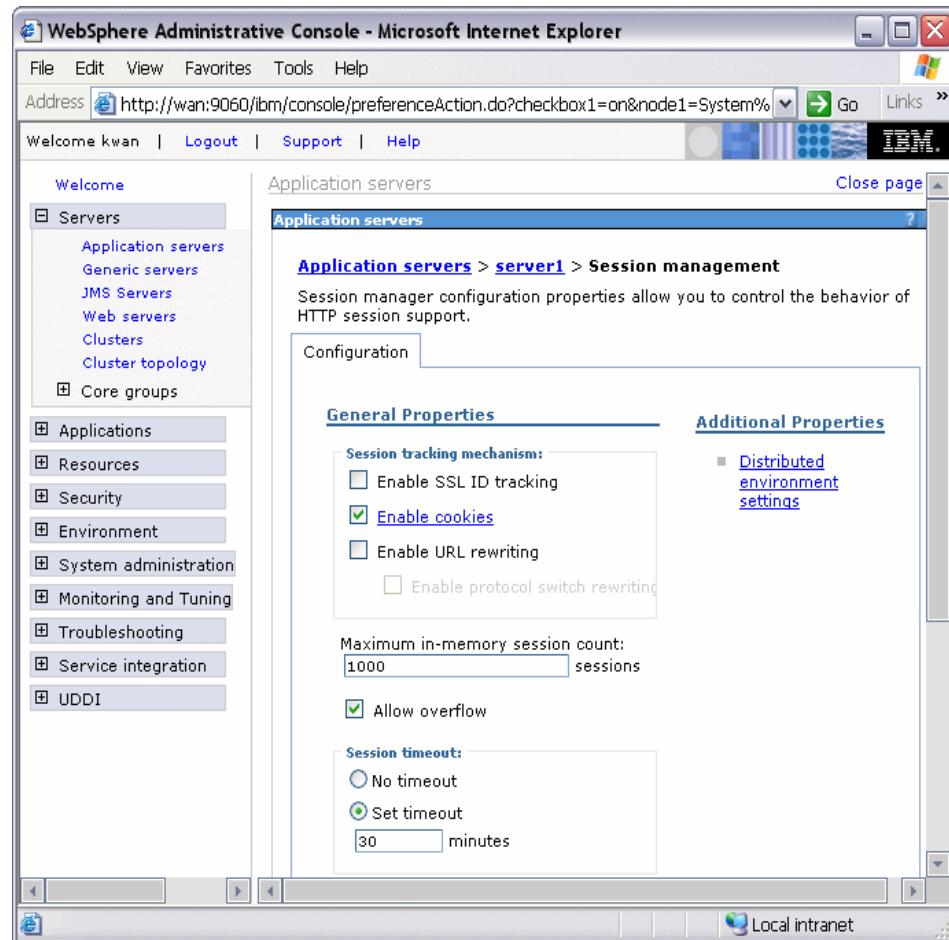


Figure 12-2 Selecting a session tracking mechanism window

3. Click **OK**.
4. Save and synchronize the configuration changes.
5. Restart the application server or the cluster.

12.5.2 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.

12.5.3 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 12-3 on page 706.

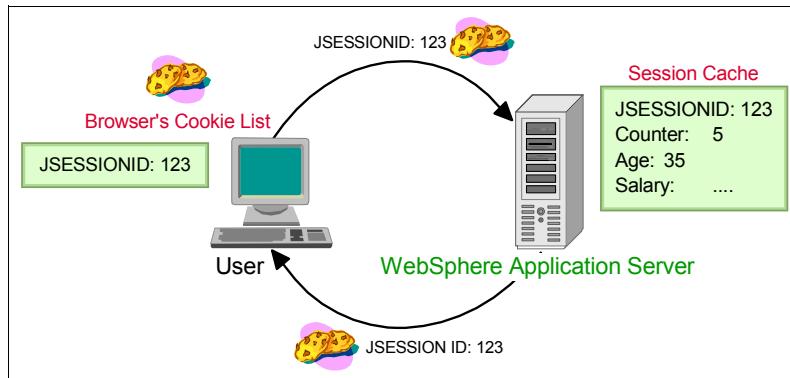


Figure 12-3 Cookie overview

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 nor 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 12-4 on page 707.

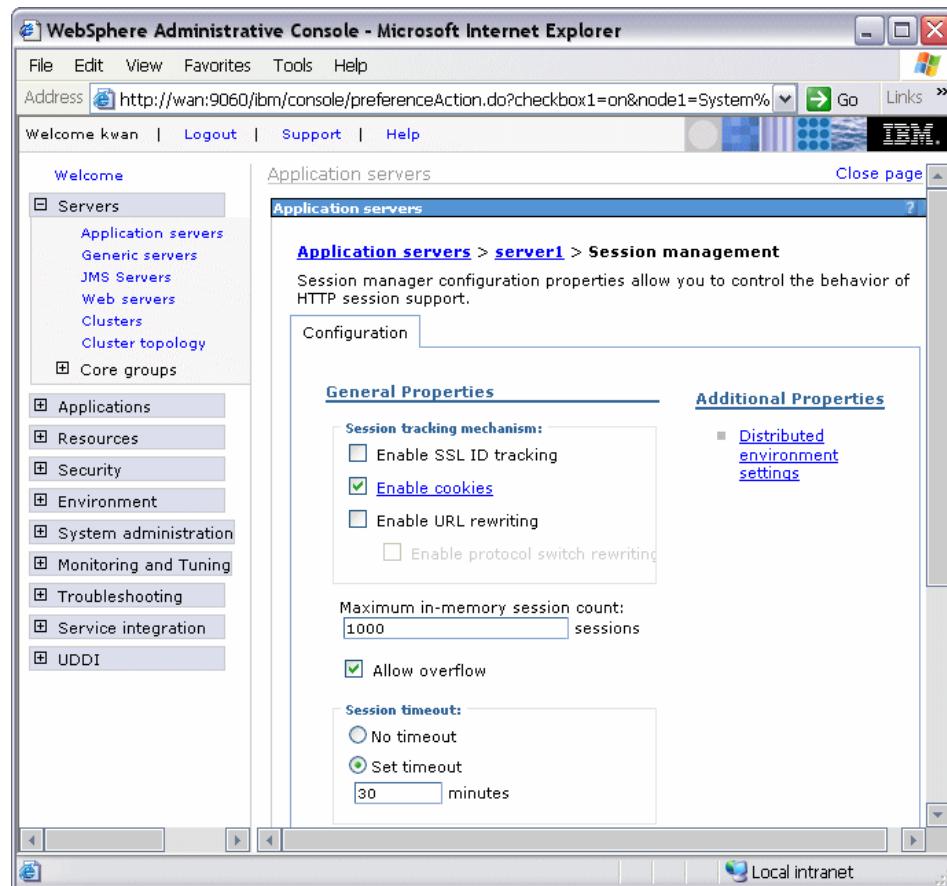


Figure 12-4 Session tracking mechanism

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 global 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

12.5.4 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 12-1, to encode the returning content.

Example 12-1 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 12-2.

Example 12-2 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.

12.6 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.

12.7 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 12-5 on page 712/

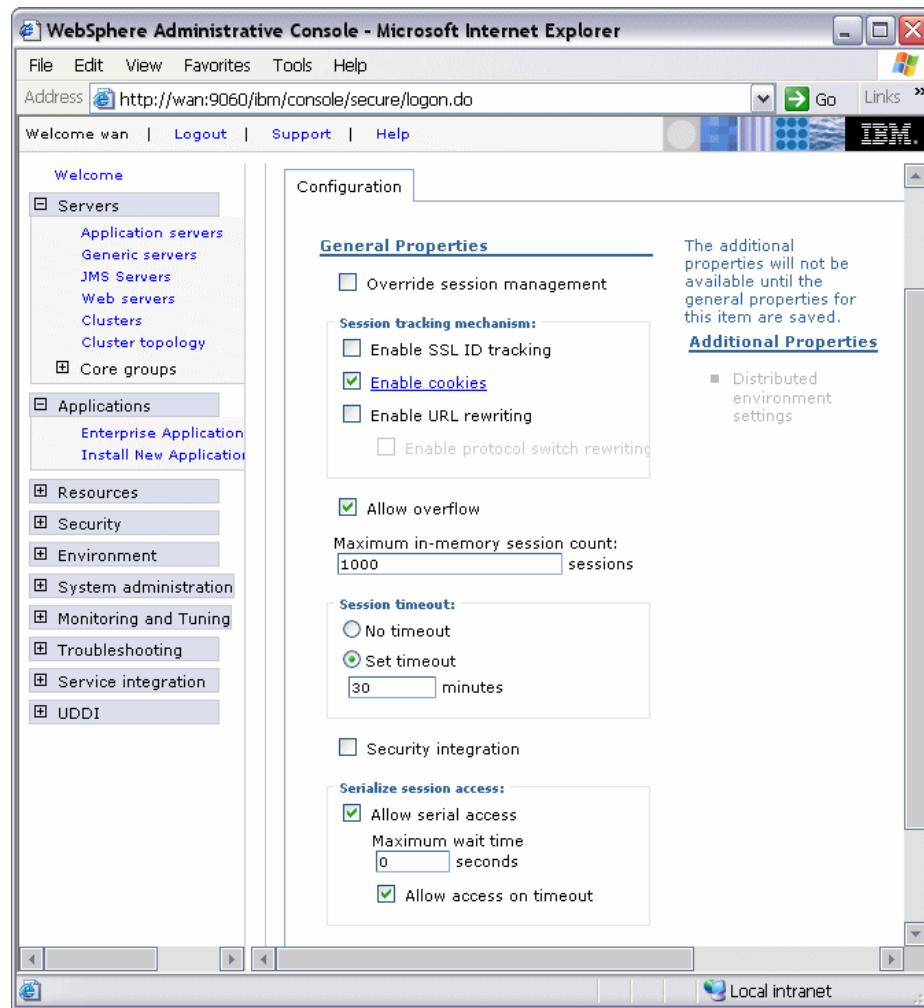


Figure 12-5 Session Management window 2

► 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 **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 12.6, “Local sessions” on page 710.

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 12.6, “Local sessions” on page 710.

As shown in Example 12-3, the IBM HttpSession extension can be used to react if sessions exceed the maximum number of sessions specified when overflow is disabled.

Example 12-3 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 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 12.11, "Session security" on page 751 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.

12.8 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 12-6 on page 716, 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.

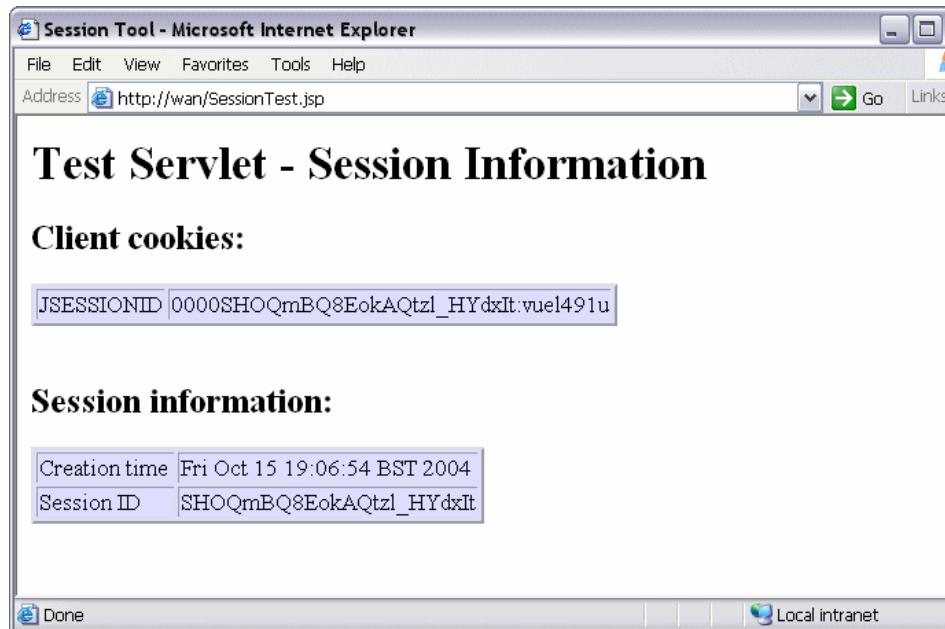


Figure 12-6 Session ID containing the server ID and cache ID

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 12-1 shows their mappings based on the example in Figure 12-6. A clone ID is an ID of a cluster member.

Table 12-1 Cookie mapping

content	value in the example
Cache ID	0000
Session ID	SHOQmBQ8EokAQtzl_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 12-4.

Example 12-4 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="vue1491u" LoadBalanceWeight="2" Name="NodeA_server1">  
            <Transport Hostname="wan" Port="9080" Protocol="http"/>  
            <Transport Hostname="wan" Port="9443" Protocol="https"/>  
.....  
</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 don't rely on the value of the session cookie remaining the same for a given session.

12.8.1 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 12-7 on page 718.

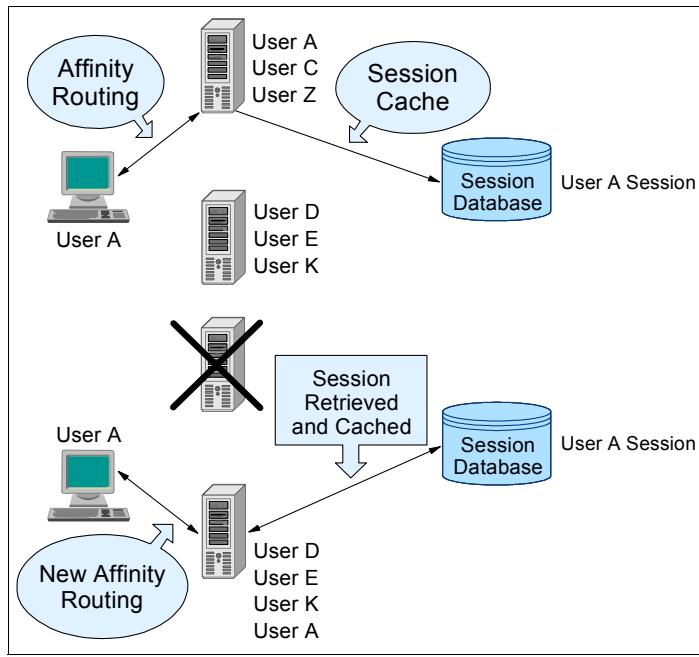


Figure 12-7 Session affinity and failover

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 narrowed 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.

12.9 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 12-8:

- ▶ Database persistence
- ▶ Memory-to-memory session state replication using the data replication service available in distributed server environments

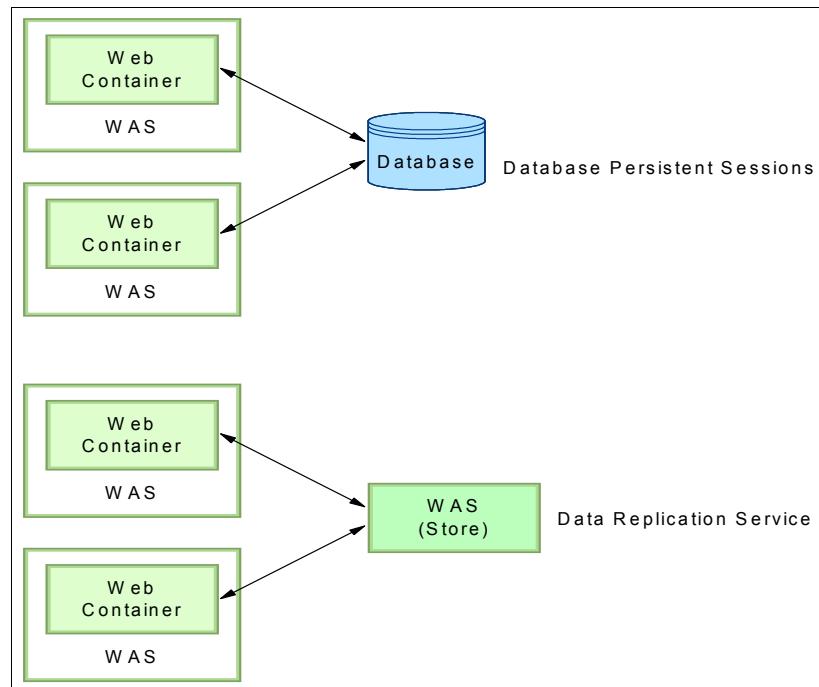


Figure 12-8 Persistent session options

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 12.9.6, “Single and multi-row schemas (database persistence)” on page 743 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.

12.9.1 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. Create a JDBC provider and data source. See 7.2, “JDBC resources” on page 321 and 7.2.3, “Creating a data source” on page 326.

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. Click **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 12-9.

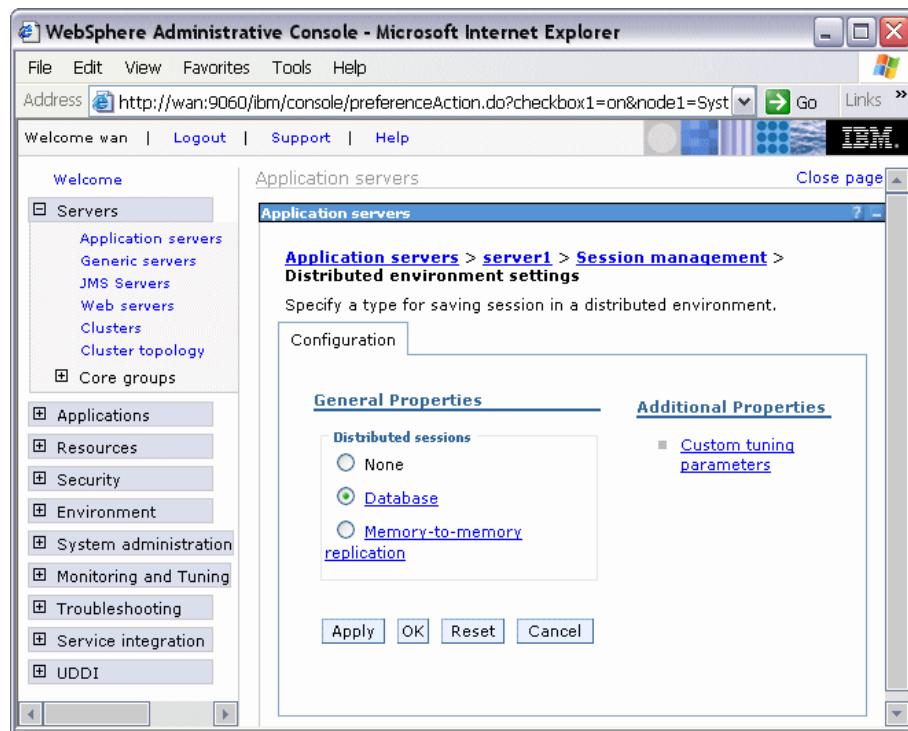


Figure 12-9 Distributed Environment Setting (database)

6. Enter the database information:
 - a. Enter the data source JNDI name. The data source must be a non-JTA enabled data source.
 - b. Enter the user ID and password to access the database.
 - c. 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 12.9.5, "Larger DB2 page sizes and database persistence" on page 742 for more information.
 - d. 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 742.
 - e. If you intend to use a multi-row schema, select **Use Multi row schema**. See 12.9.6, "Single and multi-row schemas (database persistence)" on page 743 for more information. See Figure 12-10.

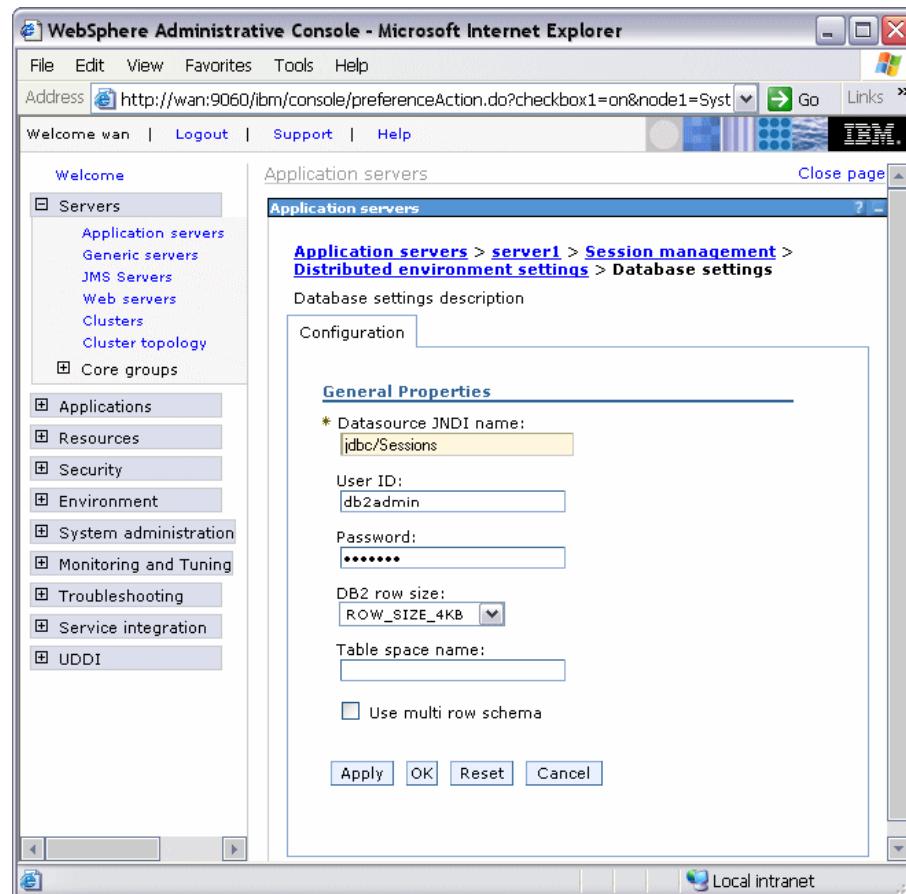


Figure 12-10 Database

7. Click **OK**.

After you have updated each server, save the configuration changes, synchronize them with the servers, and restart the application servers.

12.9.2 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 panels 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 12-11 on page 726 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.

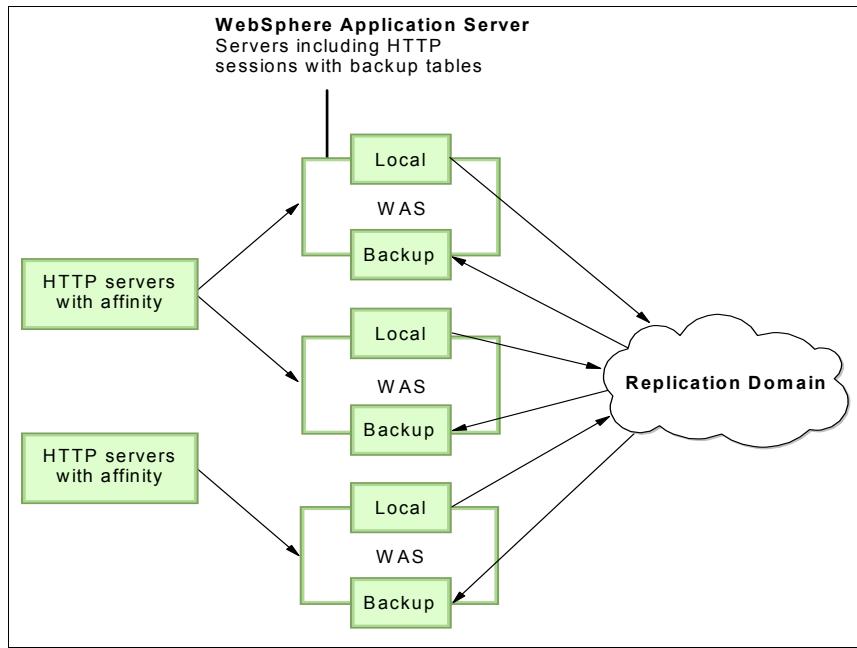


Figure 12-11 Example of peer-to-peer topology

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 12-12 on page 727 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.

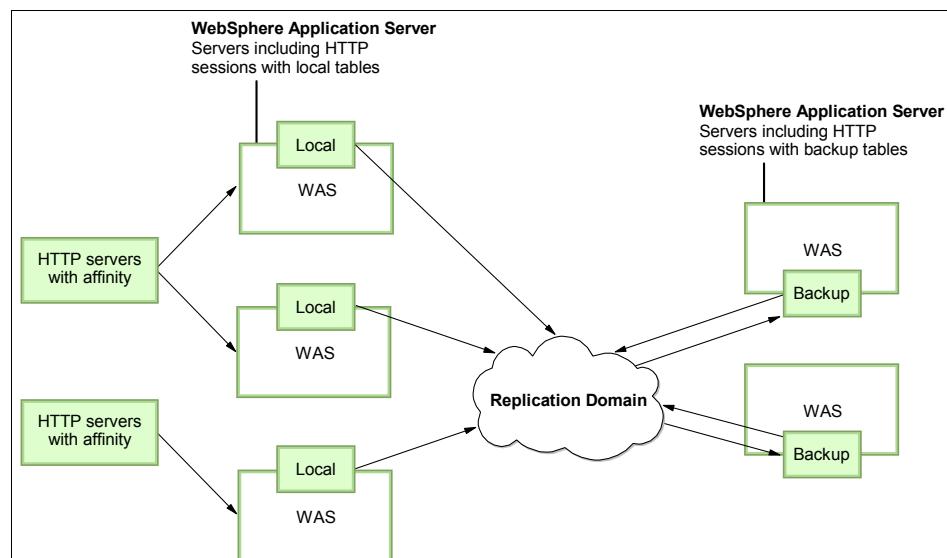


Figure 12-12 Example of client/server topology

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.

WebSphere Application Server V6 encourages the creation of 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 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 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 12-13 on page 730, and enter information in the fields

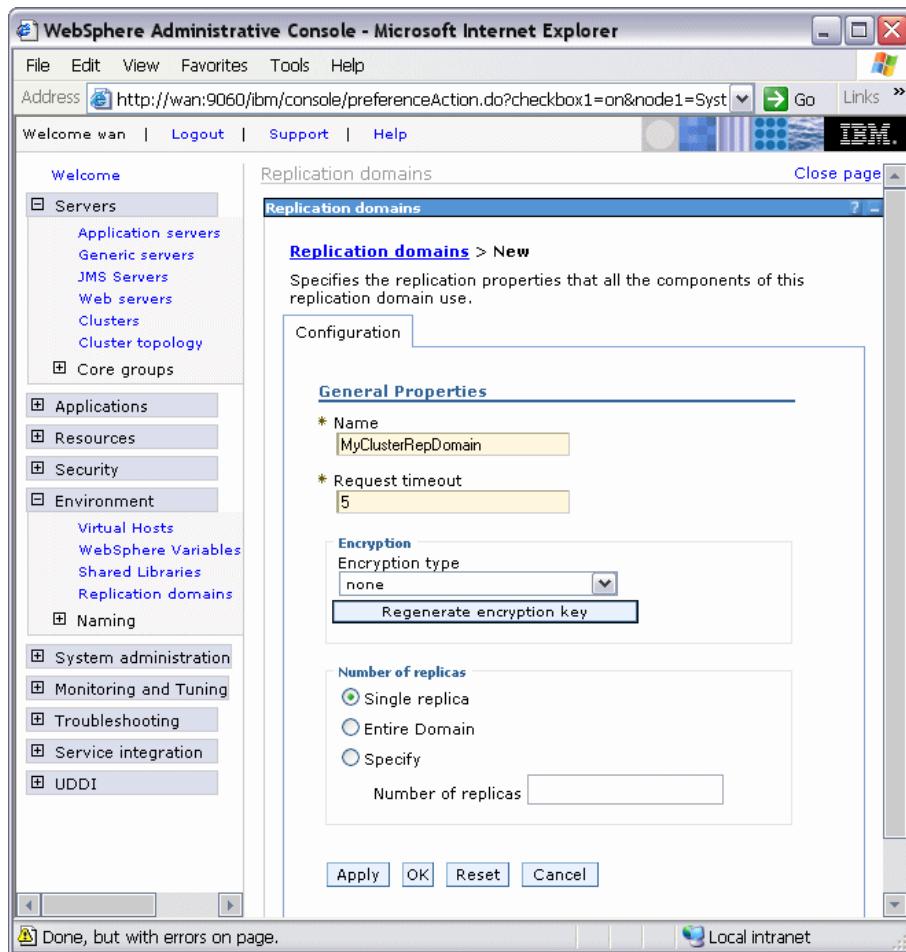


Figure 12-13 Create a replication domain

– 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 name and defaults are used as 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 12-14 on page 732.

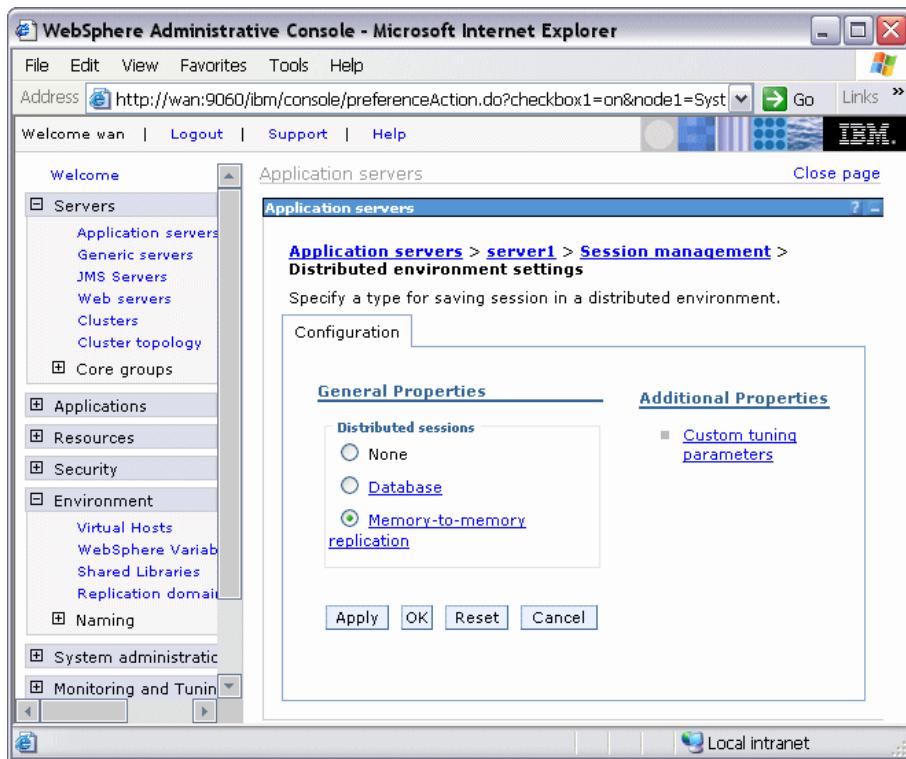


Figure 12-14 Distributed environment settings

- g. Choose a replicator domain and replicator mode either from listed domains. See Figure 12-15 on page 733.

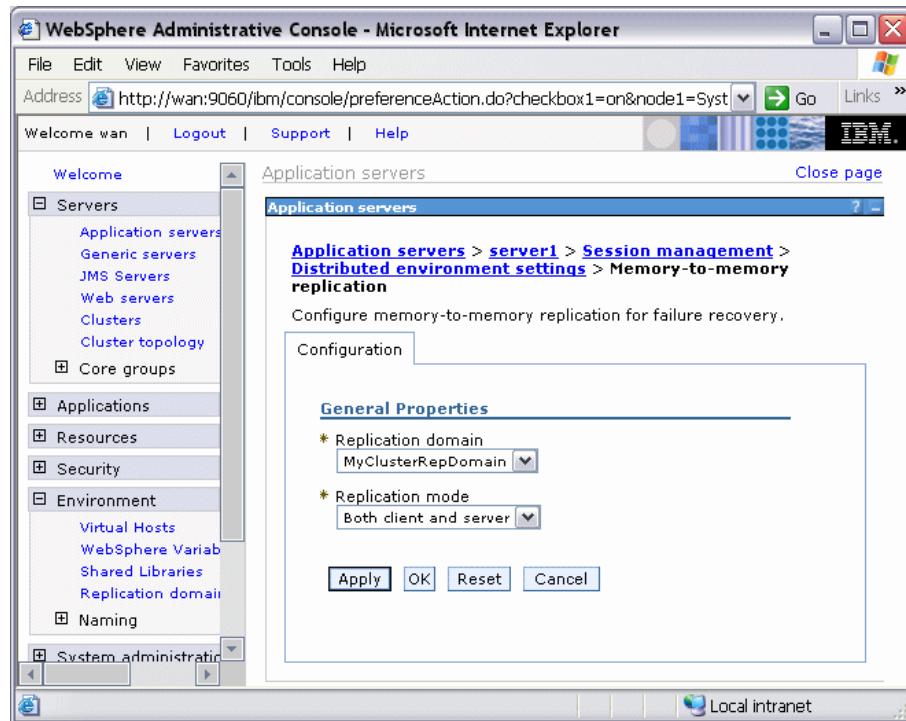


Figure 12-15 Data replication service settings

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 12-5 on page 734.

Example 12-5 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" numberOfWorkers="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 12-6.

Example 12-6 Server.xml updates

```
<sessionDRSPersistence xmi:id="DRSSettings_1097867741921"
messageBrokerDomainName="MyClusterRepDomain"/>
```

12.9.3 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 12.12.2, “Reducing persistent store I/O” on page 756 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 12-7 shows how the `IBMSession` class can be used to manually update the persistent store.

Example 12-7 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 HttpSession. 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 12.9.7, “Contents written to the persistent store using a database” on page 745 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. *Invalidate sessions* are sessions that are no longer in use and timed out. For more information, see 12.10, “Invalidate sessions” on page 749. 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 700) or at the Web module level (see “Web module session management properties” on page 700).
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 12-16 on page 740.

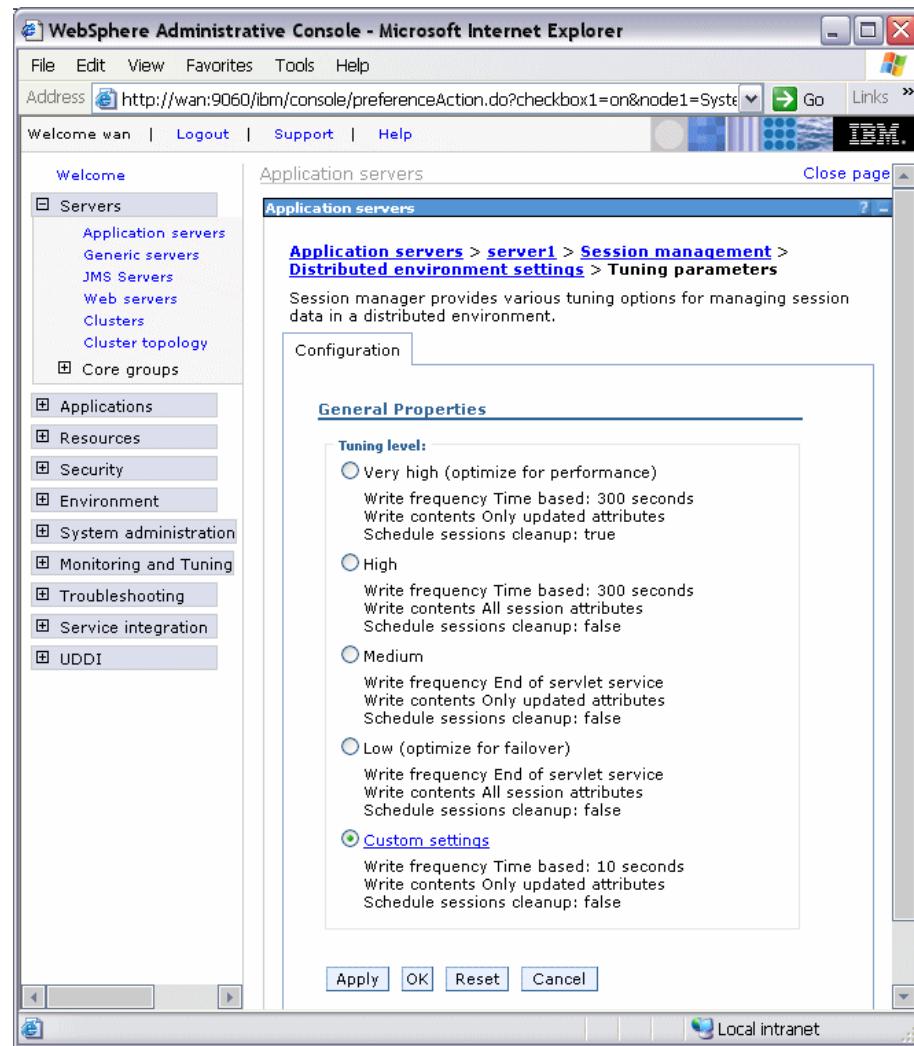


Figure 12-16 Session management tuning parameters

If you want to set each tuning parameter explicitly, select **Custom settings**. See Figure 12-17 on page 741.

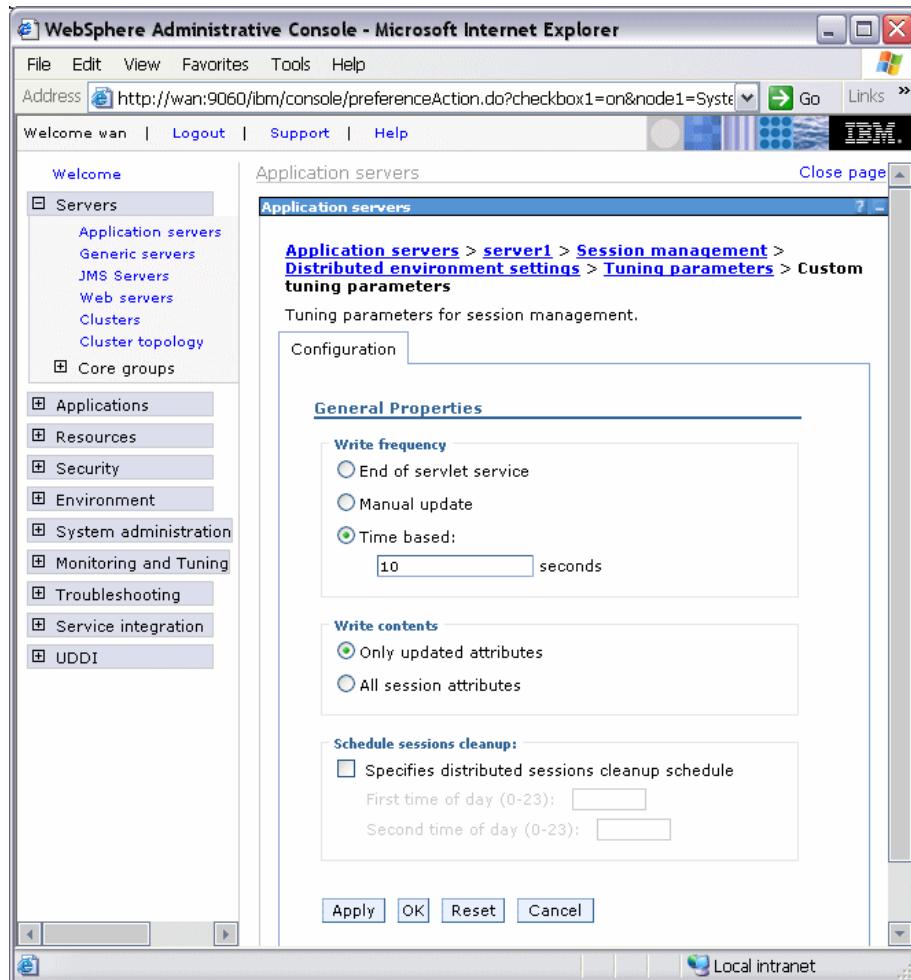


Figure 12-17 Session management tuning parameters

12.9.4 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.

12.9.5 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 12-8 shows simple steps for creating an 8 KB page:

Example 12-8 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 12-10 on page 723.

Restart WebSphere. On startup, the session manager creates a new SESSIONS table based on the page size and tablespace name specified.

12.9.6 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 12.12.3, "Multirow persistent sessions: Database persistence" on page 757.

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 12-10 on page 723.
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 12.9.1, “Enabling database persistence” on page 721.
 - 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 12-2 provides an overview.

Table 12-2 Single versus multi-row schemas

Programming issue	Application scenario
Reasons to use single-row	<ul style="list-style-type: none">▶ 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	<ul style="list-style-type: none"> ▶ 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.

12.9.7 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 12-9 and Example 12-10, the initial session creation contains a `setAttribute`, but subsequent requests for that session do not need to use `setAttribute`.

Example 12-9 Initial servlet

```
HttpSession sess = request.getSession(true);
myClass myObject = new myClass();
myObject.someInt = 1;
sess.setAttribute("myObject", myObject); // Bind object to the session
```

Example 12-10 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 12-11 and Example 12-12 show setAttribute is still required even though the write all session attributes option is enabled.

Example 12-11 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 12-12 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 12-3 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 12-3 Write contents versus write frequency

Row type	Write contents	Write frequency	Action for setAttribute	Action for removeAttribute
Single-row	Only updated attributes	End of servlet service / sync() 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.

12.10 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 invalidation 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, 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 12.10.1, “Session listeners”, processing is potentially delayed by this configuration. It is not recommended if listeners are used.

12.10.1 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 on memory or not, when persistent session is used.

Table 12-4 is a summary of the interfaces and methods.

Table 12-4 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>

12.11 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 Security V6 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 SWAM (Simple WebSphere Authentication Mechanism).

Table 12-5 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 12-5 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. UnauthorizedSession-RequestException 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. UnauthorizedSession-RequestException is thrown ¹ .	The session is not returned. UnauthorizedSession-RequestException is thrown ¹ .
¹ com.ibm.websphere.servlet.session.UnauthorizedSessionRequestException is thrown to the servlet or JSP.		

See 12.7, “General properties for session management” on page 711 for more information about the security integration setting.

12.12 Session performance considerations

This section includes guidance for developing and administering scalable, high-performance Web applications using WebSphere Application Server session support.

12.12.1 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:

$$1 \text{ MB for each user session} * 100 \text{ users} = 100 \text{ MB}$$

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 12.9.5, “Larger DB2 page sizes and database persistence” on page 742 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 12.6, “Local sessions” on page 710. 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 spread 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 12.10, “Invalidating sessions” on page 749.

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 12.10, “Invalidate sessions” on page 749.

12.12.2 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.

12.12.3 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 12-6 shows a simplified representation of a multi-row database table.

Table 12-6 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 12.9.6, "Single and multi-row schemas (database persistence)" on page 743. Also see 12.9.7, "Contents written to the persistent store using a database" on page 745.

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.

12.12.4 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 cab 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 2 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.

12.12.5 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.

12.13 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.

12.13.1 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. Click **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 panel where you can create one. If at least one domain is configured, the link takes you to a panel where you can select the replication settings to be used by the EJB container. See Figure 12-18 on page 762.

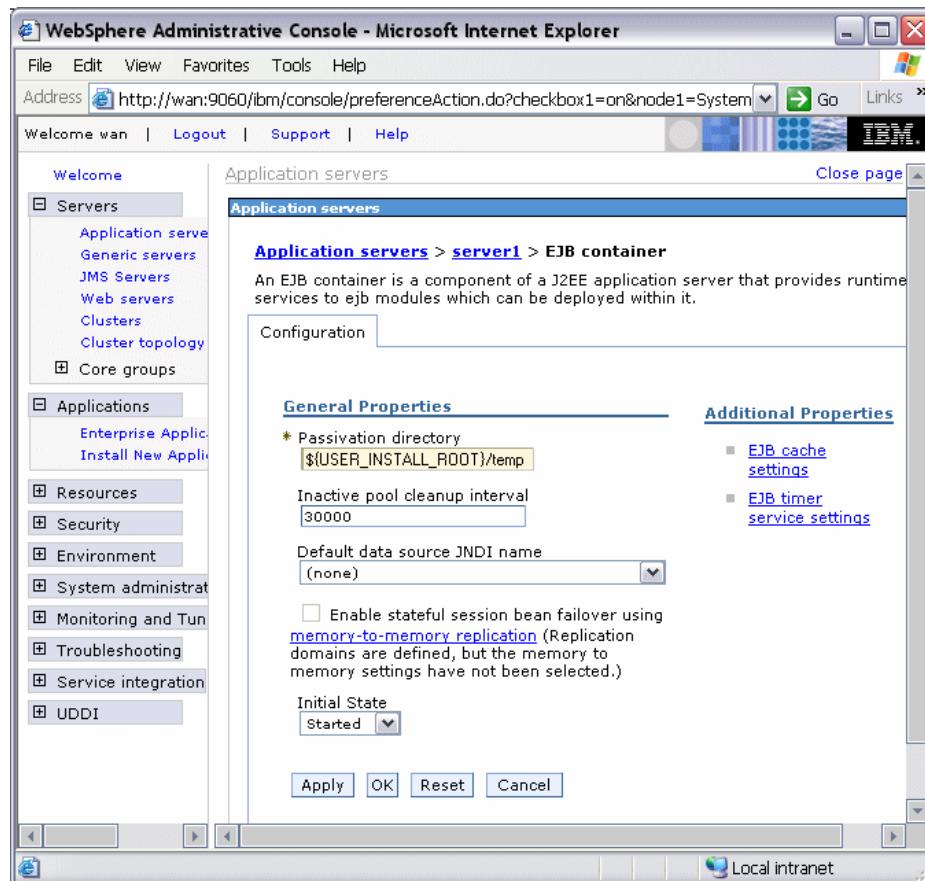


Figure 12-18 Stateful session bean failover settings at the container level

EJB module stateful session bean failover properties

To access stateful session bean failover properties at the EJB module level from the administrative console:

1. Click **Applications** → **Enterprise applications**.
2. Click the application.
3. In the Additional 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 12-19 on page 763.

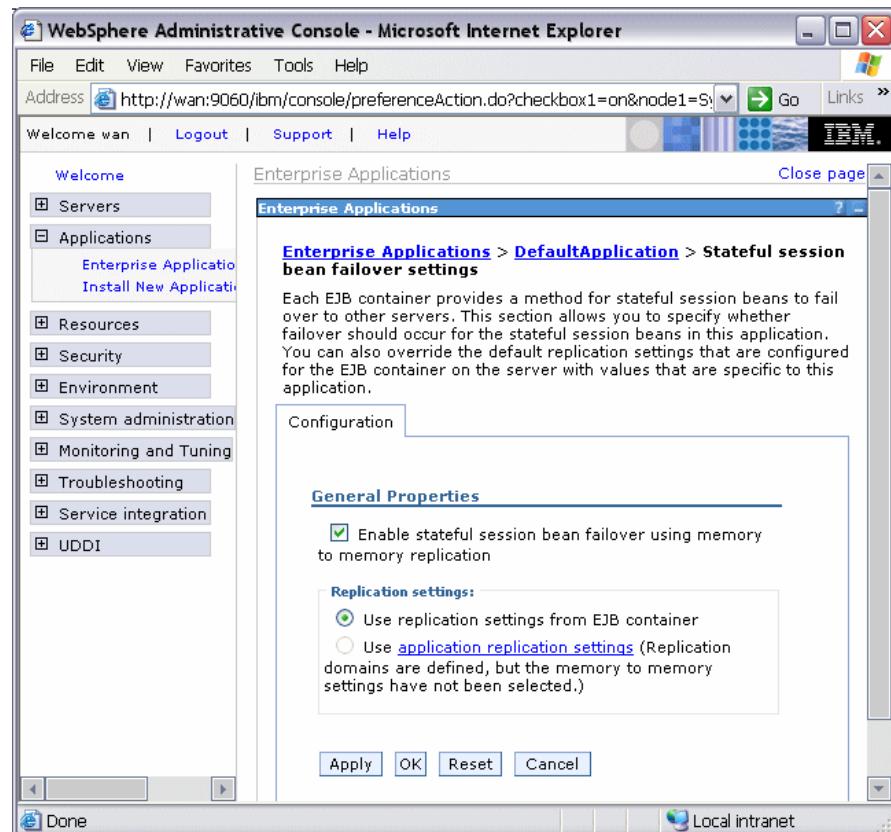


Figure 12-19 Stateful session bean failover settings at the module level

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 panel 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 panel to create one. If at least one domain is configured, the link takes you to a panel 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** or
- **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.

12.13.2 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 thought 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.

J2EE 1.4 specification has added additional requirements for Http Sessions which required 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 in WebSphere Application Server V6
- ▶ WebSphere naming architecture
- ▶ Interoperable Naming Service (INS)
- ▶ Distributed CosNaming
- ▶ Configured bindings
- ▶ Initial contexts
- ▶ Federation of name spaces
- ▶ Interoperability
- ▶ Examples
- ▶ Naming tools
- ▶ Configuration

13.1 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.

13.2 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 13.3, “Interoperable Naming Service (INS)” on page 785 and 13.4, “Distributed CosNaming” on page 787 respectively.

13.2.1 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 13-1 summarizes the naming architecture and its components.

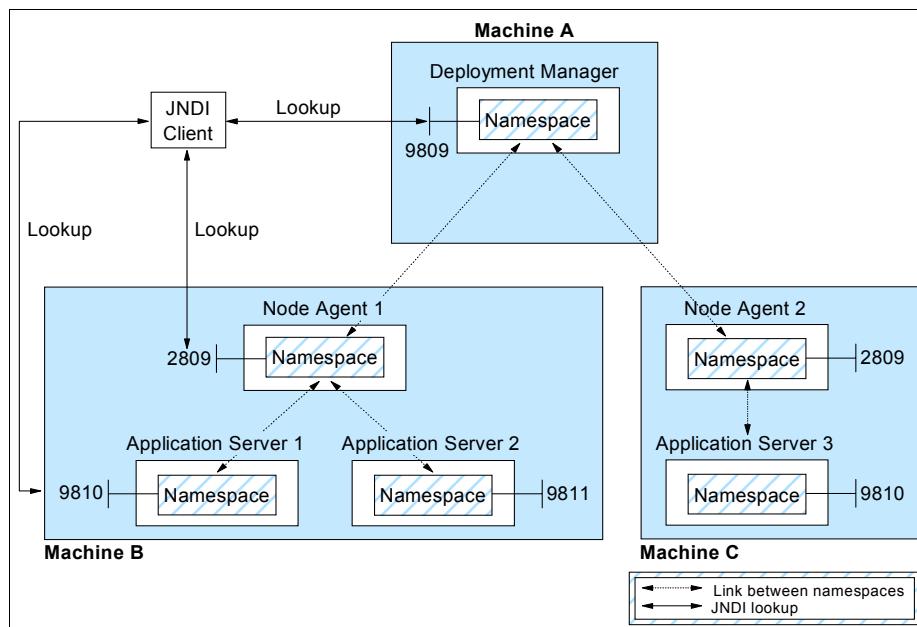


Figure 13-1 Naming topology

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.

13.2.2 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 Microsystem's implementation of:

- ▶ javax.naming.ldap
- ▶ com.sun.jndi.ldap.LdapCtxFactory

13.2.3 JNDI bindings

There are three options available for binding EJB (<ejb-ref>) and resource (<resource-ref>) object names to the WebSphere Application Server 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:

```
ce11/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
```

13.2.4 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 13-2 on page 775.

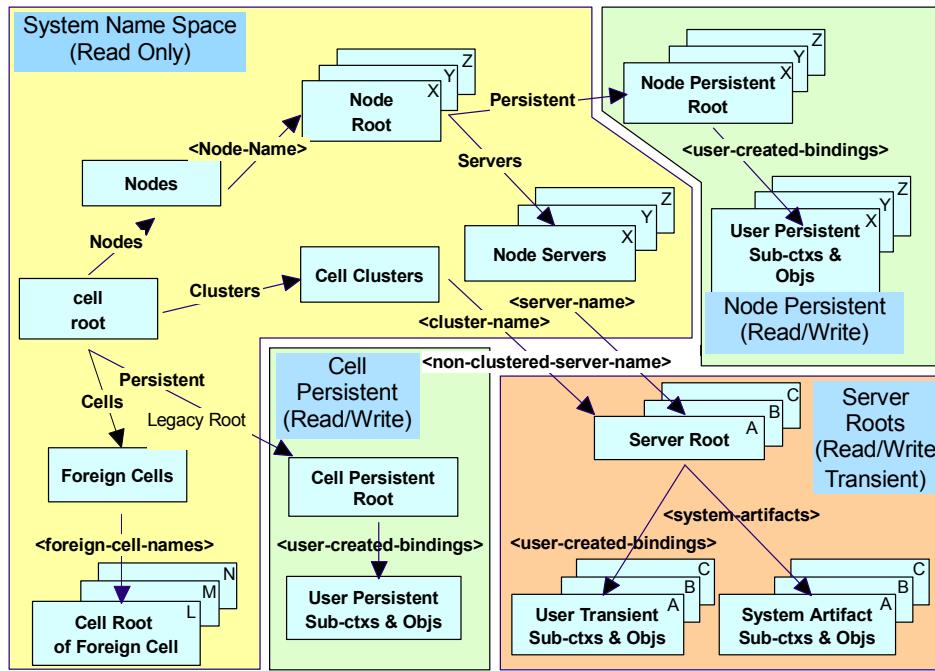


Figure 13-2 Federated name space

The name space can be broken down into distinct partitions that are updateable and persistent.

System partition

The system partition is a reflection of the cell topology and is read-only. This part of the name space cannot be changed programmatically, because it is based on the configuration rather than runtime settings.

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.

This partition of the name space is persistent, with the data stored in the XML repository containing the topological information.

Cell and node persistent partitions

The persistent partitions are primarily for the storage of resource configuration, such as data sources, JMS destinations etc. 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 runtime.

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 runtime.

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 13-2 on page 775 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.

13.2.5 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 a structure as shown in Example 13-1.

(top) represents the root of the federated name space.

Example 13-1 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: ITSOCell
  5 (top)/legacyRoot                            javax.naming.Context
  5   Linked to context: ITSOCell/persistent
  6 (top)/persistent                           javax.naming.Context
  7 (top)/persistent/cell                      javax.naming.Context
  7   Linked to context: ITSOCell
  8 (top)/cellname                             java.lang.String
  9 (top)/cell                                 javax.naming.Context
  9   Linked to context: ITSOCell
10 (top)/nodes                                javax.naming.Context
11 (top)/nodes/ITSOCellManager                 javax.naming.Context
12 (top)/nodes/ITSOCellManager/domain          javax.naming.Context
12   Linked to context: ITSOCell
13 (top)/nodes/ITSOCellManager/servers         javax.naming.Context
14 (top)/nodes/ITSOCellManager/servers/dmgr    javax.naming.Context
15 (top)/nodes/ITSOCellManager/servers/dmgr/tm javax.naming.Context
16 (top)/nodes/ITSOCellManager/servers/dmgr/tm/default
16
com.ibm.ws.asynchbeans.timer.TimerManagerImpl
17 (top)/nodes/ITSOCellManager/servers/dmgr/com.ibm.isc
17                                     javax.naming.Context
18 (top)/nodes/ITSOCellManager/servers/dmgr/com.ibm.isc/PluginRegistry
18
com.ibm.ws.PluginRegistry
19 (top)/nodes/ITSOCellManager/servers/dmgr/services javax.naming.Context
20 (top)/nodes/ITSOCellManager/servers/dmgr/services/cache
20                                     javax.naming.Context
21 (top)/nodes/ITSOCellManager/servers/dmgr/services/cache/basecache
21
com.ibm.websphere.cache.DistributedObjectCache
22 (top)/nodes/ITSOCellManager/servers/dmgr/services/cache/distributedmap
22
com.ibm.websphere.cache.DistributedObjectCache
23 (top)/nodes/ITSOCellManager/servers/dmgr/ejb      javax.naming.Context
24 (top)/nodes/ITSOCellManager/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 13-2.

Example 13-2 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: ITSOCell	
5 (top)/legacyRoot	javax.naming.Context
5 Linked to context: ITSOCell/persistent	
6 (top)/persistent	javax.naming.Context
7 (top)/persistent/cell	javax.naming.Context
7 Linked to context: ITSOCell	
8 (top)/cellname	java.lang.String
9 (top)/cell	javax.naming.Context
9 Linked to context: ITSOCell	
10 (top)/nodes	javax.naming.Context
11 (top)/nodes/ITSOCellManager	javax.naming.Context
12 (top)/nodes/ITSOCellManager/domain	javax.naming.Context
12 Linked to context: ITSOCell	
13 (top)/nodes/ITSOCellManager/servers	javax.naming.Context
14 (top)/nodes/ITSOCellManager/servers/dmgr	javax.naming.Context
14 Linked to URL: corbaloc::wan:9809/NameServiceServerRoot	
15 (top)/nodes/ITSOCellManager/node	javax.naming.Context
15 Linked to context: ITSOCell/nodes/ITSOCellManager	
16 (top)/nodes/ITSOCellManager/nodename	java.lang.String
17 (top)/nodes/ITSOCellManager/cell	javax.naming.Context
17 Linked to context: ITSOCell	
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: ITSOCell	

```

22 (top)/nodes/NodeA/domain           javax.naming.Context
22   Linked to context: ITSOCell
23 (top)/nodes/NodeA/nodeAgent        javax.naming.Context
23   Linked to context: ITSOCell/nodes/NodeA/servers/nodeagent
24 (top)/nodes/NodeA/node            javax.naming.Context
24   Linked to context: ITSOCell/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: ITSOCell
30 (top)/nodes/NodeA/servers/nodeagent/servername java.lang.String
31 (top)/nodes/NodeA/servers/nodeagent/thisNode   javax.naming.Context
31   Linked to context: ITSOCell/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 a structure as shown in Example 13-3 on page 782.

Example 13-3 Managed process-level name space

```
1 (top)                                               javax.naming.Context
2 (top)/domain                                         javax.naming.Context
2   Linked to context: ITSC0Cell
3 (top)/cellname                                       java.lang.String
4 (top)/nodes                                           javax.naming.Context
5 (top)/nodes/ITSC0CellManager                         javax.naming.Context
6 (top)/nodes/ITSC0CellManager/servers                 javax.naming.Context
7 (top)/nodes/ITSC0CellManager/servers/dmgr           javax.naming.Context
7   Linked to URL: corbaloc::wan:9809/NameServiceServerRoot
8 (top)/nodes/ITSC0CellManager/domain                  javax.naming.Context
8   Linked to context: ITSC0Cell
9 (top)/nodes/ITSC0CellManager/cell                   javax.naming.Context
9   Linked to context: ITSC0Cell
10 (top)/nodes/ITSC0CellManager/nodename              java.lang.String
11 (top)/nodes/ITSC0CellManager/node                  javax.naming.Context
11   Linked to context: ITSC0Cell/nodes/ITSC0CellManager
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: ITSC0Cell
16 (top)/nodes/NodeA/domain                           javax.naming.Context
16   Linked to context: ITSC0Cell
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: ITSC0Cell/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: ITSC0Cell/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
30 (top)/nodes/NodeA/servers/server1/com/ibm/websphere/ejbquery
30
```

```

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.asyncbeans.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.plantsbywebsphreejb.LoginHome
45 (top)/nodes/NodeA/servers/server1/plantsby/MailerHome
45                                         com.ibm.websphere.samples.plantsbywebsphreejb.MailerHome
46 (top)/nodes/NodeA/servers/server1/plantsby/BackOrderHome
46                                         com.ibm.websphere.samples.plantsbywebsphreejb.BackOrderHome
47 (top)/nodes/NodeA/servers/server1/plantsby/SuppliersHome
47                                         com.ibm.websphere.samples.plantsbywebsphreejb.SuppliersHome
48 (top)/nodes/NodeA/servers/server1/plantsby/ResetDBHome
48                                         com.ibm.websphere.samples.plantsbywebsphreejb.ResetDBHome
49 (top)/nodes/NodeA/servers/server1/plantsby/ReportGeneratorHome
49                                         com.ibm.websphere.samples.plantsbywebsphreejb.ReportGeneratorHome
50 (top)/nodes/NodeA/servers/server1/plantsby/CatalogHome
50                                         com.ibm.websphere.samples.plantsbywebsphreejb.CatalogHome
51 (top)/nodes/NodeA/servers/server1/plantsby/ShoppingCartHome
51                                         com.ibm.websphere.samples.plantsbywebsphreejb.ShoppingCartHome
52 (top)/nodes/NodeA/servers/server1/plantsby/SupplierHome
52                                         com.ibm.websphere.samples.plantsbywebsphreejb.SupplierHome
53 (top)/nodes/NodeA/servers/server1/plantsby/BackOrderStockHome
53                                         com.ibm.websphere.samples.plantsbywebsphreejb.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           java.lang.Object
57                                                               javax.naming.Context
58 (top)/nodes/NodeA/servers/server1/cell                           javax.naming.Context
58   Linked to context: ITS0Cell
59 (top)/nodes/NodeA/servers/server1/wm                           javax.naming.Context
60 (top)/nodes/NodeA/servers/server1/wm/default                   javax.naming.Context
com.ibm.websphere.asyncbeans.WorkManager
61 (top)/nodes/NodeA/servers/MyClusterServer1          javax.naming.Context
61   Linked to URL: corbaloc::wan:9811/NameServiceServerRoot
62 (top)/nodes/NodeA/servers/MyClusterServer2          javax.naming.Context
62   Linked to URL: corbaloc::wan:9812/NameServiceServerRoot
63 (top)/nodes/NodeA/servers/nodeagent                javax.naming.Context
63   Linked to URL: corbaloc::wan:2809/NameServiceServerRoot
64 (top)/clusters                                         javax.naming.Context
65 (top)/clusters/MyCluster                            javax.naming.Context
65   Linked to URL: corbaloc::wan:9811,:wan:9812/NameServiceServerRoot
66 (top)/legacyRoot                                     javax.naming.Context
66   Linked to context: ITS0Cell/persistent
67 (top)/persistent                                      javax.naming.Context
68 (top)/persistent/cell                                javax.naming.Context
68   Linked to context: ITS0Cell
69 (top)/cell                                           javax.naming.Context
69   Linked to context: ITS0Cell
70 (top)/deploymentManager                         javax.naming.Context
70   Linked to URL: corbaloc::wan:9809/NameServiceServerRoot
71 (top)/cells                                         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, etc.) read from the resources.xml of the process, as well as those registered at runtime 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.

13.3 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.

13.3.1 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.

13.3.2 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 13-1 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	Define the type of root to access. See Table 13-5 on page 795 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 13-2 corbaname options

Setting	Description
protocol	Use this protocol used 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.

Setting	Description
key	Define the type of root to access. See Table 13-5 on page 795 for details.
<INS string-formatted-n ame>	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, etc. server, is required.

- ▶ Fully qualified name access

```
corbaname::myhost:9333#cell1/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").

13.4 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.

13.5 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 13.11.1, “Name space bindings” on page 811.)

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.

13.5.1 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.

13.5.2 Types of binding references

There are several different references that can be specified for configured bindings. Valid types are summarized in Table 13-3.

Table 13-3 Types of binding reference

Binding type	Required Settings
EJB (EjbNameSpaceBinding)	<ol style="list-style-type: none"> 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)	<ol style="list-style-type: none"> 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)	<ol style="list-style-type: none"> 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)	<ol style="list-style-type: none"> 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

13.6 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 legacy 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 13-4.

Example 13-4 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 13-5 on page 792.

Example 13-5 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 13-6 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 13-6 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 legacy 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 13-7 shows an IIOP type URL as the provider URL.

Example 13-7 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);
```

13.6.1 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 13-4 summarizes the different categories of clients and the corresponding default initial context.

Table 13-4 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 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 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 13-5 lists the different root contexts and their corresponding object key.

Table 13-5 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 13-8 shows the use of a corbaloc URL with the object key set to select the cell persistent root context as the initial context.

Example 13-8 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 13-6 Name space root values

Root context	CORBA object URL object key
Server root	bootstrapserverroot
Cell persistent root	cellpersistroot
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 13-9 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 13-9 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);
```

13.7 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 13-3, 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 13-3, 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.

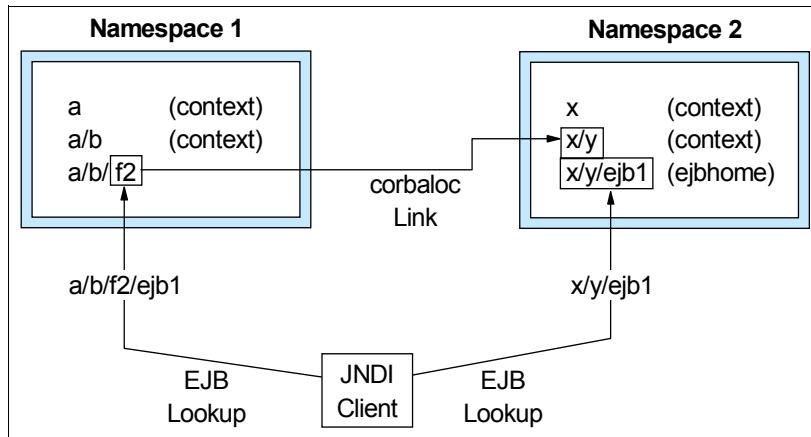


Figure 13-3 JNDI access using federated name spaces

13.8 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

13.8.1 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 as 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 as 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.

13.8.2 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.

13.8.3 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 following corbaname format to look up the EJB home:

Example 13-10 corbaname format for EJB home lookup

```
initialContext.lookup("corbaname:iiop:myHost:2809#cell/clusters/myCluster/myEJB");

```

According to the URL in Example 13-10, 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 13-11.

Example 13-11 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 13-12.

Example 13-12 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.

13.9 Examples

The following examples highlight a number of different server topologies and the affect 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

13.9.1 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 13-4.

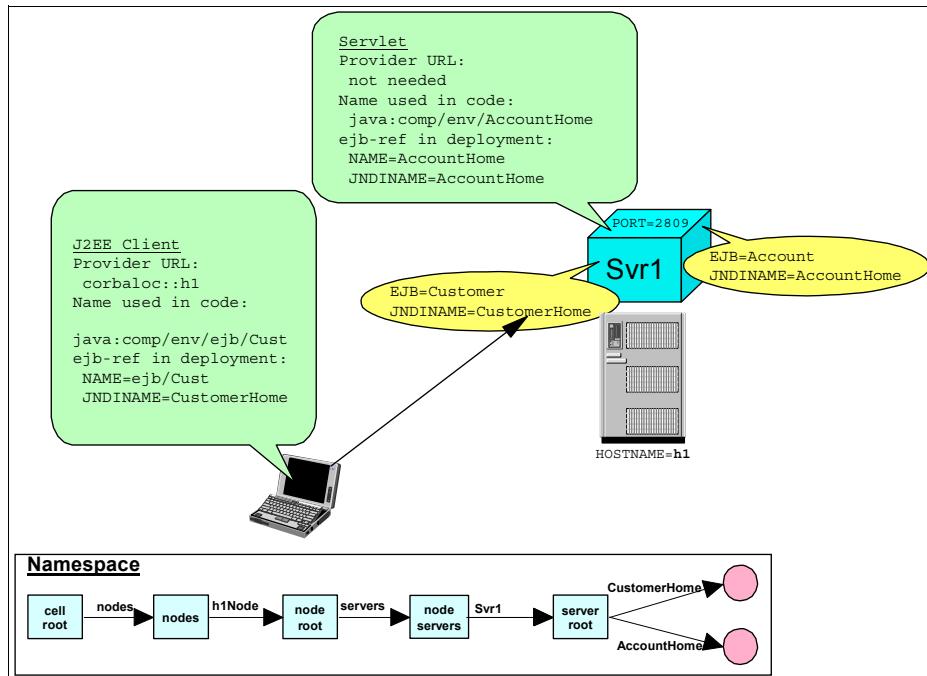


Figure 13-4 Single server

By accessing the server root directly, the J2EE or 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 13-7 illustrates the Provider URL.

Table 13-7 Lookup settings required for a single server

Component	Provider URL	JNDI name
Servlet (same process)	<i>Not needed</i>	CustomerHome

Component	Provider URL	JNDI name
J2EE client (external process)	corbaloc::<hostname>	CustomerHome

13.9.2 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 13-5.

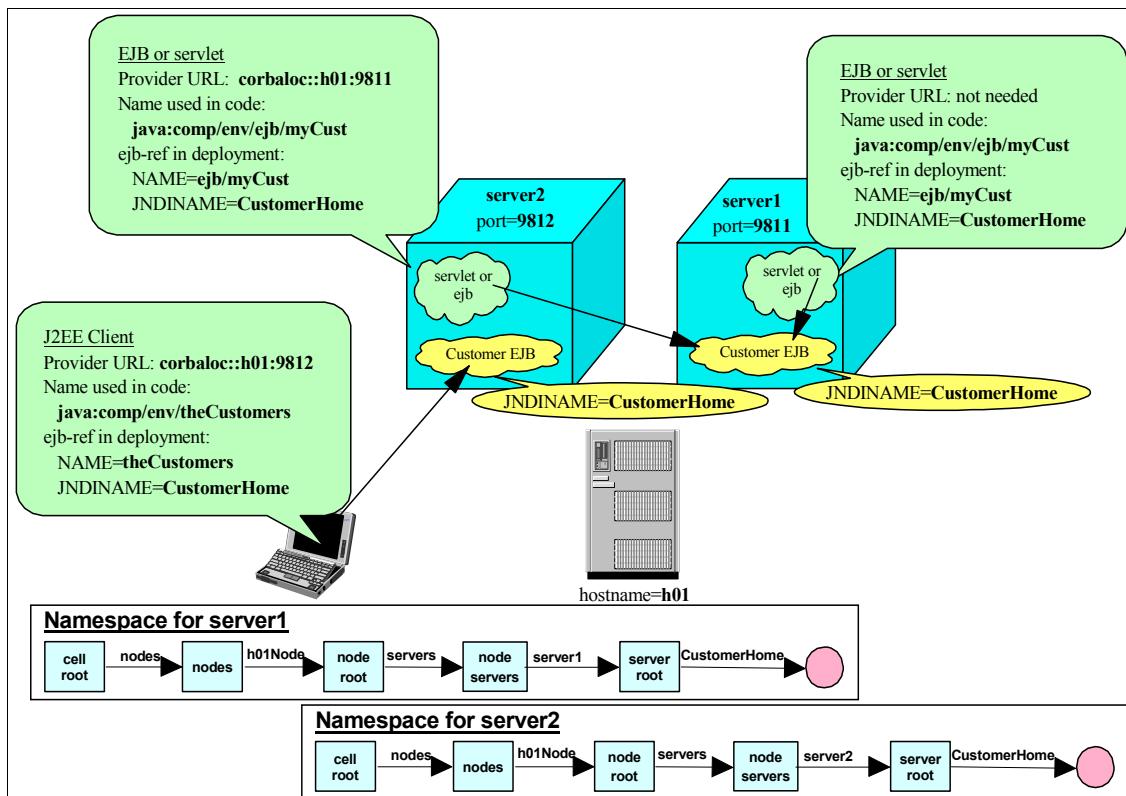


Figure 13-5 Two single servers on the same box

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 13-8 illustrates the required Provider URL settings.

Table 13-8 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

13.9.3 Network Deployment application servers on the same box

The configuration becomes more complex when we move from an standalone 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 13-6 on page 805.

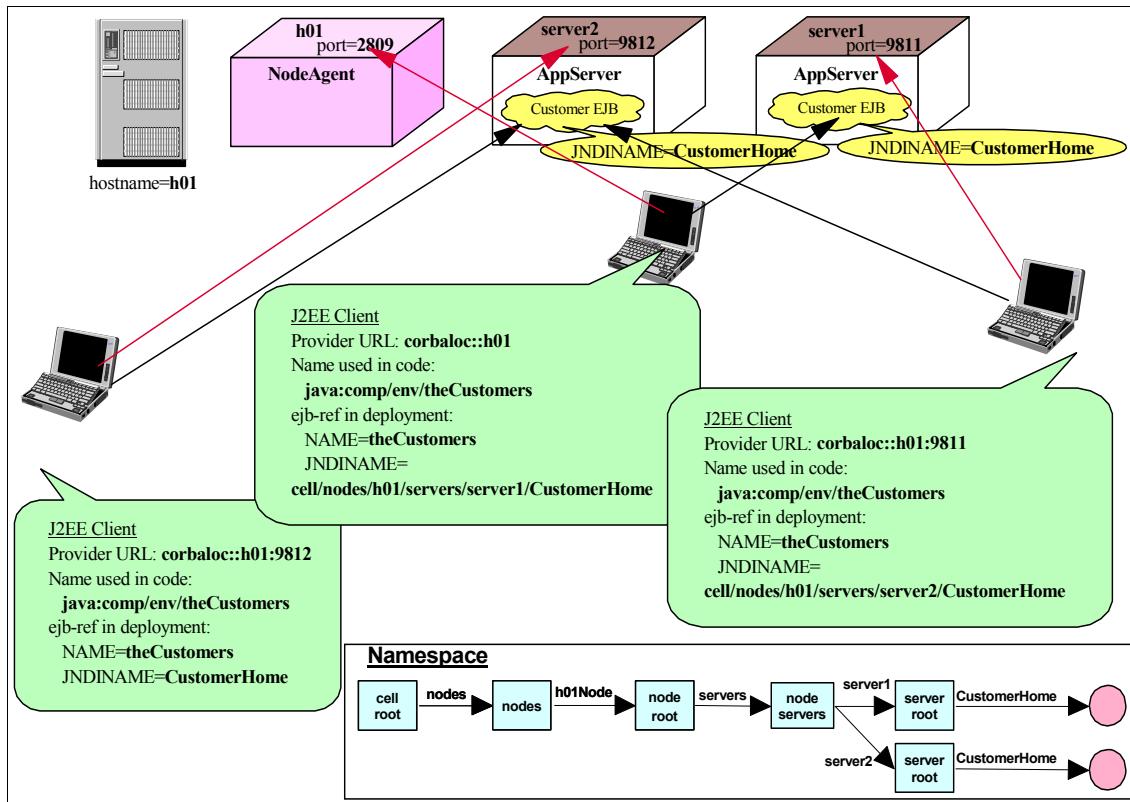


Figure 13-6 Two network deployment application servers on the same box

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 which 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 which has a stable bootstrap address, such as a designated cluster, server, or node agent.

Table 13-9 illustrates the Provider URL settings required.

Table 13-9 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
	(or) <i>Not needed</i>	cell/persistent/CustomerHome2 ¹
J2EE client (external process accessing appserver1 with object located on appserver1)	corbaloc::<appserver hostname>:<port#>	CustomerHome
	(or) <i>Not needed</i>	cell/persistent/CustomerHome1 ¹
J2EE client (external process accessing node agent)	corbaloc::<node agent hostname>	cell/nodes/<nodename>/servers/<server2>/CustomerHome
	(or) <i>Not needed</i>	cell/persistent/CustomerHome2 ¹
J2EE client (external process accessing appserver1 with object located on appserver2)	corbaloc::<appserver1 hostname>:<port#>	cell/nodes/<nodename>/servers/<server2>/CustomerHome
	(or) <i>Not needed</i>	cell/persistent/CustomerHome2 ¹

¹ You must manually configure indirect JNDI references to the respective EJB in the cell/persistent name space.

13.9.4 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 13-7.

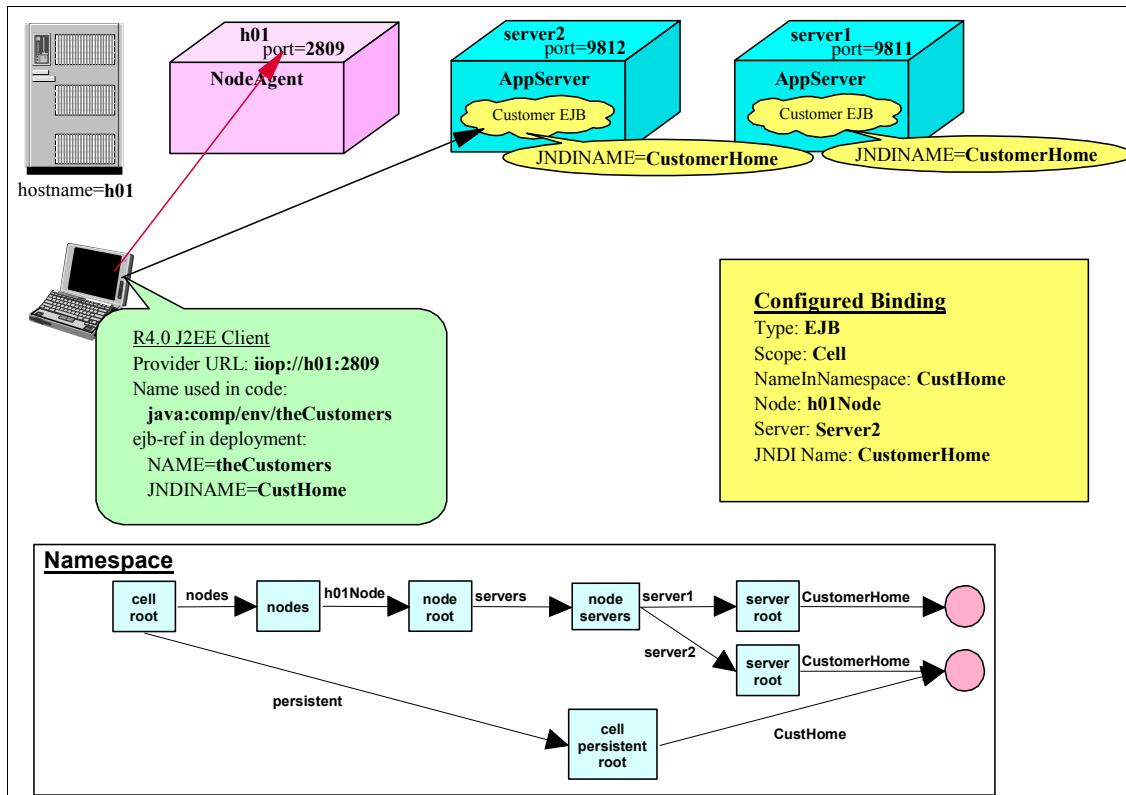


Figure 13-7 WebSphere Application Server V4 client

Table 13-10 illustrates the Provider URL settings required.

Table 13-10 Lookup settings for WebSphere Application Server V4 client interoperability

Component	Provider URL	JNDI name
V4 client	<code>iiop://<hostname>:2809</code>	<code>CustHome</code>

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 runtime *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.

13.10 Naming tools

IBM WebSphere Application Server provides the following tools for the support of the naming architecture.

13.10.1 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 13-11 on page 809 shows the available options.

Table 13-11 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 ok for most use.
-root [cell server node host legacy tree default]	<p>WebSphere V5.0 or later</p> <ul style="list-style-type: none"> ▶ 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") <p>WebSphere V4.0</p> <ul style="list-style-type: none"> ▶ 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. <p>All WebSphere and other name servers</p> <ul style="list-style-type: none"> ▶ default: Dump the tree starting at the initial context which 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>	<ul style="list-style-type: none"> ▶ jndi: Display name components as atomic strings. ▶ ins: Display name components parsed against INS rules (id.kind). <p>The default format is jndi.</p>

Option	Description
-report <length>	<ul style="list-style-type: none"> ▶ 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. <p>The default report option is short.</p>
-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 on 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 13-13.

Example 13-13 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:iop:yourhost:901
```

13.11 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.

13.11.1 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 13-7. 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.

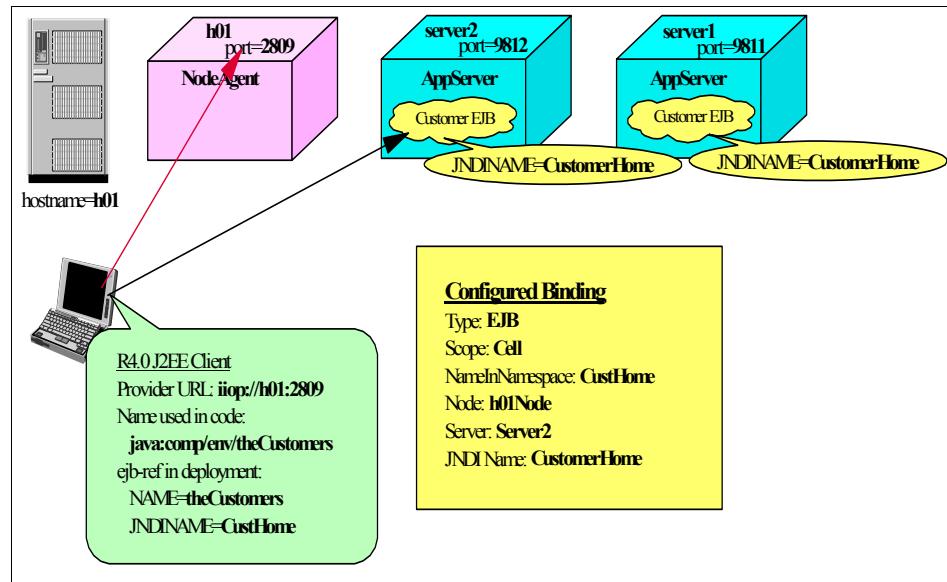


Figure 13-8 WebSphere Application Server V4 client

To create the binding, do the following:

1. Select **Environment** → **Naming** → **Name Space Bindings**.
2. Set the scope to **cell1**.
3. Click **New**. See Figure 13-9 on page 813.

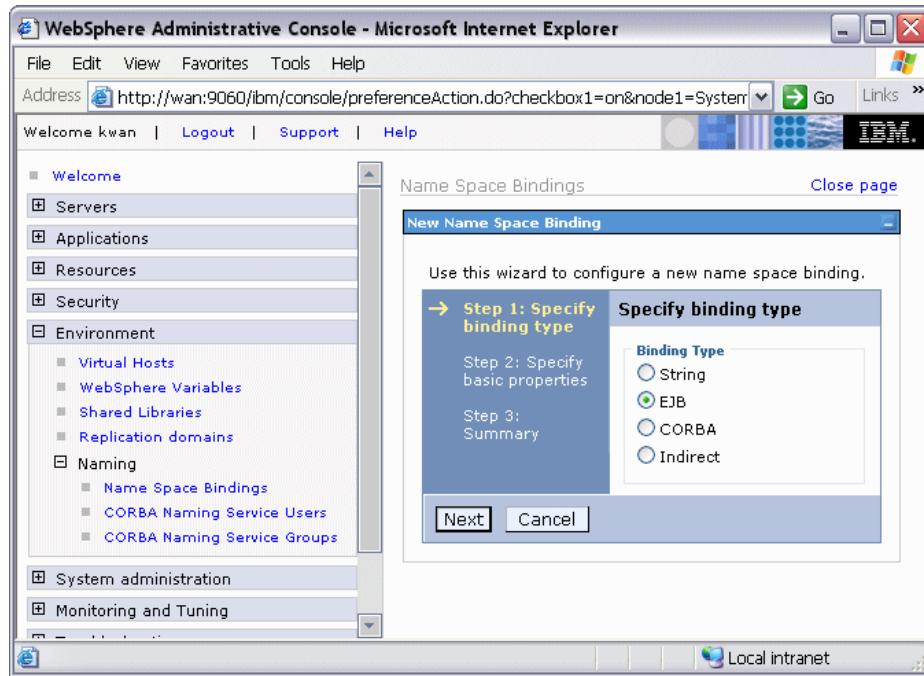


Figure 13-9 Name space binding

4. Choose **EJB** and click **Next**.
5. Enter the values as shown in Figure 13-10 on page 814.
 - **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 the JNDI name of the deployed EJB. Use the name in the enterprise beans bindings, not the java:comp name.

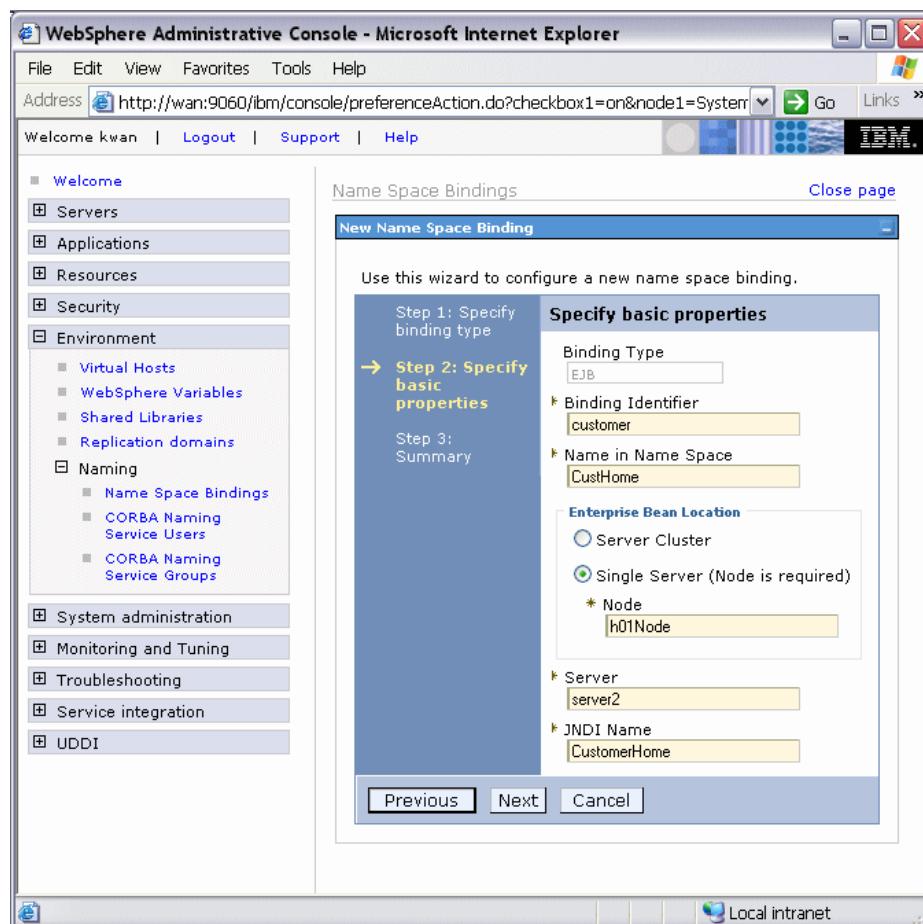


Figure 13-10 Defining an EJB name space binding

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 runtime name space of all application servers in that cell.
- Bindings configured at the node scope area included in the local runtime name space of all application servers in that node.
- Bindings configured at the server scope are included in the local runtime name space of only that application server.

13.11.2 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 affect 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 global security is enabled. Before enabling global security, you should design your entire security solution. See *WebSphere Application Security V6 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 13-11.

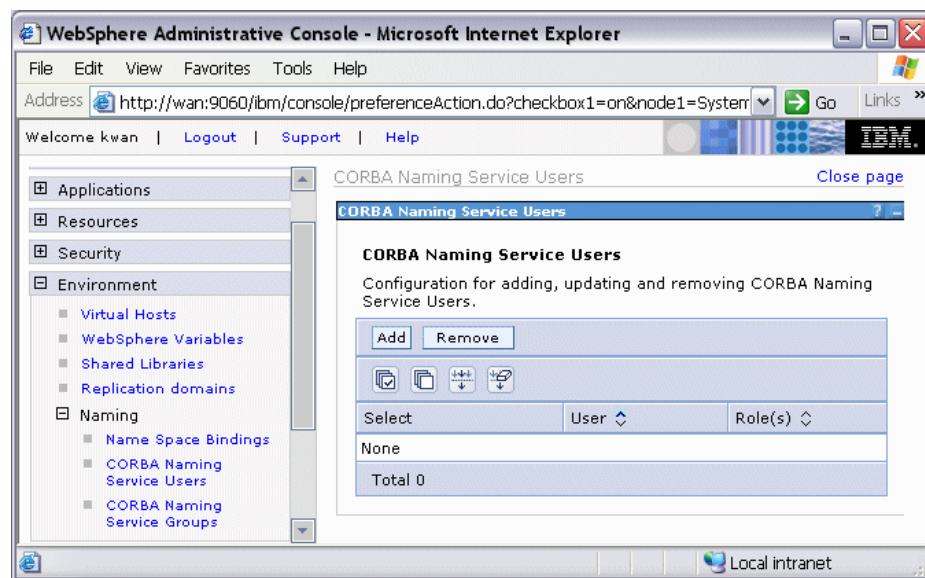


Figure 13-11 Add CORBA naming service users

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 global security, the local operating system user registry will be used.

Note: Before these settings take effect, you will have to enable and configure WebSphere global security.

To specify multiple roles, hold the Ctrl key while you click the applicable roles. See Figure 13-12.

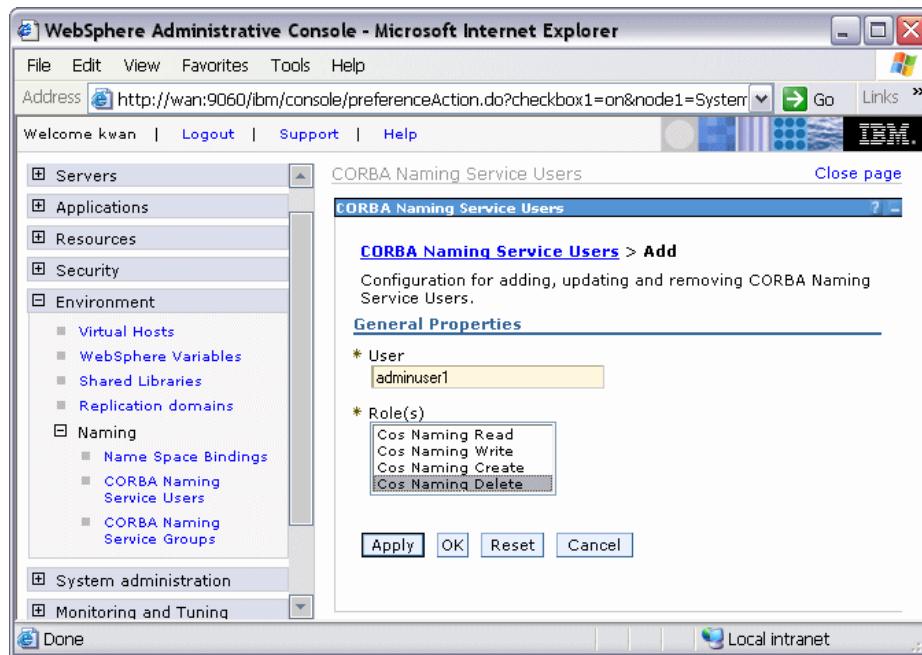


Figure 13-12 Assign an authorization level

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 13-13 on page 818 shows the initial settings.

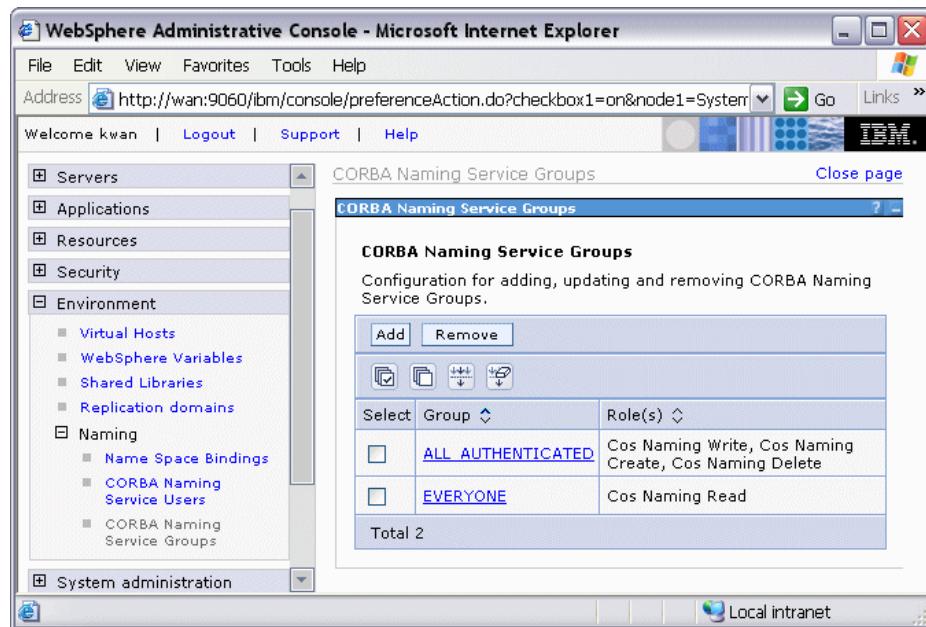


Figure 13-13 Default settings for CORBA naming service groups

Note: The two special groups are already defined and have roles assigned. To change the roles assigned to the two special groups, click the group name link).

2. To add a new group, click **Add**. See Figure 13-14 on page 819.

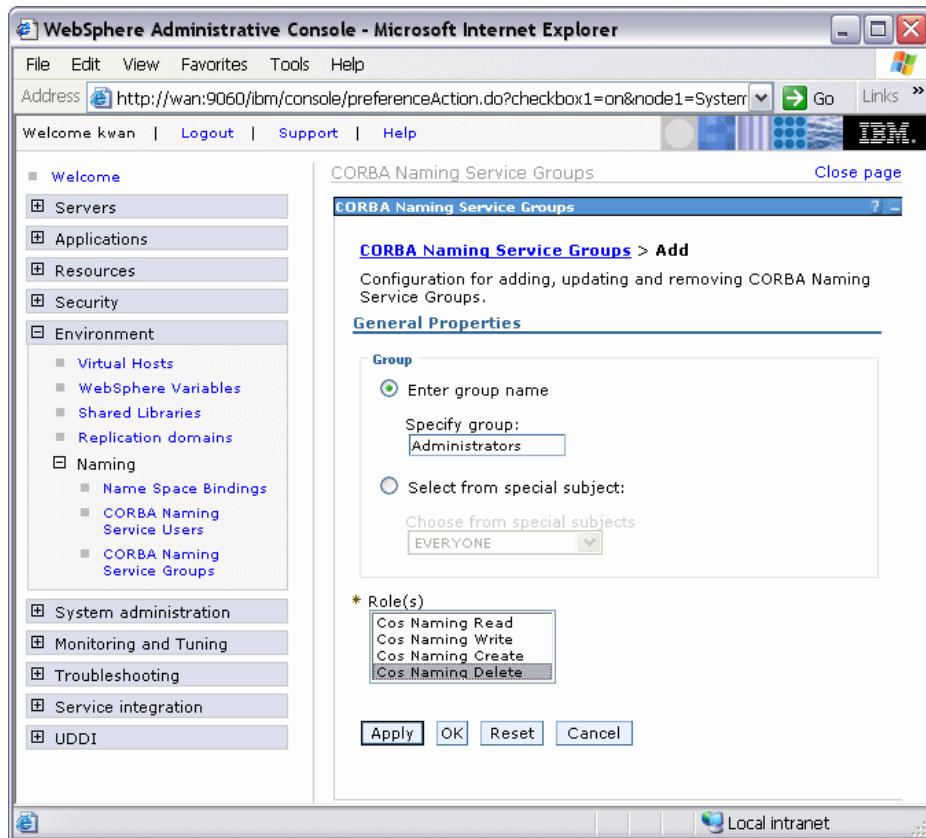


Figure 13-14 Assign an authorization level

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 global security, the local operating system user registry will be used. Remember, before these settings take effect, you will have to enable and configure WebSphere global 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 J2EE packaging. Failure to set up the class loaders properly most likely results in a cascade of the infamous class loading exceptions like `ClassNotFoundException` when trying to start your application.

This chapter starts by giving a little background on Java class loaders. Then, we describe the different WebSphere class loaders and finally how you can customize the behavior of the WebSphere class loaders to suit your particular application's requirements.

The chapter concludes with an example designed to illustrates these concepts.

14.1 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 System class loader.

- ▶ The *bootstrap* class loader is responsible for loading the core Java libraries, that is core.jar, server.jar etc. in the <JAVA_HOME>/lib directory. This class loader, which is part of the core JVM, is written in native code.

Note: Beginning with JDK 1.4 the core, Java libraries in the IBM JDK are no longer packaged in rt.jar, but, instead, are split into multiple JAR files. Previously, they were packaged together and still are for the Sun JDKs.

- ▶ The *extensions* class loader is responsible for loading the code in the extensions directories (<JAVA_HOME>/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 *system* 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.

Delegation is a key concept to understand when dealing with class loaders. It states that a *custom* class loader, a class loader other than the bootstrap, extension or system class loaders, 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.

The Extensions class loader is the parent for the System class loader. The Bootstrap class loader is the parent for the Extensions class loader. The class loaders hierarchy is shown in Figure 14-1 on page 823.

If the System 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.

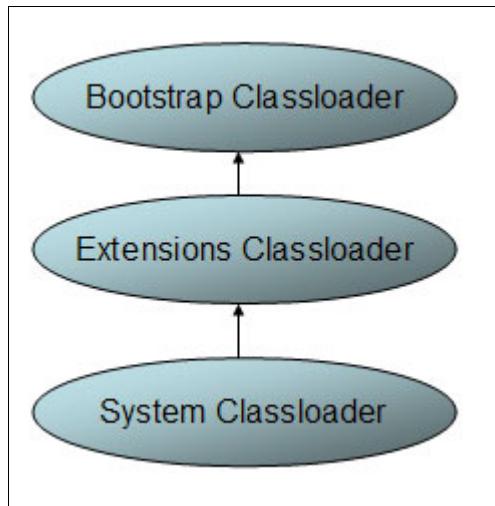


Figure 14-1 Java class loaders hierarchy

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 14-1. A class called WhichClassLoader1 loads a class called WhichClassLoader2, in turn invoking a class called WhichClassLoader3.

Example 14-1 WhichClassLoader1 and WhichClassLoader2 source code

```

public class WhichClassLoader1 {

    public static void main(String[] args)
        throws javax.naming.NamingException
    {
        // Getting Classpathes Value
        StringBuffer bootstrapClassPath =
            new StringBuffer(System.getProperty("sun.boot.class.path"));
        StringBuffer extClassPath =
            new StringBuffer(System.getProperty("java.ext.dirs"));
        StringBuffer systemClassPath =
            new StringBuffer(System.getProperty("java.class.path"));
        // Printing them out
        System.out.println("Bootstrap classpath=\t" +
                           bootstrapClassPath + "\t");
        System.out.println("\nExtension classpath=\t" + extClassPath + "\t");
        System.out.println("\nSystem classpath=\t" + systemClassPath + "\t\n");

        //Loading Classes
        Object obj = new Object();
        WhichClassLoader1 wcl1 = new WhichClassLoader1();
        WhichClassLoader2 wcl2 = new WhichClassLoader2();
    }
}

```

```

//Who loaded what?
System.out.println("->Object was loaded by " +
                     obj.getClass().getClassLoader());
System.out.println(
    "->WCL1 class was loaded by " + wcl1.getClass().getClassLoader());
System.out.println(
    "->WCL2 class 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("->WCL 3 was loaded
by:"+wcl3.getClass().getClassLoader());
    }
}

```

If all WhichClassLoaderX classes are put on the system class path, the three classes are loaded by the System 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>/lib/ext directory. You see the output in Example 14-2.

Example 14-2 NoClassDefFoundError exception trace

```

Bootstrap classpath=
F:\WSADV5\clipse\jre\lib\rt.jar;F:\WSADV5\clipse\jre\lib\i18n.jar;F:\WSADV5\e
clipse\jre\classes
Extension classpath=F:\WSADV5\clipse\jre\lib\ext
System classpath=E:\WSADworkspaces\WebbankV5\ClassloadersTest\bin

->Object was loaded by null
->WCL1 class was loaded by sun.misc.Launcher$AppClassLoader@5059e39d
->WCL2 class was loaded by sun.misc.Launcher$ExtClassLoader@505ca39d
java.lang.NoClassDefFoundError:
com.ibm.wss.lge.classloaders/test/WhichClassLoader3
at
com.ibm.wss.lge.classloaders.test.WhichClassLoader2.getTheClass(WhichClassLoade
r2.java:6)
at
com.ibm.wss.lge.classloaders.test.WhichClassLoader1.main(WhichClassLoader1.java
:27)
Exception in thread "main"

```

As you can see, the program fails with a `NoClassDefFoundError` exception, which might sound strange because `WhichClassLoader3` is on the system class path. The problem is that it is on the wrong class path.

As you can see from the trace, the `WhichClassLoader2` class was loaded by the Extensions class loader. In fact, the System class loader delegated the load of the `WhichClassLoader2` class to the Extensions class loader, which delegated the load 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 and loaded it.

Now, the Extensions class loader needs to load the `WhichClassLoader3` class. It delegates to the Bootstrap class path, which cannot find the class, then tries to load it itself and does not find it either. A `NoClassDefFoundError` exception is thrown. Once a class is loaded by a class loader, any new classes that it tries to load reuse the same class loader, or go up the hierarchy to find a class.

A class loader can only find classes by going up the hierarchy, never down.

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 Extension class loader and the `myApp.properties` file is only seen by the System class loader, the loading of the property file fails.

14.2 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), such as a Network Deployment configuration, this means the class loaders for the JVMs are completely separated even if they are running on the same physical machine.

WebSphere provides several custom delegated class loaders as shown in Figure 14-2 on page 826.

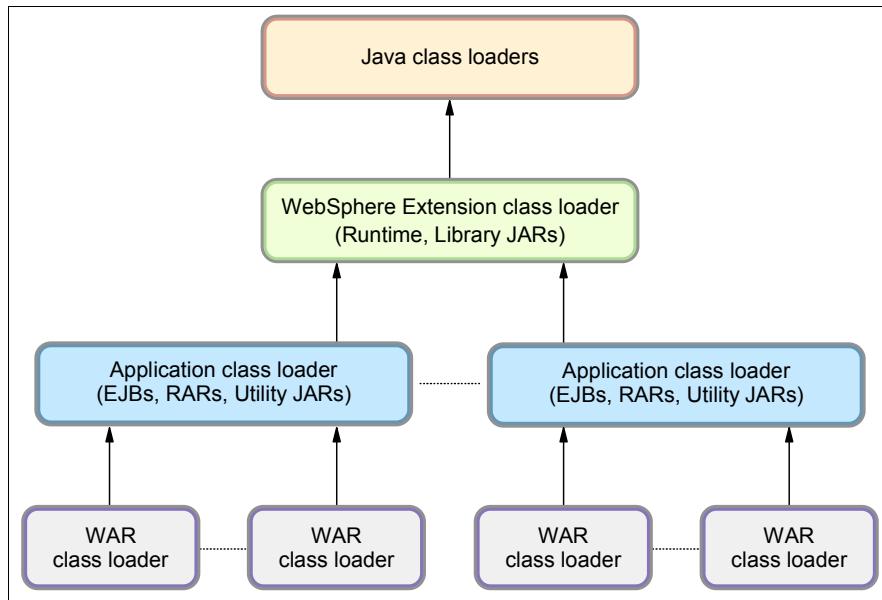


Figure 14-2 WebSphere class loaders hierarchy

The top box in red represents the Java (Bootstrap, Extension and System) class loaders. WebSphere loads just enough here to get itself bootstrapped and initialize the WebSphere extension class loader.

14.2.1 WebSphere extensions class loader

The WebSphere extensions class loader is where WebSphere itself is loaded. It uses the following directories to load the required WebSphere classes:

- ▶ <JAVA_HOME>\lib
- ▶ <WAS_HOME>\classes (Runtime Class Patches directory, or RCP)
- ▶ <WAS_HOME>\lib (Runtime class path directory, or RP)
- ▶ <WAS_HOME>\lib\ext (Runtime Extensions directory, or RE)
- ▶ <WAS_HOME>\installedChannels

The WebSphere runtime is loaded by the WebSphere extensions class loader based on the ws.ext.dirs system property, initially derived from the WS_EXT_DIRS environment variable set in the setupCmdLine script file. The default value of ws.ext.dirs is displayed in Example 14-3 on page 827.

Example 14-3 Default value of ws.ext.dirs

```
SET  
WAS_EXT_DIRS=%JAVA_HOME%\lib;%WAS_HOME%\classes;%WAS_HOME%\lib;%WAS_HOME%\insta  
lledChannels;%WAS_HOME%\lib\ext;%WAS_HOME%\web\help;%ITP_LOC%\plugins\com.ibm.e  
tools.ejbdeploy\runtime
```

The RCP directory is intended to be used for fixes and other APARs that are applied to the application server runtime. These patches override any copies of the same files lower in the RP and RE directories. The RP directory contains the core application server runtime files. The bootstrap class loader first finds classes in the RCP directory then in the RP directory. The RE directory is used for extensions to the core application server runtime.

Each directory listed in the ws.ext.dirs environment variable is added to the WebSphere extensions class loaders class path. In addition, every JAR file and ZIP file in the directory is added to the class path.

You can extend the list of directories and files loaded by the WebSphere extensions class loaders by setting a ws.ext.dirs custom property to the Java virtual machine settings of an application server.

14.2.2 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 adapters 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 application class loader policy, this application 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. Modify the default behavior by changing the application's WAR class loader policy. The default is Module. If the WAR class loader policy is set to Application, 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 Web module 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.

14.2.3 Handling JNI code

Due to a JVM limitation, code that access to native code through a Java Native Interface (JNI) must be placed on a static class path, not on a reloadable class path. This includes shared libraries for which you can define a native class path, or the application server class path. If you have a class loading native code through JNI, this class must not be placed on the WAR or Application class loaders, but rather on the WebSphere extensions class loader.

You can break out just the lines of code that load the native library into a class of its own and place this class on a static class loader. This way you can have all the other code on a reloadable class loader.

14.3 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.

14.3.1 Class loader policies

For each application server in the system, the application class loader policy can be set to Single or Multiple.

When the application 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 Application, the Web module contents for this particular application are also loaded by this single class loader.

When the application class loading 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 Module 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 Module, also the default, the result is as shown in Figure 14-3.

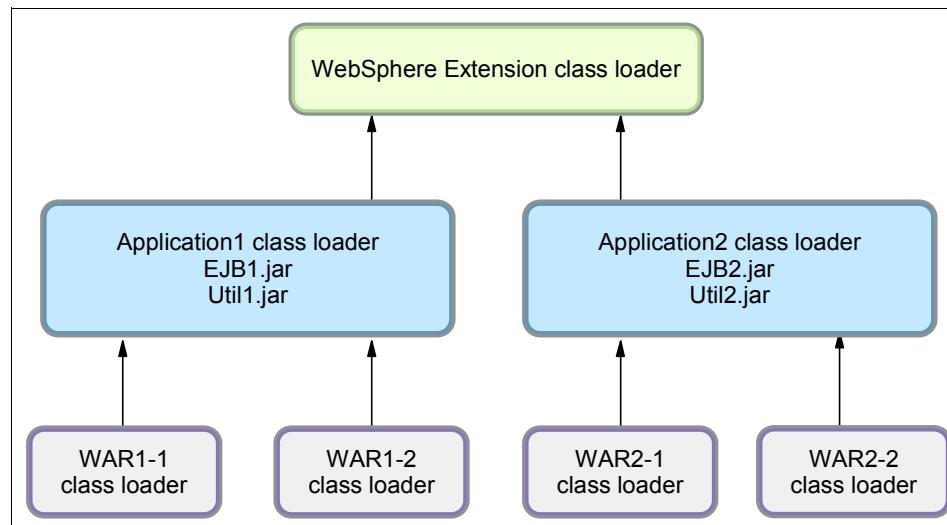


Figure 14-3 Class loader policies: Example 1

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 Module to Application, the result is as shown in Figure 14-4 on page 830.

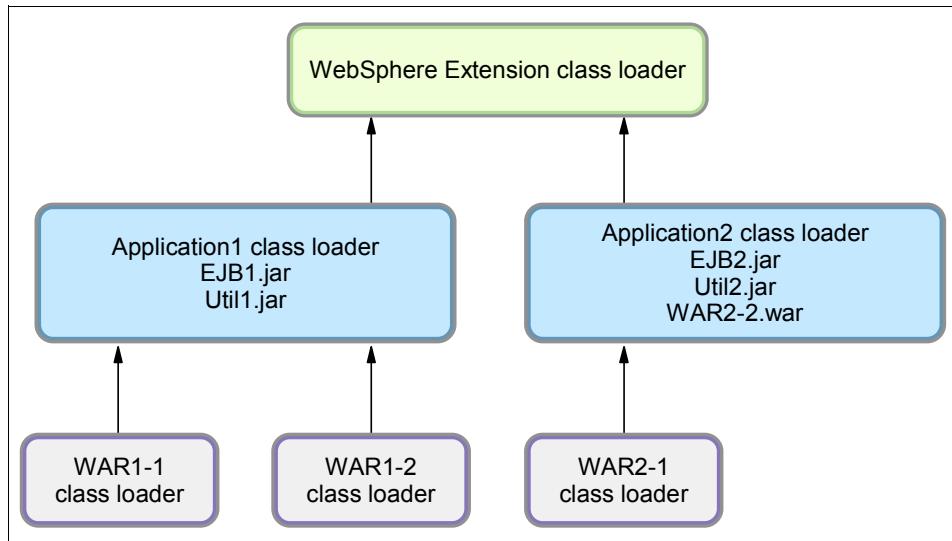


Figure 14-4 Class loader policies: Example 2

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 Module to Application, the result is as shown in Figure 14-5 on page 831.

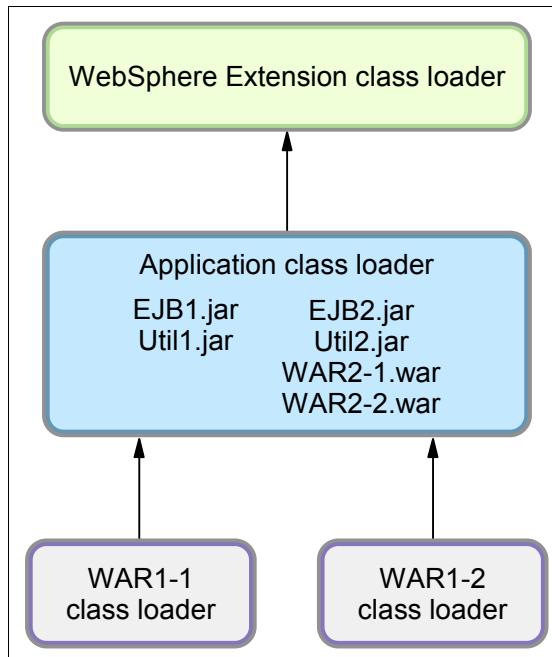


Figure 14-5 Class loader policies: Example 3

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 can still not, 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.

14.3.2 Class loader/delegation mode

WebSphere's application class loader and WAR class loader both have a setting called the class loader mode. This setting determines whether they should follow the normal Java class loader delegation mechanism as described in 14.1, “A brief introduction to Java class loaders” on page 822 or override it. There are two possible values for the class loader mode: PARENT_FIRST and PARENT_LAST.

The default value for class loader mode is 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 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.

Delegation mode can be set for the following class loaders:

- ▶ Application class loader
- ▶ WAR class loader
- ▶ Shared library class loader

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 doing so you would perhaps be surprised to see that all modules, also 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, 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 loader mode for the Web modules to 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.

14.3.3 Class preloading

The first time that a WebSphere Application Server process starts, the name of each run-time class that is loaded and the name of the JAR file that contains the class are written to a preload file. The names of non-runtime classes such as custom services, resource classes such as db2jcc.jar, classes on the JVM class path, and application classes, your own J2EE applications, are not written to the preload file. Subsequent startups of the process use the preload file to start the process more quickly.

Preload files have the .preload extension. WebSphere Application Server processes that have preload files include those listed in Table 14-1.

Table 14-1 WebSphere class preloading file names

Process	Preload file name
Application server	cell_name.node_name.server_name.preload
startServer	WsServerLauncher.preload
launchClient	launchClient.preload

Running the startServer server1 command causes the startServer command to use a WsServerLauncher.preload file and the server to use a cell_name.node_name.server1.preload file. Later, running a command such as startServer server1 -script, where the -script option creates a new script, uses the cell_name.node_name.server1.preload file only.

Preload files, by default, are created in the logs/preload directory of the profile directory.

New classes required during the startup of a process are added to the preload file. Any classes removed from a process are ignored during subsequent startups. Although it is not necessary, an administrator can delete the preload file and force a refresh that removes the ignored classes from the file.

Class preloading is enabled by default. If you want to disable it for a particular application server, set the JVM property `ibm.websphere.preload.classes` to `false` for that application server. To do this, do the following:

1. In the administrative console, click **Servers > Application Servers > server_name > Java and Process Management > Process Definition > Java Virtual Machine**.
2. On the Java Virtual Machine page, specify
`-Dibm.websphere.preload.classes=false` for Generic JVM arguments.
3. Click **OK**.
4. Save your administrative configuration.
5. Stop the application server and then restart it.

To enable class preloading again, either remove the `ibm.websphere.preload.classes` setting or set it to true and then stop and restart the application server.

14.3.4 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 or 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, or server level. However, defining a library at one of the three levels does not cause the library to be loaded. You must associate the library to an application and an application server, or just an application server, for the classes represented by the shared library to be loaded.

You can associate the shared file to an application in one of two ways:

- ▶ You can use the administration tools. The library is added under the Libraries section of the additional properties 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 *installed optional packages* in the Information Center.

Shared files are associated with 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**. At this point, you have two options:

- ▶ If you associate the shared library to an application, the JARs listed on the shared library path are loaded by the application class loader together with EJB JARs, RARs and utility JARs.
- ▶ If you associate the shared library at the application server level, the JARs listed on the shared library path are loaded by a specific class loader, which you define.

See “Step 4: Sharing utility JARs among multiple applications” on page 840 for more details.

14.4 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 JSP and one EJB. Both call a class, VersionChecker, shown in Example 14-4. 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 14-4 VersionChecker class source code

```
public class VersionChecker {  
  
    static final public String classVersion = "1.0";  
  
    public String printClassLoaderInfo() {  
        return ("Class Version Checker was loaded by: " +  
               this.getClass().getClassLoader());  
    }  
  
    public String printVersionInfo () {  
        return ("Class version is: " + classVersion);  
    }  
}
```

Once installed, the application can be invoked through `http://localhost:9080/testclassloaders`. This invokes a JSP (`callVersionChecker.jsp`) which returns the sample information in Example 14-5.

Example 14-5 Invoking callVersionChecker.jsp

```
Class Version Checker was loaded by:  
com.ibm.ws.classloader.CompoundClassLoader@1284e9f8  
Local ClassPath:  
C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\kcfn7r6Cell\ClassloaderTest.ear\ClassloaderTestEJB.jar;C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\kcfn7r6Cell\ClassloaderTest.ear\myjars\ClassloadersTest.jar; Delegation Mode: PARENT_FIRST  
Class version is: 1.0
```

The ClassloadersTest.jar file contains the VersionChecker class file. 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 PARENT_FIRST. We assume the application has been deployed to an application server called AppSrv01. See Example 14-6.

Example 14-6 CallVersionChecker.jsp source

```
...  
<TITLE>WAS V6 Classloaders Info</TITLE>  
</HEAD>  
<BODY>  
<h2> Version Checker Information </h2>  
<h3> Direct call from Servlet to Version Checker Class </h3>  
<%  
    com.ibm.wss.lge.classloaders.VersionChecker versionChecker  
        = new com.ibm.wss.lge.classloaders.VersionChecker();  
    out.println("<br/>" + versionChecker.printClassLoaderInfo());  
    out.println("<br/>" + versionChecker.printVersionInfo());  
%>  
<h3> Call the VersionChecker via Access bean-> enterprise session bean </h3>  
<%  
    com.ibm.wss.lge.ejbs.ClassLoaderTesterAccessBean versionCheckerAB  
        = new com.ibm.wss.lge.ejbs.ClassLoaderTesterAccessBean();  
    out.println("<br />" + versionCheckerAB.callVersionChecker());  
%>  
</BODY>  
...
```

During the tests, we worked directly on the file system by modifying the contents of the `<profile_home>installedApps\kcfn7r6Cell\ClassloaderTest.ear` directory. Obviously, this is not a wise thing to do in production!

14.4.1 Step 1: Simple WAR packaging

Start with the following assumption: our utility class is only used by a servlet. In the JSP, we have commented out the lines that invoke the EJB part of the application. We have placed the ClassloadersTest.jar file under the WEB-INF/lib directory of the WAR file.

Tip: You put under **WEB-INF/lib** JAR files used by a **single** Web module, or a JAR file that *only* this Web module should see.

When we run the application in such a configuration, we obtain the results shown in Example 14-7.

Example 14-7 Class loader: Example 1

Version Checker Information

Direct call from Servlet to Version Checker Class

Class Version Checker was loaded by:

com.ibm.ws.classloader.CompoundClassLoader@31d13f47

Local ClassPath:

C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest2.ear\ClassLoaderTestWeb2.war\WEB-INF\classes;C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest2.ear\ClassLoaderTestWeb2.war\WEB-INF\lib\ClassloadersTest.jar;C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest2.ear\ClassLoaderTestWeb2.war;

Delegation Mode: PARENT_FIRST

Class version is: 1.0

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:

ClassLoaderTestWeb.war\WEB-INF\classes
ClassLoaderTestWeb.war\WEB-INF\lib\ClassloadersTest.jar
ClassLoaderTestWeb.war

The WEB-INF/classes folder is usually used for servlets and property files, while the WEB-INF/lib is used for JARs (not ZIPs). Note that the class loader class path is built at application startup, which is why you see only the ClassloadersTest JAR listed there. If multiple JARs had been placed under this folder, you would see all of them listed on the local class path. The root of the WAR file is the next place where you can put code or properties classes.

14.4.2 Step 2: Sharing the utility JAR among multiple modules

Next, we decided to run the EJB part of the application, which also depends on our ClassloadersTest.jar JAR file. Because the JAR file is to be used by multiple modules in the same application, the best solution is to place it relative to the root of the EAR file and to reference it from a Class-Path entry in the JAR manifest file.

In Example 14-7 on page 837, the VersionChecker stored in in WEB-INF/LIB/ClassloadersTest.jar returned a class version of 1.0 in the output. For illustration purposes, we have created a new JAR file called ClassloadersTestV2.jar. The VersionChecker stored in this JAR file returns version 2.0.

The Manifest Class-Path entry

In this situation, we place the JAR file in a /myjars/ directory at the root of the EAR file. From a J2EE perspective, this directory is unknown, and the directory contents will not be added to the class path automatically. The solution is to create a Class-Path entry in the META-INF/MANIFEST.MF file for each module which uses classes from the classloadersTest.jar file. Paths are relative to the root of the EAR file.

Update the WAR file as in Example 14-8:

Example 14-8 Updating the WAR file

```
#This is the Manifest file for ClassloaderTestWeb file  
Manifest-Version: 1.0  
Class-Path: ClassloaderTestEJB.jar myjars/ClassloadersTestV2.jar
```

In the EJB file, update the MANIFEST.MF as in Example 14-9:

Example 14-9 Updating MANIFEST.MF

```
#This is the Manifest file for ClassloaderTestEJB  
Manifest-Version: 1.0  
Class-Path: myjars/ClassloadersTestV2.jar
```

The test results are then as shown in Example 14-10.

Example 14-10 Class loader: Example 2

Version Checker Information
Direct call from Servlet to Version Checker Class

Class Version Checker was loaded by:
com.ibm.ws.classloader.CompoundClassLoader@3096c631

```
Local ClassPath:  
C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.ear\ClassloaderTestEJB.jar;C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.ear\myjars\ClassloadersTestV2.jar;
```

```
Delegation Mode: PARENT_FIRST  
Class version is: 2.0
```

```
Call the VersionChecker via Access bean to enterprise session bean  
VersionChecker Classloader Info
```

```
Class Version Checker was loaded by:  
com.ibm.ws.classloader.CompoundClassLoader@3096c631 Local ClassPath:  
C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.ear\ClassloaderTestEJB.jar;C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.ear\myjars\ClassloadersTestV2.jar;
```

```
Delegation Mode: PARENT_FIRST  
Class version is: 2.0
```

Although the class loader type is the same (CompoundClassLoader), this is the application class loader, not the WAR class loader. You can see this from the class path, which is very different. Remember, all utility JARs are loaded by the application class loader. As soon as you reference a JAR with a Class-Path directive, it is considered a utility JAR. The *same* class loader is used by the servlet and the EJBs to load the VersionChecker class.

14.4.3 Step 3: Changing the WAR class loader delegation mode

What happened to the ClassloadersTest.jar under WEB-INF/lib? It is still there, but it is ignored because the delegation mode of the WAR is set to PARENT_FIRST.

Set the delegation mode to PARENT_LAST, using the following steps:

1. Select the **Enterprise Applications** entry in the navigation area.
2. Select the **ClassloaderTest** application.
3. Select **Web Modules** under the Related Items section.
4. Select the **ClassloaderTestWeb.war**.
5. Change the Class loader Mode to PARENT_LAST.
6. Click **OK**.
7. Save the configuration.
8. Restart the application.

The VersionChecker in WEB-INF/lib returns a class version of 1.0. You can see in Example 14-11 on page 840 that this is the version used by the WAR file.

Example 14-11 Class loader: Example 3

Version Checker Information

Direct call from Servlet to Version Checker Class

Class Version Checker was loaded by:

com.ibm.ws.classloader.CompoundClassLoader@30a70501

Local ClassPath:

C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.ear\ClassloaderTestWeb.war\WEB-INF\classes;C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.ear\ClassloaderTestWeb.war\WEB-INF\lib\ClassloadersTest.jar;C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.ear\ClassloaderTestWeb.war;

Delegation Mode: PARENT_LAST

Class version is: 1.0

Call the VersionChecker via Access bean to enterprise session bean

VersionChecker Classloader Info

Class Version Checker was loaded by:

com.ibm.ws.classloader.CompoundClassLoader@30c94501

Local ClassPath:

C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.ear\ClassloaderTestEJB.jar;C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.ear\myjars\ClassloadersTestV2.jar;

Delegation Mode: PARENT_FIRST

Class version is: 2.0

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.

14.4.4 Step 4: Sharing utility JARs among multiple applications

In this situation, the ClassloaderTest.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.

You have two solutions to this problem:

- ▶ Use the application extensions class loader, inherited from Version 4.0.
- ▶ Use shared libraries, the recommended solution.

Using the application extensions class loader

The role of this class loader is to load any JAR file placed in the `<was_home>/lib/app` directory. This class loader has a delegation mode set to `PARENT_LAST`.

Note: All applications running within the same WebSphere node running with `PARENT_FIRST` delegation mode see what is contained in this directory, regardless of the application server in which they run. This might be what you want to achieve. You might have a framework that needs to be shared by all your applications. But if some applications want to use the level 1.0 of the framework while others want to use Version 2.0, using the `lib/app` directory is not the right solution.

By default, the `<was_home>/lib/app` directory does not exist. However, at application startup, the class loader runtime checks for its existence and starts the application extensions class loader if it does exist.

For testing purposes only, if you create this `<was_home>/lib/app` directory, drop in the `ClassloadersTest.jar`, and restart the application server, the results are shown in Example 14-12.

Example 14-12 Class loader: Example 4

Version Checker Information

Direct call from Servlet to Version Checker Class

Class Version Checker was loaded by:

`com.ibm.ws.classloader.CompoundClassLoader@3128395a`

Local ClassPath:

`C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.ear\ClassloaderTestWeb.war\WEB-INF\classes;C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.ear\ClassloaderTestWeb.war\WEB-INF\1ib\ClassloadersTest.jar;C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.ear\ClassloaderTestWeb.war;`

Delegation Mode: PARENT_LAST

Class version is: 1.0

**Call the VersionChecker via Access bean to enterprise session bean
VersionChecker Classloader Info**

```
Class Version Checker was loaded by:  
com.ibm.ws.classloader.ExtJarClassLoader@1483061593 Local ClassPath:  
C:\WebSphere\AppServer\lib\app\ClassloadersTest.jar;
```

```
Delegation Mode: PARENT_LAST  
Class version is: 1.0
```

Note that the servlet still uses the code from WEB-INF/lib because its delegation mode is set to PARENT_LAST. The EJB, however, uses the code from the lib/app directory.

Using shared libraries

A much cleaner and manageable way to work is to use the shared libraries support. Shared libraries can be defined at the cell, node, and application server levels. Once you have defined a shared library, you must associate it to an application or to a server. JARs and folders listed on a shared library are always loaded by the application class loader.

You can define as many shared libraries as you want. You can also associate multiple shared libraries with an application 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 level at which you want this shared library to be defined, such as Cell, and click **New**.
3. Specify the following properties as in Figure 14-6 on page 843.

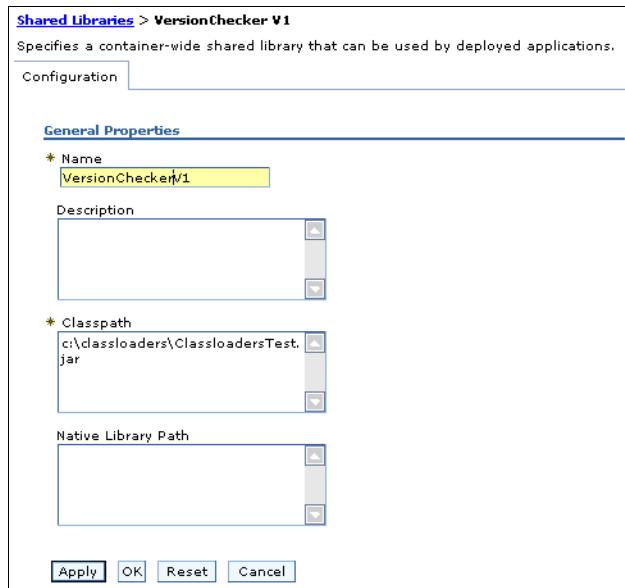


Figure 14-6 Shared library configuration

- **Name:** Enter VersionCheckerV1.
 - **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%/ClassloadersTest.jar. Make sure that you declare this variable at the same scope as the shared library for cell, node, or server).
 - **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 **ClassloadersTest** application.
 7. In Additional Properties, select **Libraries**.
 8. Click **Add**. You should see the library you just created. If not, you probably have a scoping problem.
 9. Select the shared library, and click **OK**.
 10. Save the configuration.

If we now remove the ClassloadersTest.jar file from the lib/app folder and restart the application server, we see the following test results in Example 14-13:

Example 14-13 Class loader: Example 5

**Call the VersionChecker via Access bean to enterprise session bean
VersionChecker Classloader Info**

Class Version Checker was loaded by:
com.ibm.ws.classloader.CompoundClassLoader@30ec422c Local ClassPath:
C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.ear\ClassloaderTestEJB.jar;C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.ear\myjars\ClassloadersTestV2.jar;C:\classloaders\ClassloadersTest.jar;

Delegation Mode: PARENT_FIRST
Class version is: 2.0

Although we have put the V1.0 of the ClassloadersTest.jar on the shared library, we see V2.0 being loaded. This is because we are loading the ClassloadersTestV2.jar, which is inside the EAR file. We have not removed it! The list of JAR files you declare in a shared library is appended to the application class loader class path. Keep in mind that order is very important.

Therefore, if you want the shared library code to be loaded, remove the Class-path entry from the EJB module MANIFEST.MF file, restart the application, and try again.

Using shared libraries at the application server level

A shared library can also be associated to an application server. All applications deployed in this server see the code listed on that shared library. To associate a shared library to an application server, you must first create a class loader 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 loading policy for this class loader, **PARENT_FIRST** or **PARENT_LAST**. If you set the application policy to **PARENT_LAST**, you will be able to use your own classes in place of some WebSphere runtime classes , for example typically, XML parsers or transformers.
4. Click **Apply**.
5. Select the **Libraries** entry.

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.
7. Click **OK**.
8. Save the configuration.

If we attach the VersionCheckerV1 shared library to the application server, we obtain the results in Example 14-14 with our application:

Example 14-14 Class loader: Example 6

Version Checker Information
Direct call from Servlet to Version Checker Class

Class Version Checker was loaded by:
com.ibm.ws.classloader.CompoundClassLoader@3118cdc9

Local ClassPath:
C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.ear\ClassloaderTestWeb.war\WEB-INF\classes;C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.ear\ClassloaderTestWeb.war\WEB-INF\1ib\ClassloadersTest.jar;C:\WebSphere\AppServer\profiles\AppSrv01\installedApps\Cell01\ClassloadersTest.ear\ClassloaderTestWeb.war;

Delegation Mode: PARENT_LAST
Class version is: 1.0

Call the VersionChecker via Access bean to enterprise session bean
VersionChecker Classloader Info

Class Version Checker was loaded by:
com.ibm.ws.classloader.ExtJarClassLoader@1204276682

Local ClassPath: C:\classloaders\ClassloadersTest.jar;

Delegation Mode: PARENT_FIRST
Class version is: 1.0

Note that the WAR class loader continues to load its own version due to the delegation mode, PARENT_LAST, while the EJB module loads the code from the highest class loader in the hierarchy, the one we just defined at the application server level.



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 WebSphere Bank sample application that ships with WebSphere Application Server V6. We will show you how to:

- ▶ Set the JNDI bindings for its EJBs
- ▶ Change the sample to use a DB2 database
- ▶ Generate deployment code for the application

We will also discuss some advanced configuration options such as EJB caching and access intents.

Finally, we explain the concept of the Enhanced EAR file, introduced in WebSphere Application Server V6, and show how to add more deployment information to the WebSphere Bank EAR file.

To package the application, we will use the Application Server Toolkit. The toolkit is based on the Eclipse V3 platform and contains a subset of the plug-ins that make up Rational Application Developer V6. It allows you to create and modify J2EE applications and modules, edit deployment descriptors, and map databases. The toolkit also allows you to perform basic testing, debugging and profiling of WebSphere applications.

15.1 WebSphere Bank sample application

WebSphere Bank is an Internet bank that provides online checking and savings banking. Using WebSphere Bank, customers can open accounts, get account balances, and transfer funds between accounts. It has both a Web front-end and a simple text-mode interface accessible by running its J2EE application clients.

The application uses many of the J2EE and WebSphere Application Server functions such as EJBs (session, entity and message-driven beans), servlets, JSPs, Web services, JMS, and so forth.

We will not go into details on how WebSphere Bank 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 WebSphere Bank is installed and configured properly, it can be invoked at:

<http://localhost:9080/WebSphereBank>

The WebSphereBank EAR file consists of the modules shown in Figure 15-1.

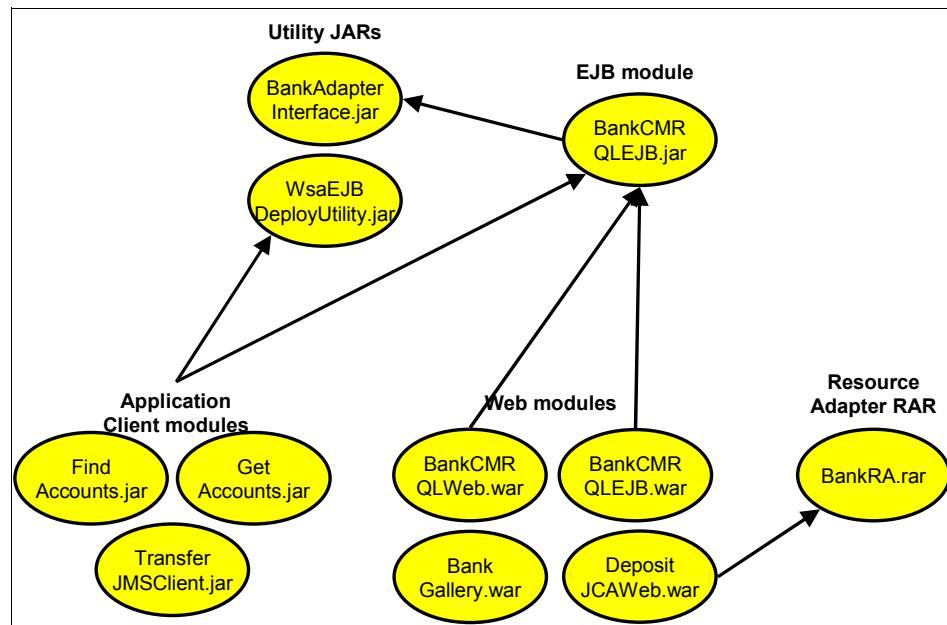


Figure 15-1 WebSphereBank modules

As shown, the WebSphere Bank EAR file has two utility JARs, one EJB module, four Web modules and one resource adapter module. It also has three application client modules which are used to invoke services on the system from a standalone JVM.

15.1.1 WebSphere Bank resources used

To run successfully, WebSphere Bank requires the following resources to be configured:

- ▶ JAAS authentication alias
 - Scope: cell level
 - Name: <cellname>/samples
 - Userid: samples
 - Password: s1amples
- ▶ JDBC provider
 - Scope: server level
 - Name: Samples Cloudscape JDBC Provider (XA)
 - Driver: Built-in Cloudscape JDBC Provider (XA)
- ▶ Data source
 - Scope: server level
 - JDBC provider: Samples Cloudscape JDBC Provider (XA)
 - Name: BANKDS
 - JNDI name: jdbc/Bank
 - Database name:
 \${APP_INSTALL_ROOT}/\${CELL}/WebSphereBank.ear/Database/BankDB
- ▶ Data source connection factory
 - Scope: server level
 - JDBC provider: Samples Cloudscape JDBC Provider (XA)
 - Datasource name: BANKDS
 - Name: BANKDS_CF
 - Authentication mechanism: BASIC_PASSWORD
 - Authentication alias: N_O_N_E
 - Interface: javax.resource.cci.ConnectionFactory
- ▶ Service integration bus
 - Scope: node level
 - Name: <nodename>SamplesBus
- ▶ JMS connection factory
 - Scope: node level
 - Name: BankJMSConnFactory

- JNDI name: jms/BankJMSConnectionFactory
- Authentication alias: <cell_name>/samples
- Service Integration Bus name: <node_name>SamplesBus
- ▶ JMS queue
 - Scope: node level
 - Name: BankJMSQueue
 - JNDI name: jms/BankJMSQueue
 - Service Integration Bus queue: BankJSQueue
- ▶ Service integration bus queue
 - Scope: server level
 - Name: BankJSQueue
 - Service Integration Bus: <node_name>SamplesBus
- ▶ JMS activation spec
 - Scope: node level
 - Name: BankActivationSpec
 - JNDI name: eis/BankActivationSpec
 - Destination JNDI name: jms/BankJMSQueue
 - Destination type: javax.jms.Queue
 - Authentication alias: <cell_name>/samples
 - Service integration bus: <node_name>SamplesBus

The resources can be configured automatically by running the WebSphere Bank install script, which uses wsadmin. When you first access the WebSphere samples, you will see two options in the navigation panel on the left: Installed Samples and Installable Samples. Expand **Installable Samples** and select **WebSphere Bank** under Applications. This will show you information about how to run the script.

You can also define the environment manually by inspecting the installation scripts and using the administrative console to define the resources.

15.2 Packaging using the Application Server Toolkit

To illustrate packaging techniques, we will import the WebSphere Bank application into the Application Server Toolkit and go through the various aspects of packaging an application for deployment. The application can then be exported as an EAR file for deployment.

15.2.1 Preparing the sample code

In our example, we used the WebSphere Bank sample application shipped with WebSphere Application Server V6. To obtain sample code suitable for import to the Application Server Toolkit, we used the documentation available with the Samples Gallery to rebuild the sample code using ANT. The steps required to prepare the code for import on a Windows system are as follows:

1. Create an application server profile. The new profile includes the sample code and the build script.

2. Open a command prompt and change to this directory:

```
<profile_root>\samples\src\WebSphereBank
```

3. Run the Sample build script:

```
<profile_root>\samples\bin\WebSphereBank\buildbank.bat
```

The script will remove the deploy code from the sample and repackage the EAR file. The resulting EAR file will be in the following directory:

```
<profile_root>\samples\lib\WebSphereBank
```

15.2.2 Importing an EAR file

To work with the WebSphere Bank EAR file in the Application Server Toolkit, we first need to import it.

1. Select **Start → Programs → IBM WebSphere Application Server Toolkit, V6.0 → 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 15-2.

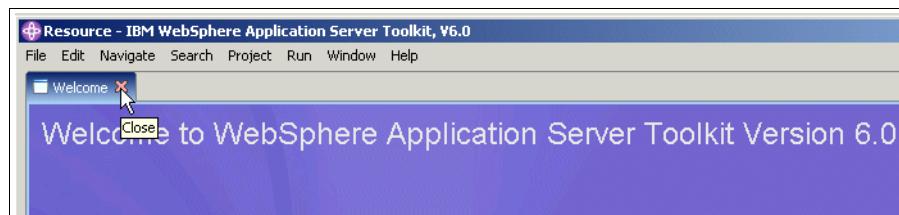


Figure 15-2 Welcome page

4. Select **File → Import**.
5. Select **EAR file** from the list, click **Open**, and then click **Next**.

6. Click **Browse**, select the **WebSphereBank.ear** file and click **Open**. The file is located in the <was_home>\samples\lib\WebSphereBank directory. Then click **Next**.
7. On the second page, accept the defaults with nothing selected and click **Next**.
8. On the third page, accept the defaults and click **Finish**. This will import the code into your workspace.
9. When the code has been imported, click **Yes** on the dialog box asking you to switch to the J2EE perspective.

Note: When you import 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.

The Project explorer view in Figure 15-3 shows the EAR file contents organized by module.

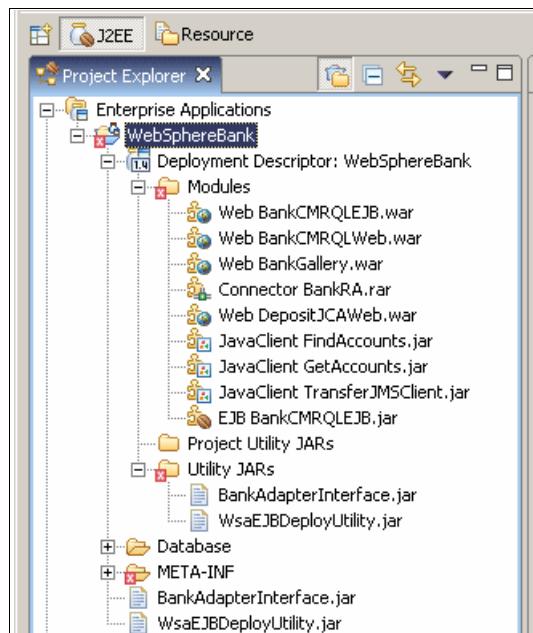
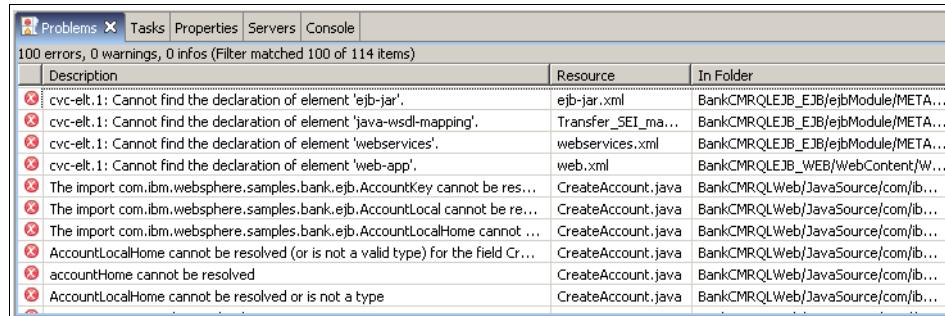


Figure 15-3 WebSphere Bank modules list

As you can see, it has the modules shown in Figure 15-1 on page 848.

If you click the **Problems** tab you see that the workspace has a lot of problems. See Figure 15-4. WebSphere V6 samples were not built using Application Server Toolkit or Rational Application Developer and do not have the extra meta information about the classpaths these tools need. Before doing anything else, we need to remove these problems.



The screenshot shows the Eclipse IDE's Problems view. The title bar says "Problems". Below it, a message says "100 errors, 0 warnings, 0 infos (Filter matched 100 of 114 items)". A table lists the errors:

Description	Resource	In Folder
✗ cvc-elt.1: Cannot find the declaration of element 'ejb-jar'.	ejb-jar.xml	BankCMRQLWeb/EJB/ejbModule/META...
✗ cvc-elt.1: Cannot find the declaration of element 'java-wsdl-mapping'.	Transfer_SEI_ma...	BankCMRQLWeb/EJB/ejbModule/META...
✗ cvc-elt.1: Cannot find the declaration of element 'webservices'.	webservices.xml	BankCMRQLWeb/EJB/ejbModule/META...
✗ cvc-elt.1: Cannot find the declaration of element 'web-app'.	web.xml	BankCMRQLWeb/WEB/WebContent/W...
✗ The import com.ibm.websphere.samples.bank.ejb.AccountKey cannot be resolved.	CreateAccount.java	BankCMRQLWeb/JavaSource/com/ibm/...
✗ The import com.ibm.websphere.samples.bank.ejb.AccountLocal cannot be resolved.	CreateAccount.java	BankCMRQLWeb/JavaSource/com/ibm/...
✗ The import com.ibm.websphere.samples.bank.ejb.AccountLocalHome cannot be resolved.	CreateAccount.java	BankCMRQLWeb/JavaSource/com/ibm/...
✗ AccountLocalHome cannot be resolved (or is not a valid type) for the field Cr...	CreateAccount.java	BankCMRQLWeb/JavaSource/com/ibm/...
✗ accountHome cannot be resolved	CreateAccount.java	BankCMRQLWeb/JavaSource/com/ibm/...
✗ AccountLocalHome cannot be resolved or is not a type	CreateAccount.java	BankCMRQLWeb/JavaSource/com/ibm/...

Figure 15-4 Problems after importing WebSphere Bank EAR file

1. In the Project Explorer, expand **Dynamic Web Projects**.
2. Select and right-click the **BankCMRQLWeb** and select **Properties**. See Figure 15-5 on page 854.

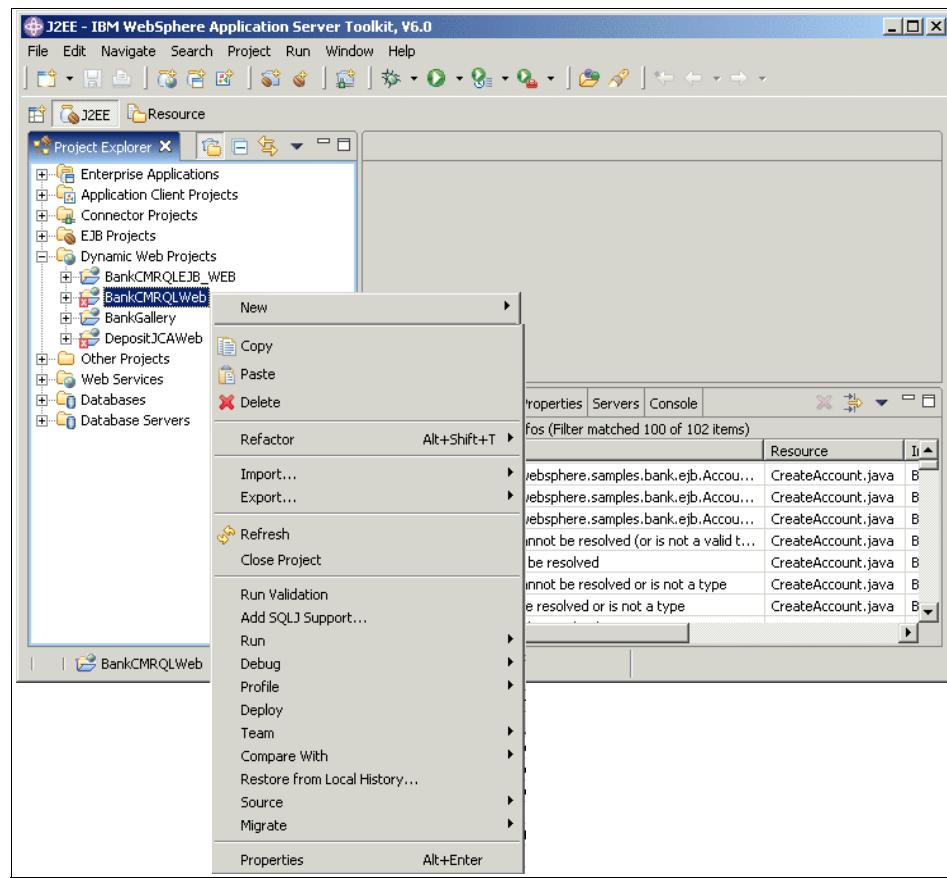


Figure 15-5 Open the properties for a Web module

3. Select the **Java Build Path** section and click the **Projects** tab.
4. Check the **BankCMRQLEJB_EJB** project in the list as shown in Figure 15-6 on page 855 and click **OK**.

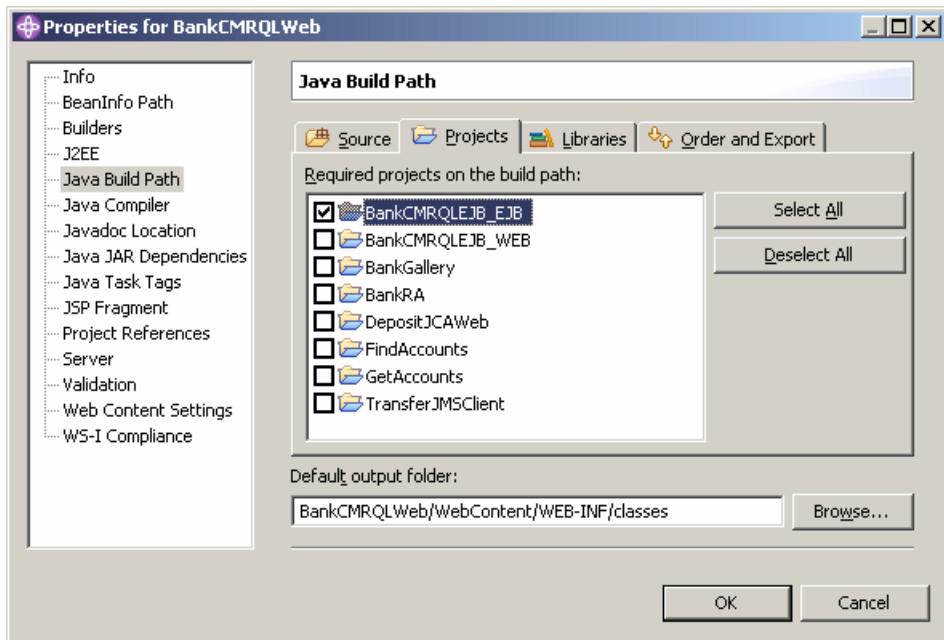


Figure 15-6 Configure Java Build Path for BankCMRQLWeb project

5. Next we need to configure the build path also for the DepositJCAWeb project. Right-click the **DepositJCAWeb** project and select **Properties**.
6. Select the **Java Build Path** section and click the **Projects** tab.
7. Check the **BankRA** project in the list as shown in Figure 15-7 on page 856 and click **OK**.

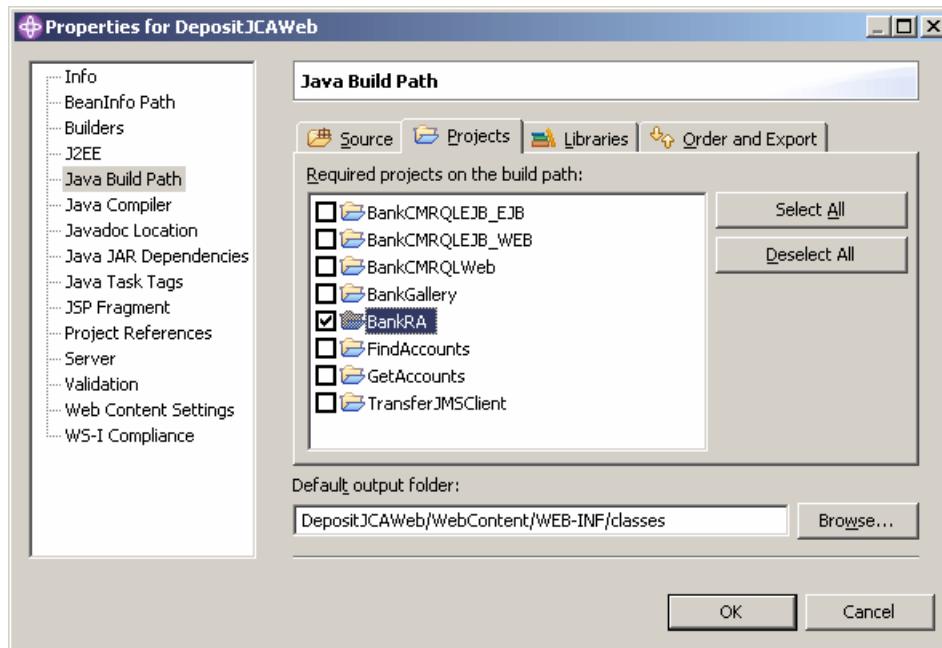


Figure 15-7 Configure Java Build Path for DepositJCAWeb project

This reduces the number of problems to no errors, three warnings and one informational message. We do not need to care about these warnings because the EAR file now builds properly and should work fine for our purposes.

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**.

15.2.3 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 15-8 shows a schematic overview of a J2EE EAR file. In this figure the various deployment descriptors are designated with DD after their name.

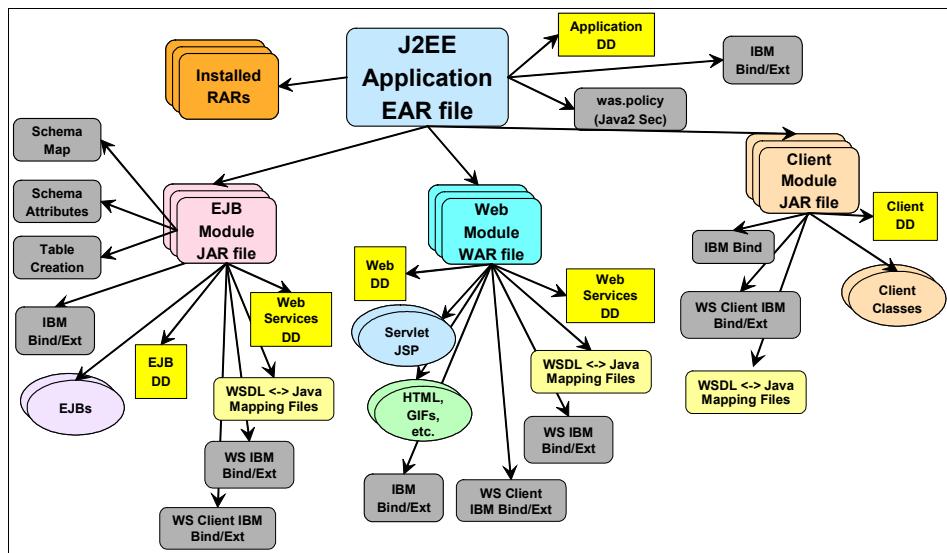


Figure 15-8 J2EE EAR file structure

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 which 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.

New in WebSphere Application Server V6 is also the information contained in the Enhanced EAR files. This information, including settings for the resources required by the application, is stored in an `ibmconfig` subdirectory of the EAR file's `META-INF` directory. In the `ibmconfig` directory are the directories for a WebSphere cell configuration.

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 (EJB Projects), and then expand the module you want to work with.
3. Double-click the **Deployment Descriptor** to open the editor for it. See Figure 15-9 on page 859.

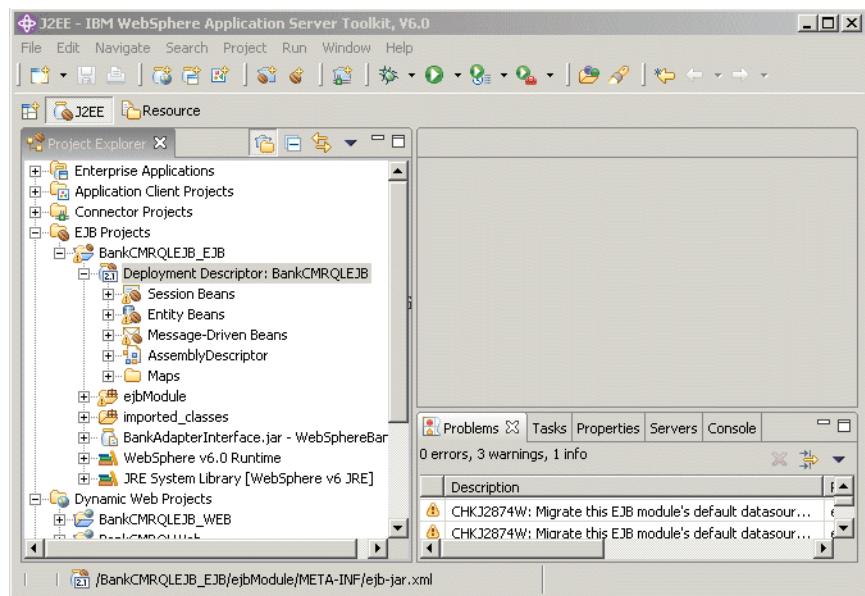


Figure 15-9 Finding the deployment descriptor

4. Figure 15-10 on page 860 shows the deployment descriptor for the WebSphere Bank EJB module, BankCMRQLEJB, open with the deployment descriptor editor.

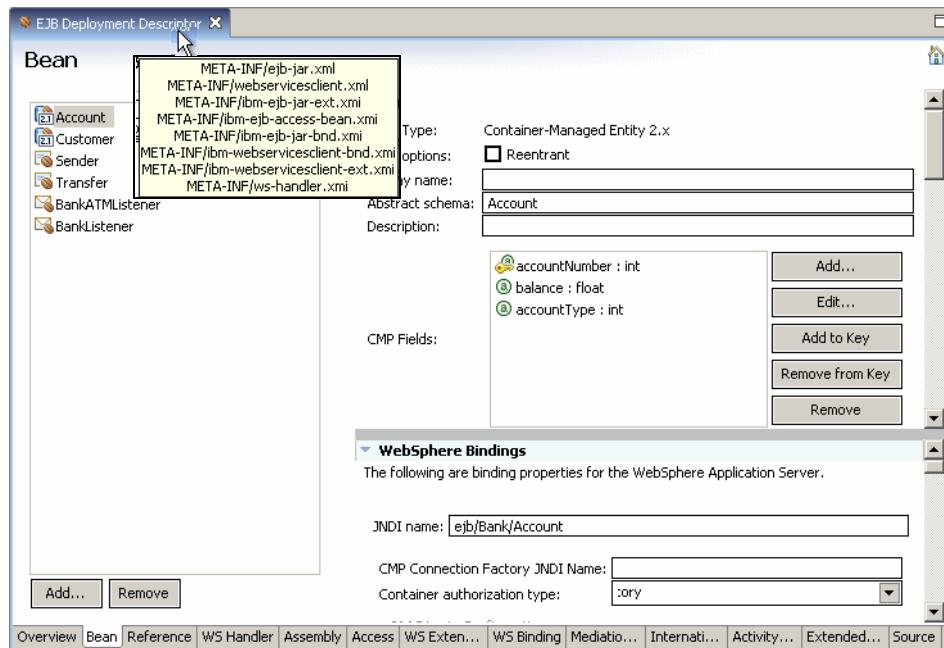


Figure 15-10 EJB deployment descriptor

While the editor shows you information stored in all the relevant deployment descriptor files on the appropriate tabs, the Source tab only shows you the source of the deployment descriptor itself (for example, ejb-jar.xml or web.xml) and not the IBM extensions and bindings stored in the WebSphere-specific deployment descriptor files. If you want to view the results of updates to those files in the source, open each file individually. By hovering over the EJB Deployment Descriptor caption tab you can see the different files that make up the EJB deployment descriptor you are editing. 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.

15.3 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 which will then be modified at deployment time by the deployment team to suit the target environment. Developers use a tool like Rational Application Developer to define bindings, while deployers use the Application Server Toolkit.

All binding definitions are stored in the ibm-xxx-bnd.xmi 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
- ▶ ActivationSpecs for message-driven beans
- ▶ Data source for entity beans

All sections below assume that you have started the Application Server Toolkit and opened the WebSphere Bank application EAR file.

15.3.1 Defining EJB JNDI names

For each session and entity enterprise 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.

The WebSphere Bank enterprise beans are declared to be bound in the ejb/Bank/ subcontext. For clarity, we recommend that you place all enterprise bean JNDI names for an application in a separate subcontext. You can find the JNDI names for the WebSphere Bank session and entity EJBs in Table 15-1.

Table 15-1 Webbank enterprise bean JNDI names

EJB Name	JNDI Name
Sender session bean	ejb/Bank/Sender
Transfer session bean	ejb/Bank/Transfer
Account entity bean	ejb/Bank/Account
Customer entity bean	ejb/Bank/Customer

Use this table and the instructions below to define a JNDI name for each WebSphere Bank enterprise bean:

1. In the Project Explorer view, expand the EJB Projects section.
2. Expand the **BankCMRQLEJB_EJB** module and then expand the **Session Beans** and **Entity Beans** sections under Deployment Descriptor.

3. Double-click the **Sender** session bean. The EJB deployment descriptor editor opens to the Bean page, shown in Figure 15-11.
4. Look for the WebSphere Bindings section in the editor.
5. In the JNDI name field, enter ejb/Bank/Sender.

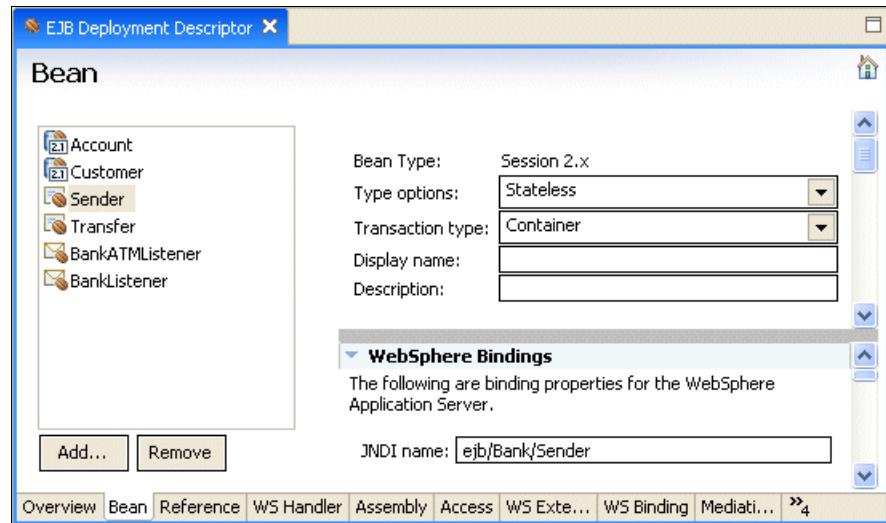


Figure 15-11 Defining EJB JNDI names

6. Repeat these steps for each of the session and entity enterprise beans in the EJB module, that is, Transfer, Account and Customer.
7. Save the deployment descriptor.

15.3.2 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 the WebSphere Bank sample, we use the resource references shown in Table 15-2 on page 863.

Table 15-2 EJB and resource references : JNDI names list

EJB/Resource Reference	Corresponding JNDI Name
ejb/Account	ejb/Bank/Account
ejb/Customer	ejb/Bank/Customer
ejb/Transfer	ejb/Bank/Transfer
ejb/CustomerHome	ejb/Bank/Customer

The last reference, ejb/CustomerHome, is a reference to the remote home interface of the Customer entity EJB. All other are references to their respective EJB's local home.

Follow these steps to bind an EJB local reference to a JNDI name:

1. In the Project Explorer view, expand the **Dynamic Web Projects**.
2. Expand the **BankCMRQLWeb** module and then double-click the deployment descriptor. The editor opens to the overview.
3. Click the **References** tab.
4. Click the **ejb/Account** reference, as shown in Figure 15-12.

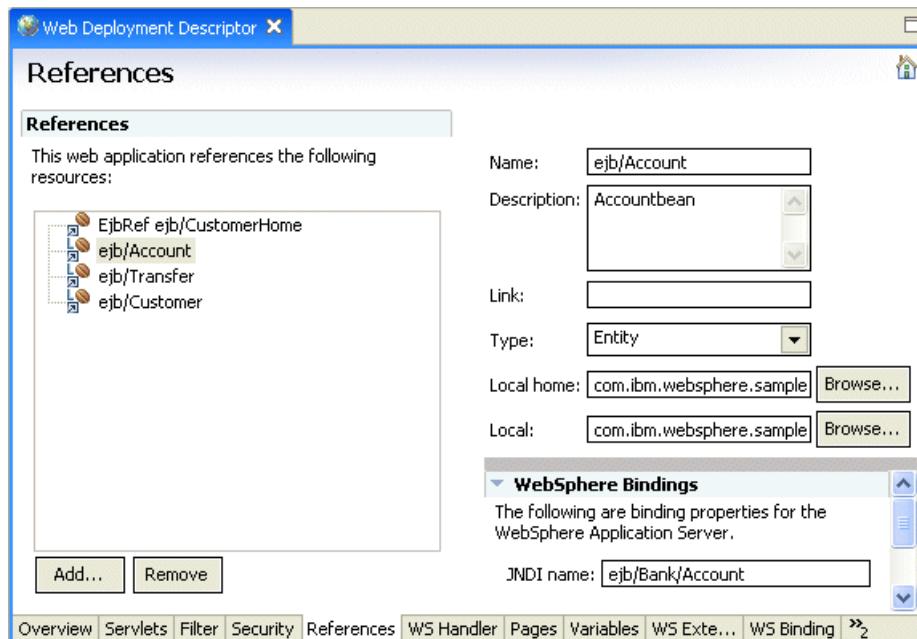


Figure 15-12 Setting EJB References bindings

5. In the WebSphere bindings section, specify ejb/Bank/Account.
6. Repeat these steps for all references in this Web module, using Table 15-2 as a guide for JNDI and resources names.
7. 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.

15.3.3 Binding the message-driven bean to an ActivationSpec

The WebSphere Bank sample includes two message-driven beans. The BankListener is a JMS-type bean and the BankATMListener is a non-JMS-type bean. These message-driven beans must be bound to ActivationSpecs so they can be activated when an incoming message for them arrives.

To do this, follow these steps:

1. In the Project Explorer view, expand the EJB Projects section.
2. Expand the **BankCMRQLEJB_EJB** module and double-click the deployment descriptor.
3. Click the **Bean** tab at the bottom of the editor and select the **BankListener** bean, as shown in Figure 15-13 on page 865.

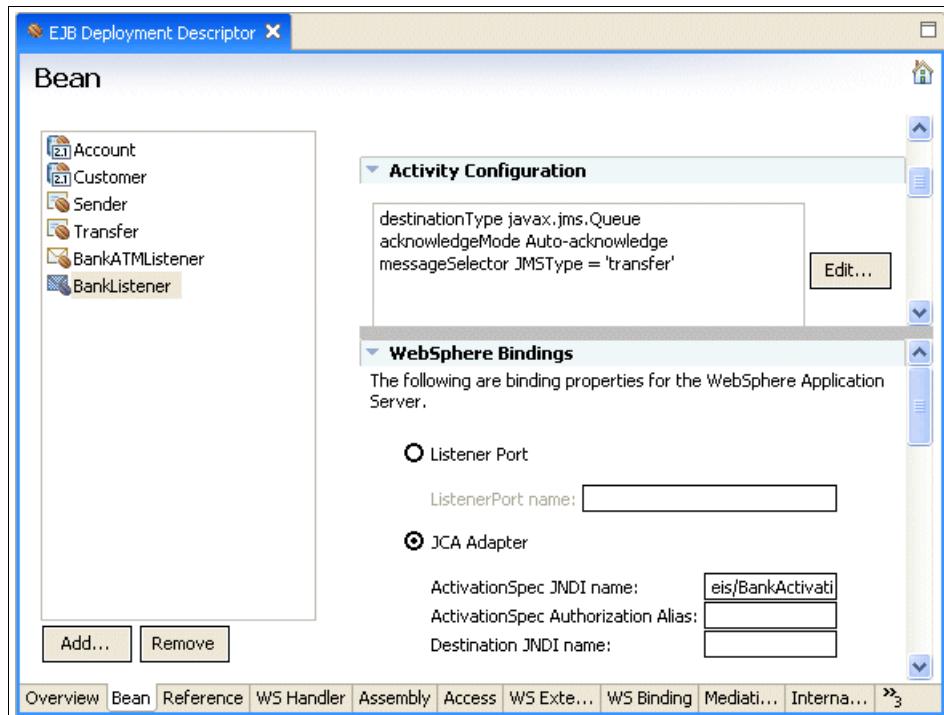


Figure 15-13 Configuring an ActivationSpec for a Message-driven bean

4. In the WebSphere bindings section, set the ActivationSpec JNDI name to eis/BankActivationSpec.
5. Repeat steps 3 on page 864 and 4 for the BankATMListener, but use the ActivationSpec JNDI name eis/com.ibm.websphere.samples.bank.adapter.BankMessageListener.
6. Save the deployment descriptor file.

All bindings are now complete. Save the ready-to-deploy EAR file.

15.3.4 Defining data sources for entity beans

The entity beans in the WebSphere Bank 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 WebSphere Bank 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 jdbc/Bank. See “WebSphere Bank resources used” on page 849.

To bind the WebSphere Bank EJB module to this data source, follow these steps:

1. In the Project Explorer view, expand the **EJB Projects** section.
2. Expand the **BankCMRQLEJB_EJB** module and then double-click the deployment descriptor.
3. In the Overview tab, scroll down and find the WebSphere bindings section, as in Figure 15-14. Check the following fields:

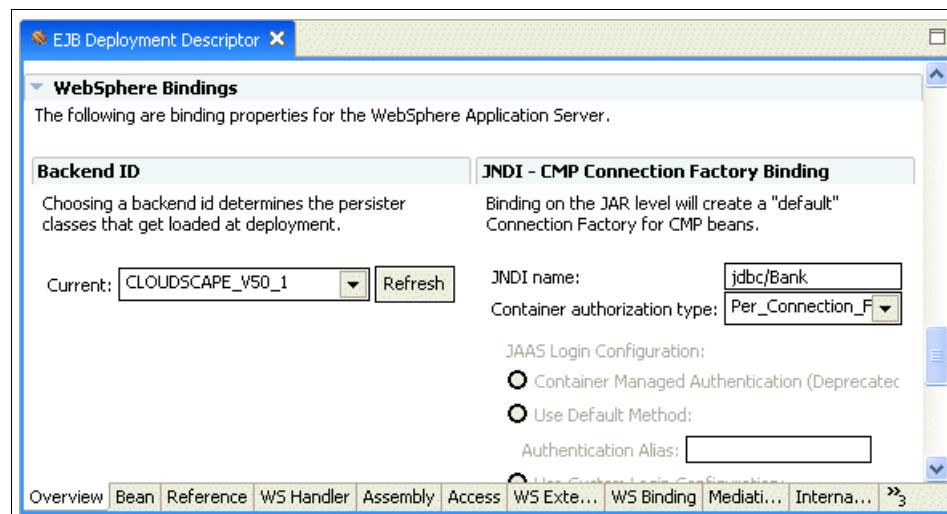


Figure 15-14 Specifying the default CMP data source for the entity EJBs

– **Backend ID**

In EJB 2.x, mapping and schema files make up a back end for EJB 2.x projects. The WebSphere Bank sample ships with a Cloudscape back end defined. Leave this for now, though in “Creating a new database mapping and schema” on page 867, we will create a new backend for DB2 and change this binding to point it.

– **JNDI name**

Enter jdbc/Bank 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 backend ID is used.

- **Container authorization type**

Select **Per_Connection_Factory** for the Container authorization type.

4. Save the deployment descriptor.

Tip: An EJB JAR can contain database mappings and EJB deployed code for multiple databases. Currently the WebSphere Bank sample application contains a database mapping and EJB deployed code only for Cloudscape 5, but we will also add it for DB2 UDB. You can set which backend ID will be used at runtime in the WebSphere bindings section. This choice can also be overridden at deployment time.

Creating a new database mapping and schema

The WebSphere Bank sample is configured to run against a Cloudscape database. However, for the purpose of showing how to create a new database backend 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
2. Right-click the **BankCMRQLEJB_EJB** project 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 Universal Database Express V8.2**, or the corresponding DB2 product and version you are using as the target database. Enter database name **BANK** and leave **NULLID** as the schema name. See Figure 15-15 on page 868.

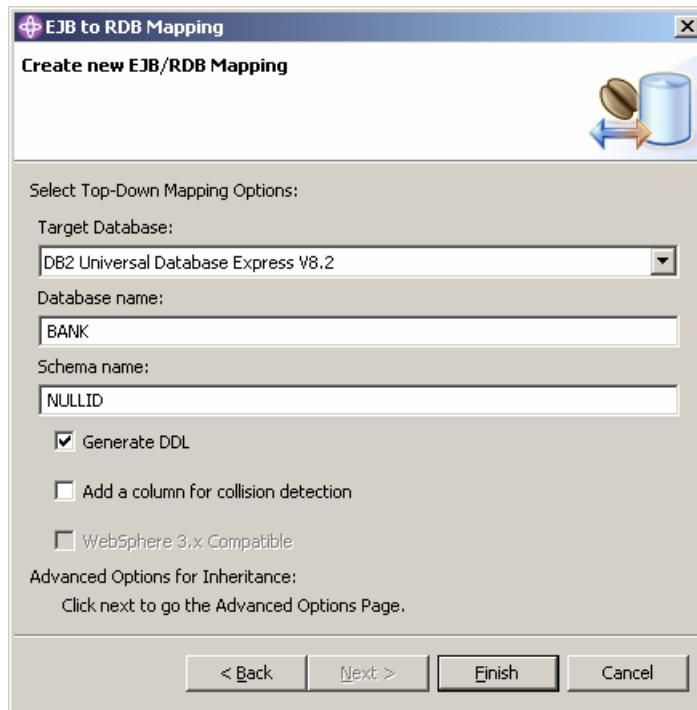


Figure 15-15 Generating a DB2 mapping

6. Click **Finish**.

A Table.ddl file containing the script to set up the DB2 tables is created in the `BankCMRQLEJB_EJB\ejbModule\META-INF\backends\DB2EXPRESS_V82_1` directory of the Application Server Toolkit workspace. You will need this script when creating the DB2 BANK database before deploying WebSphere Bank. The file has the commands shown in Example 15-1.

Example 15-1 WebSphere Bank Table.ddl

```
CREATE TABLE ACCOUNT
(ACCOUNTNUMBER INTEGER NOT NULL,
 BALANCE REAL NOT NULL,
 ACCOUNTTYPE INTEGER NOT NULL,
 ACCOUNTSCUSTOMERINVERSE_CUST02 BIGINT);

ALTER TABLE ACCOUNT
 ADD CONSTRAINT PK_ACCOUNT PRIMARY KEY (ACCOUNTNUMBER);

CREATE TABLE CUSTOMER
(CUSTOMERNUMBER BIGINT NOT NULL,
 FIRSTNAME VARCHAR(250),
```

```
TAXID VARCHAR(250),  
LASTNAME VARCHAR(250));  
  
ALTER TABLE CUSTOMER  
ADD CONSTRAINT PK_CUSTOMER PRIMARY KEY (CUSTOMERNUMBER);
```

7. 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.
8. Right-click the **BankCMRQLEJB_EJB** project again and select **Deploy**. This generates the EJB deployed code, as shown in Figure 15-16.

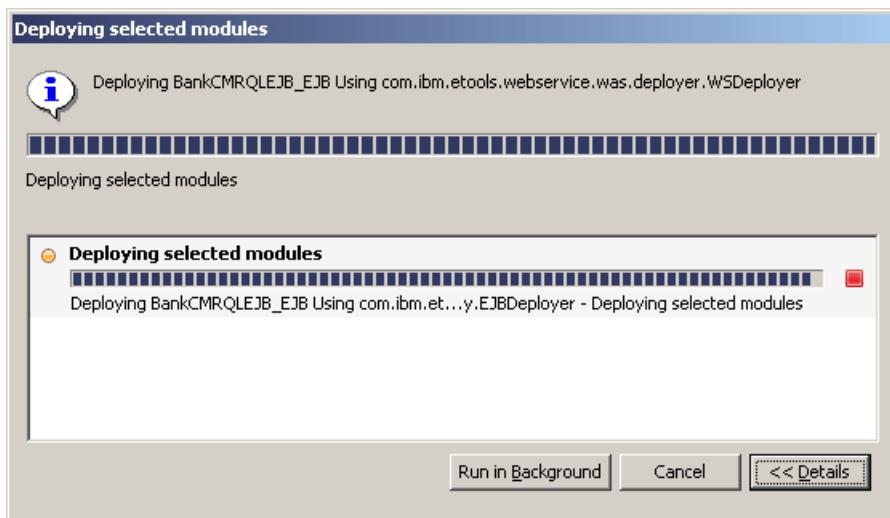


Figure 15-16 Generating EJB deployed code

If you receive the following error during the deploy code generation you can ignore it:

```
Deployment from com.ibm.etools.webservice.was.deployer.WSDeployer had  
errors: Deployment error:
```

Change the backend ID

Because we have now created a new database backend map, we can set the default backend 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 do the bottom of the Overview tab and select **DB2EXPRESS_V82_1** as the Current Backend ID as shown in Figure 15-17.

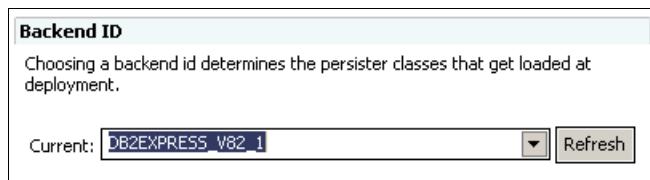


Figure 15-17 Setting default backend id for EAR file

2. Press **Ctrl-S** to save the deployment descriptor.

15.4 IBM EJB extensions: EJB caching options

This section discusses the caching options for entity and stateful session beans.

15.4.1 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 can 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 15-2 on page 871. The first time the entity bean is used, its runtime 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 15-2 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 15-3.

Example 15-3 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 15-4.

Example 15-4 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 15-3 to choose the right combination.

Table 15-3 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 15-18.

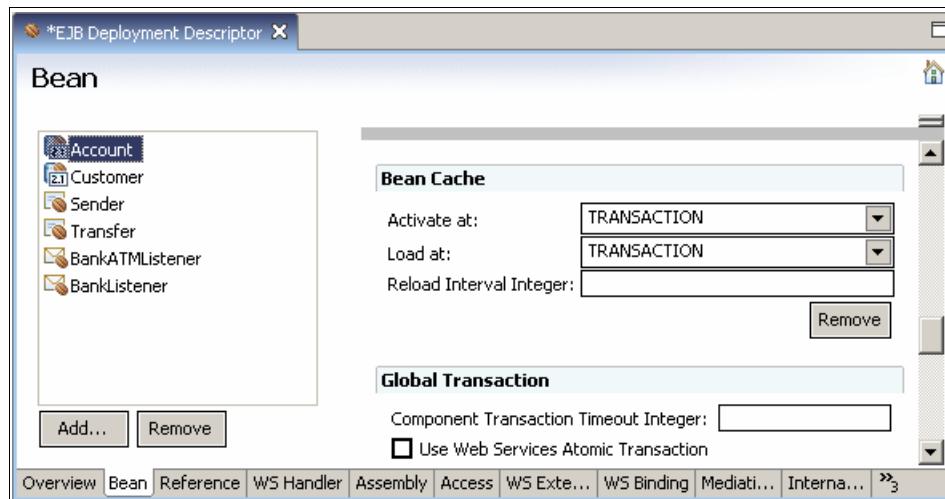


Figure 15-18 Setting the activate and load settings for entity beans

4. Select the Activate at and Load at options according to Table 15-3 on page 872.
5. Close and save the deployment descriptor.

The settings are saved in the `ibm-ejb-jar-ext.xmi` 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.

15.4.2 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 15-19). Select the bean and scroll down to the Bean Cache category.

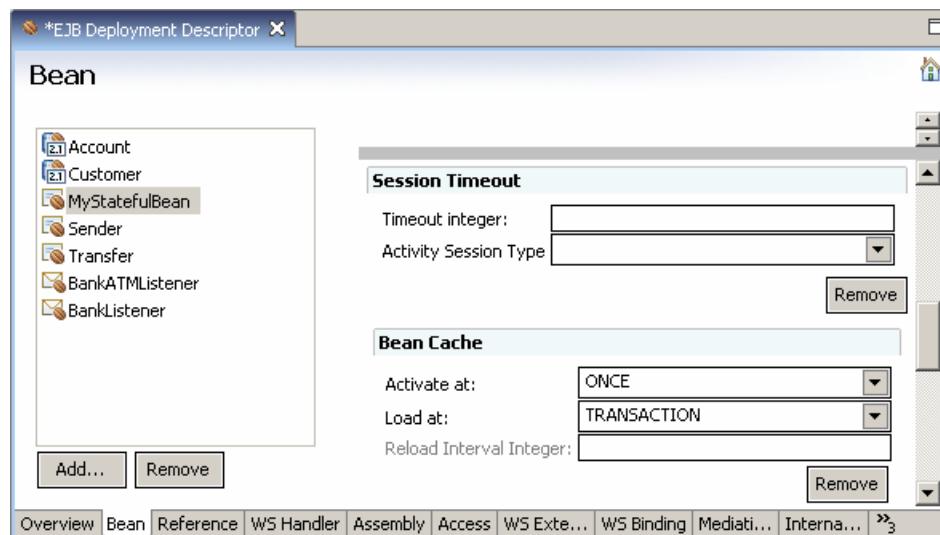


Figure 15-19 Activate settings for stateful session beans

15.4.3 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 15-5 on page 875:

Example 15-5 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 15-19 on page 874. 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.

15.5 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.

15.5.1 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 assures 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 Jim Gray's classic book on transaction processing, *Transaction Processing: Concepts and Techniques*. Table 15-4 shows a mapping between EJB and DB2 isolation levels.

Table 15-4 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.

15.5.2 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 a 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.

15.5.3 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 15-5 on page 879.

Table 15-5 Access intent policies

Access Intent Policy	Concurrency control	Used for update	Transaction isolation level	Notes
wsPessimisticRead	pessimistic	No	read committed	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	No	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	No (Oracle, 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	
wsOptimisticUpdate	optimistic	No	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.

The Consultation session bean obtains the CustomerAccount balance using getBranchAccountBalance, as in Example 15-6:

Example 15-6 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 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 also includes a new 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 15-20 on page 883. 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.

In V6, support has been added for a new optimistic concurrency control scheme for EJB 2.x CMP entity beans. This new support 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 `<beanname>FunctionSet` generated class.

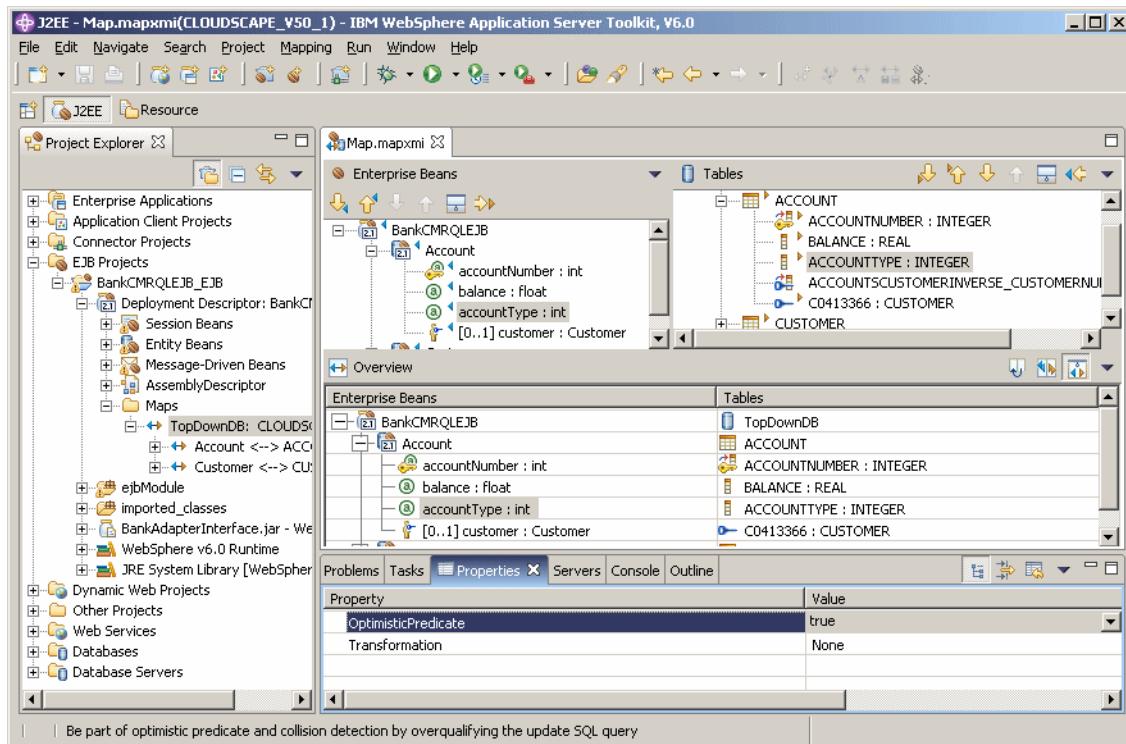


Figure 15-20 Optimistic predicate property

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.

15.5.4 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 15-21.

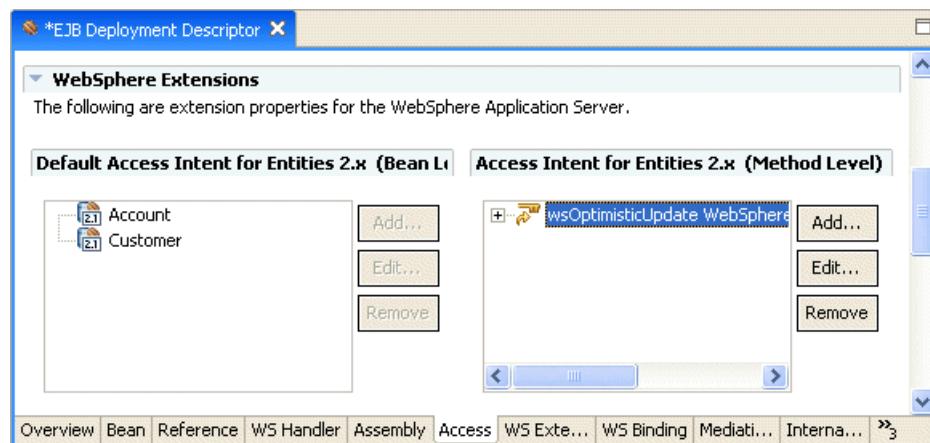


Figure 15-21 Adding a read ahead hint

In the wizard, the Read Ahead Hint check box is enabled only with access intent policies with optimistic concurrency. Read-ahead is limited to optimistic policies because locking persistent data store for all beans represented in the hint would be more likely to cause lock conflicts, and optimistic policies do not obtain locks until immediately before the database operation.

See Figure 15-22 on page 885.

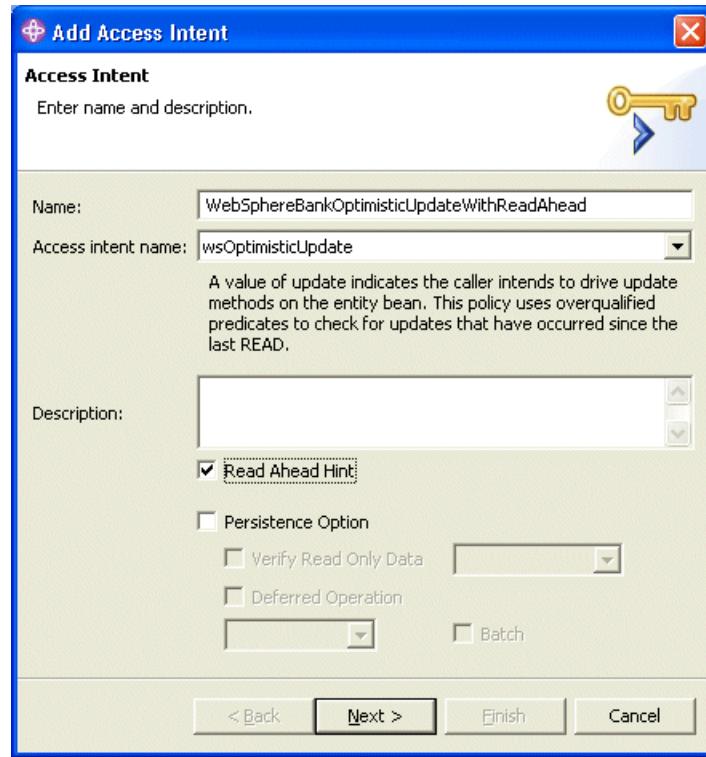


Figure 15-22 Specifying read-ahead hint

4. Follow the panels in the wizard to select the beans and methods, then click **Finish**.

15.5.5 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.
```

15.6 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.

15.7 IBM Web module extensions

WebSphere Application Server V6 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 15-23.

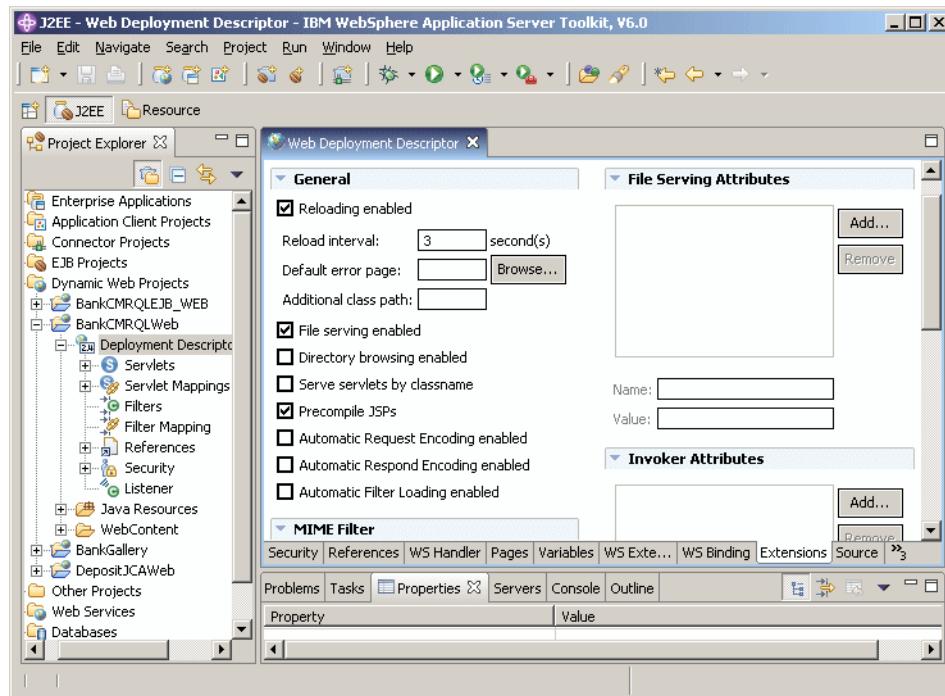


Figure 15-23 Web module extensions

15.7.1 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, WebSphere's HTTP 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 HTTP server to serve static content, you can experience better performance than using WebSphere in this instance, because the HTTP server is serving the content directly. Moreover, an HTTP server has much 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.

15.7.2 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.

15.7.3 Serve servlets by class name

The invoker servlet can be used to invoke servlets by class name. Note 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.

This option is turned off by default.

15.7.4 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.

15.7.5 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.

15.7.6 JSP attributes

The following options can be set for the JSP compiler:

- ▶ 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 servlets code will be generated. If this string is not set, code is created under:

`<was_home>\temp\<hostname>\<application_server_name>\<applicationname>\<web modulename>.`

15.7.7 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** 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.

15.8 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.
1. Check the **Shared httpsession context** option in the WebSphere Extensions section. See Figure 15-24 on page 890.

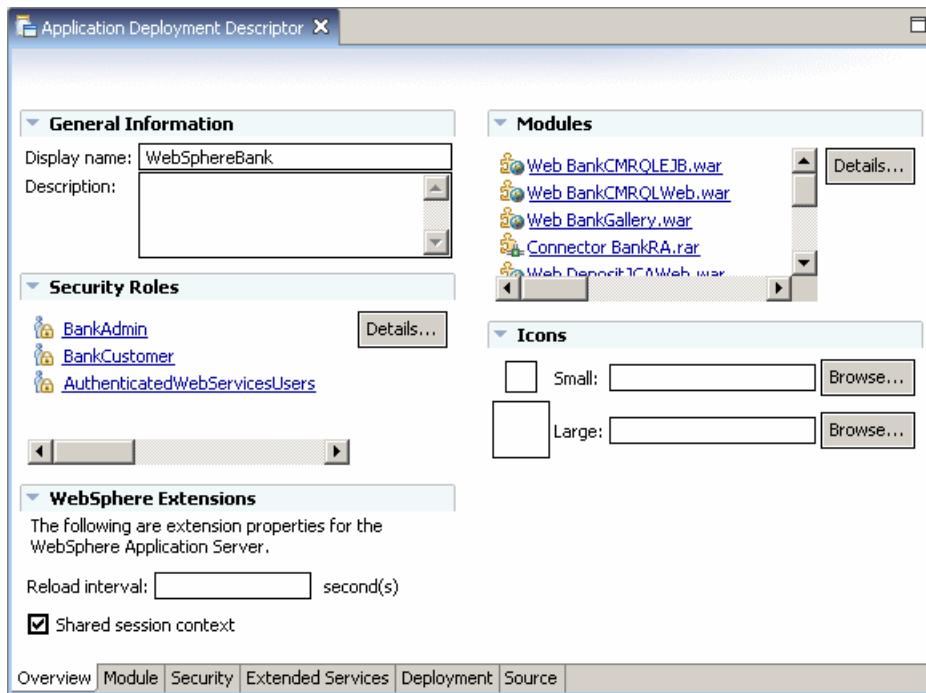


Figure 15-24 EAR deployment descriptor

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.

15.9 Exporting WebSphere Bank 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 WebSphere Bank sample application, do the following:

1. Select **File → Export**.
2. Select **EAR file** as export target and click **Next**.
3. Select to export the **WebSphereBank EAR** project and enter a suitable destination for the EAR file, such as **C:\WebSphereBank.ear**. Then click **Finish**. See Figure 15-25.

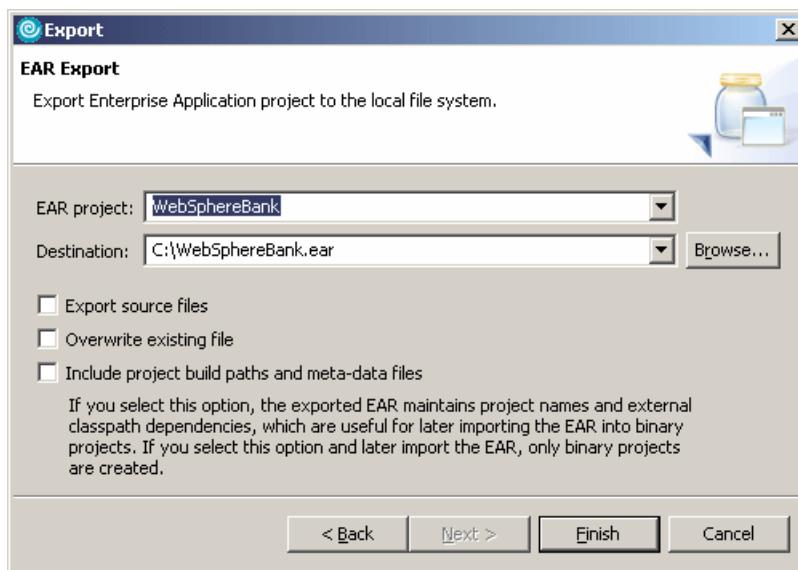


Figure 15-25 Exporting WebSphere Bank EAR file

15.10 WebSphere Enhanced EAR

The Enhanced EAR, introduced in WebSphere Application Server V6, 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 15-6 shows the resources supported by the Enhanced EAR and the scope in which they are created:

Table 15-6 Scope for resources in WebSphere Enhanced EAR file

Resource	Scope
JDBC providers	Application
Data sources	Application
Substitution variables	Application
Class loader policies	Application
Shared libraries	Server
JAAS authentication aliases	Cell
Virtual hosts	Cell

When an Enhanced EAR is deployed to a WebSphere Application Server V6 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.

Note: Resources created at Application level scope are limited in visibility to only that application. Also, these resources are not visible from the WebSphere administrative console. To modify these resources once the application is deployed in WebSphere, you have to use wsadmin. An example of this is changing the connection pool settings for a data source.

Worth noticing is that in WebSphere Application Server V6 the Enhanced EAR file does not support configuration information about, for example, JMS queues. These resources will still have to be configured using the WebSphere administrative console or wsadmin before the application is deployed.

15.10.1 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**, then the application.
2. Double-click **Deployment Descriptor** and select the **Deployment** tab. This opens up the Enhanced EAR editor as shown Figure 15-26.

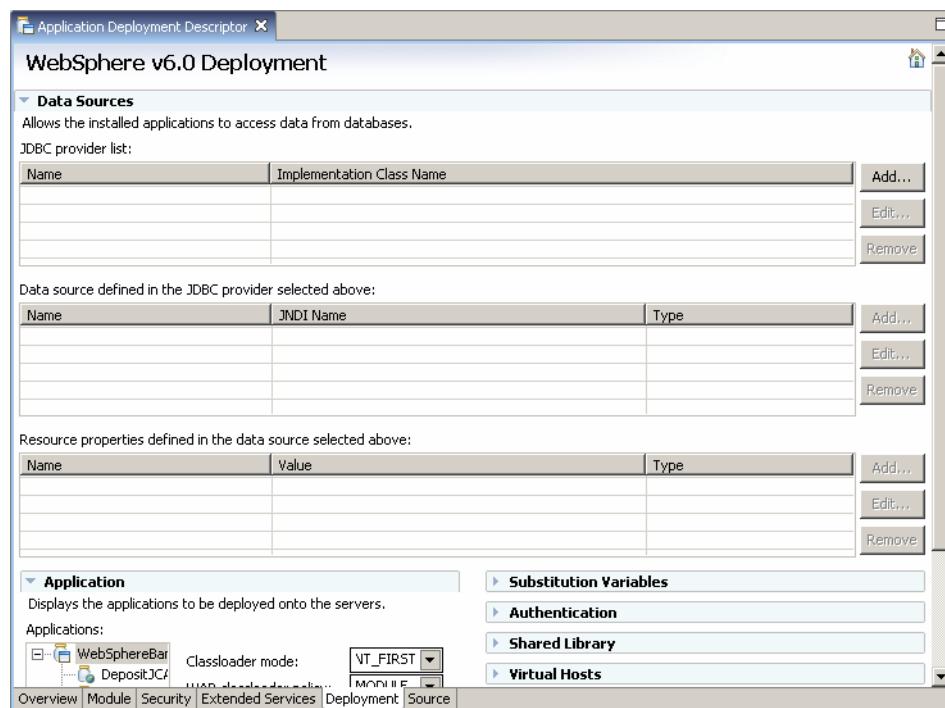


Figure 15-26 WebSphere Enhanced EAR editor

In the Application section in Figure 15-27, you can see the class loader policies and class loader mode configured for each of the containing module. WebSphere Bank runs fine with the default policies and modes so they do not need to be changed.

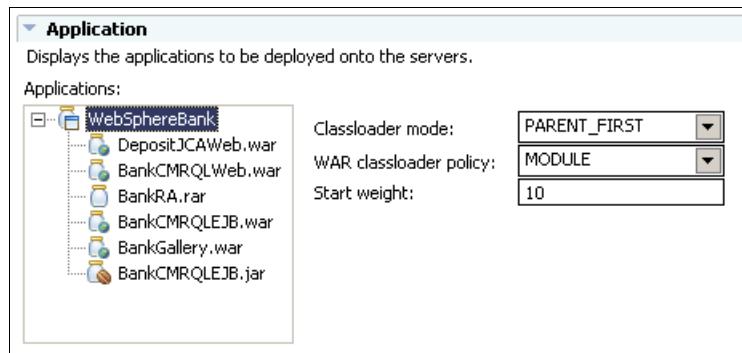


Figure 15-27 Configuring class loader mode and class loader policies

In the Enhanced EAR editor you also configure resources such as JDBC providers, datasources, class loader policies, JAAS authentication aliases, virtual hosts and so forth.

To configure the WebSphere Bank 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.webspherebank.com.

Configuring a JAAS authentication alias

To configure the the JAAS authentication alias, do the following:

1. In the Deployment tab, expand the **Authentication** section.
2. Click the **Add** button. See Figure 15-28 on page 895.

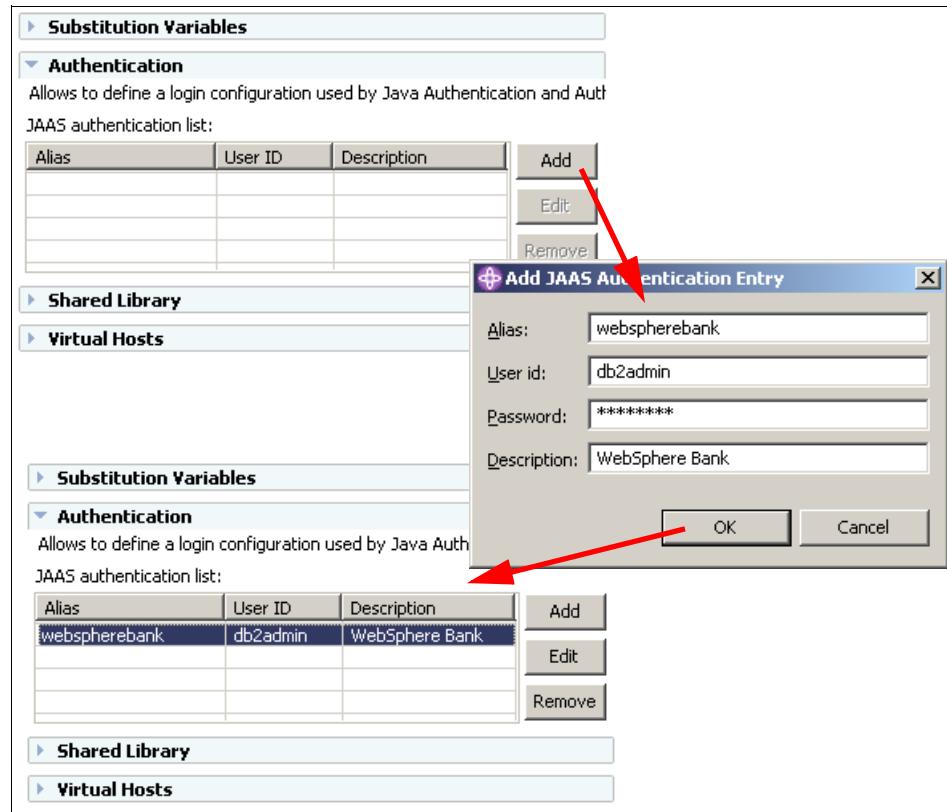


Figure 15-28 Configuring JAAS authentication alias for WebSphere Bank

3. In the dialog box that displays, enter:
 - webspherelbank as the alias
 - A user ID with access to the BANK database
 - The password for the user ID.
 - WebSphere Bank as the description
4. Click **OK**.

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. See Figure 15-29 on page 896.

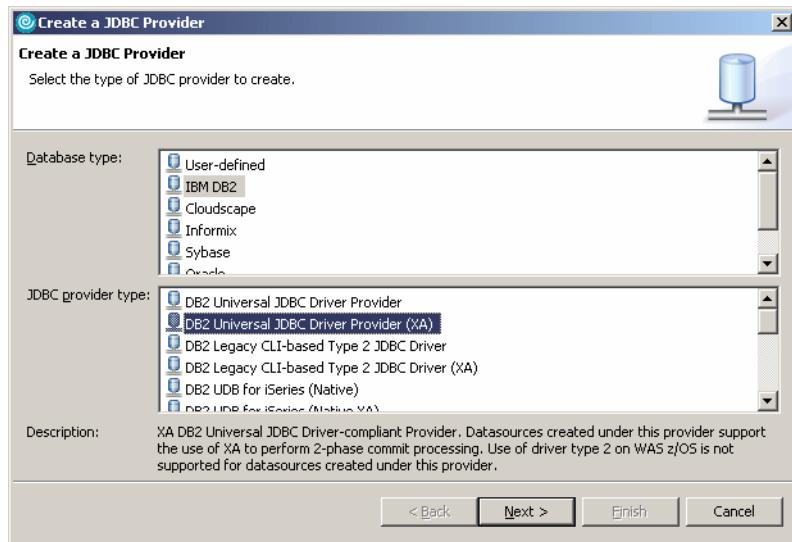


Figure 15-29 Creating a DB2 JDBC Provider

2. In the dialog box:
 - Select **IBM DB2** as the Database type
 - Select **DB2 Universal JDBC Driver Provider (XA)** as the JDBC provider typeClick **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 15-30 on page 897.

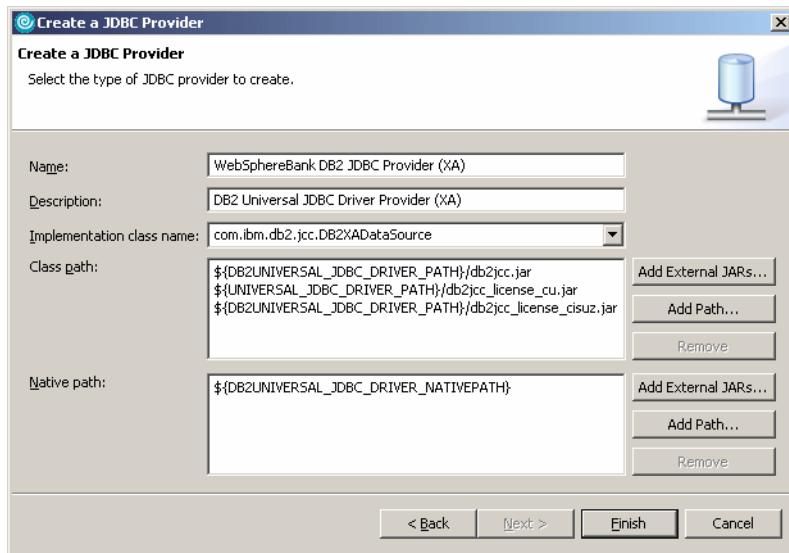


Figure 15-30 Creating a DB2 JDBC provider

Click **Finish**.

4. Select the **WebSphereBank DB2 JDBC Provider (XA)** you just created and click the **Add** button next to the Data source list, as in Figure 15-31.

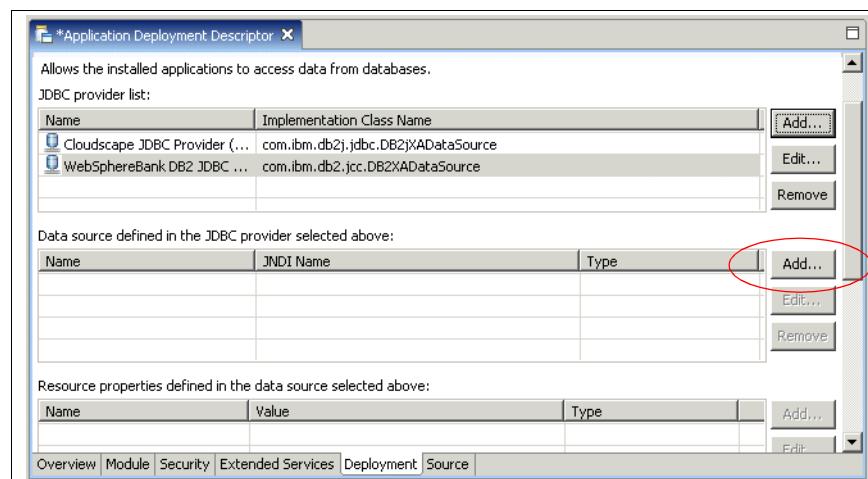


Figure 15-31 Add a data source

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 15-32.

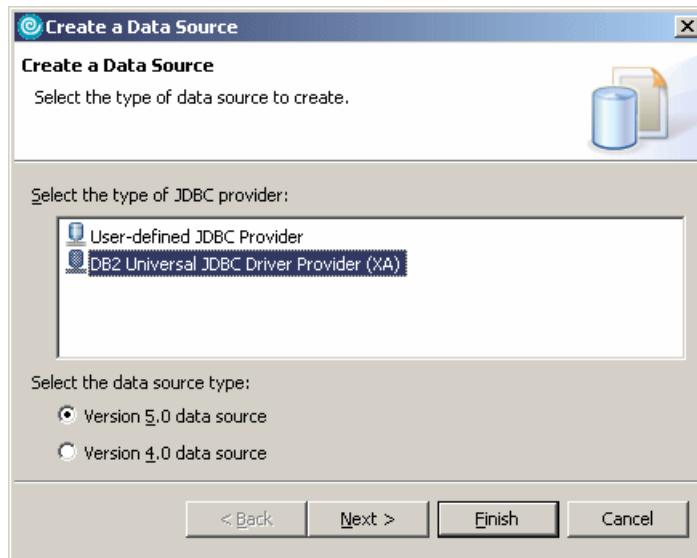


Figure 15-32 Creating a DB2 data source

Click **Next**.

6. In the dialog box displayed enter the appropriate values for the DB2 data source. See Figure 15-33 on page 899.

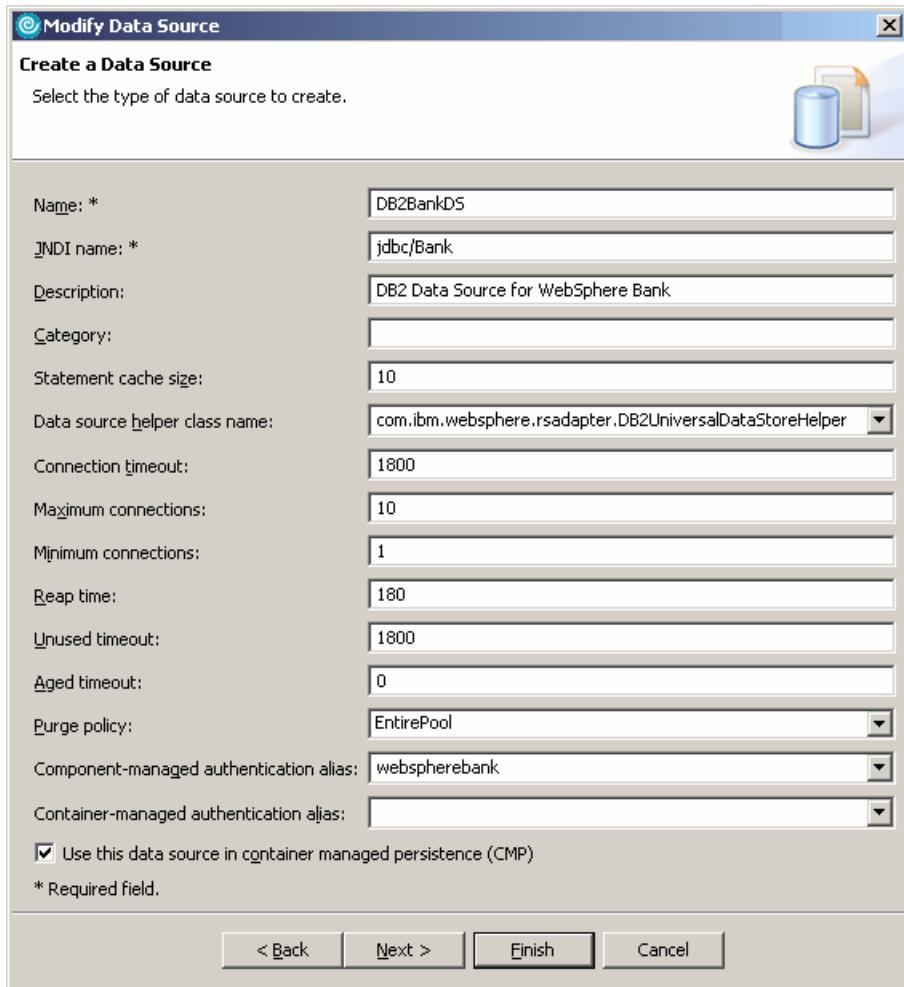


Figure 15-33 Creating a DB2 data source

- Enter DB2BankDS as the name.
 - Enter jdbc/Bank as the JNDI name.
 - Enter DB2 Data Source for WebSphere Bank as the description.
 - Select **webspheredbank** 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 BANK as the value. See Figure 15-34 on page 900.

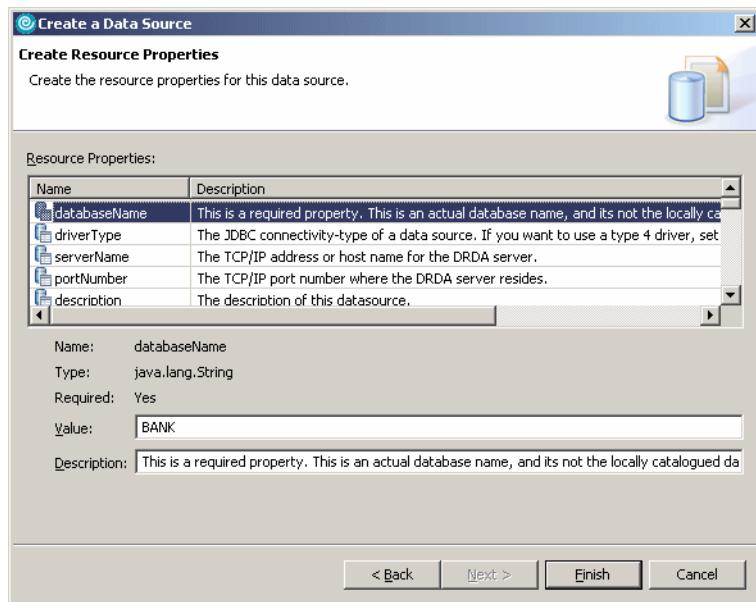


Figure 15-34 Setting database properties for DB2 data source

Click **Finish**.

When you are finished, your data source configuration should look like Figure 15-35 on page 901.

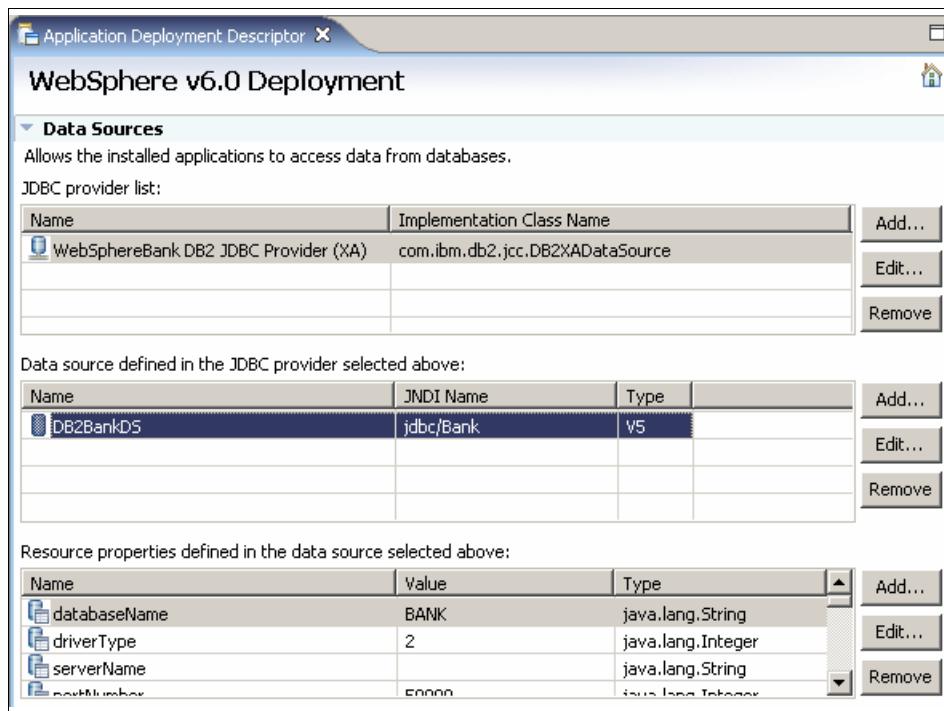


Figure 15-35 DB2 data source configured

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 `webspherebank_host` and click **OK**.
3. Click the **Add** button next to the Host aliases list.
4. In the Add Host Alias Entry dialog box, enter `www.webspherebank.com` for the host name and 80 for the port number. Click **OK**.

Repeat the procedure to add port numbers 9085 and 443 as well. We will use these port numbers when we deploy the application later.

The flow is shown in Figure 15-36 on page 902.

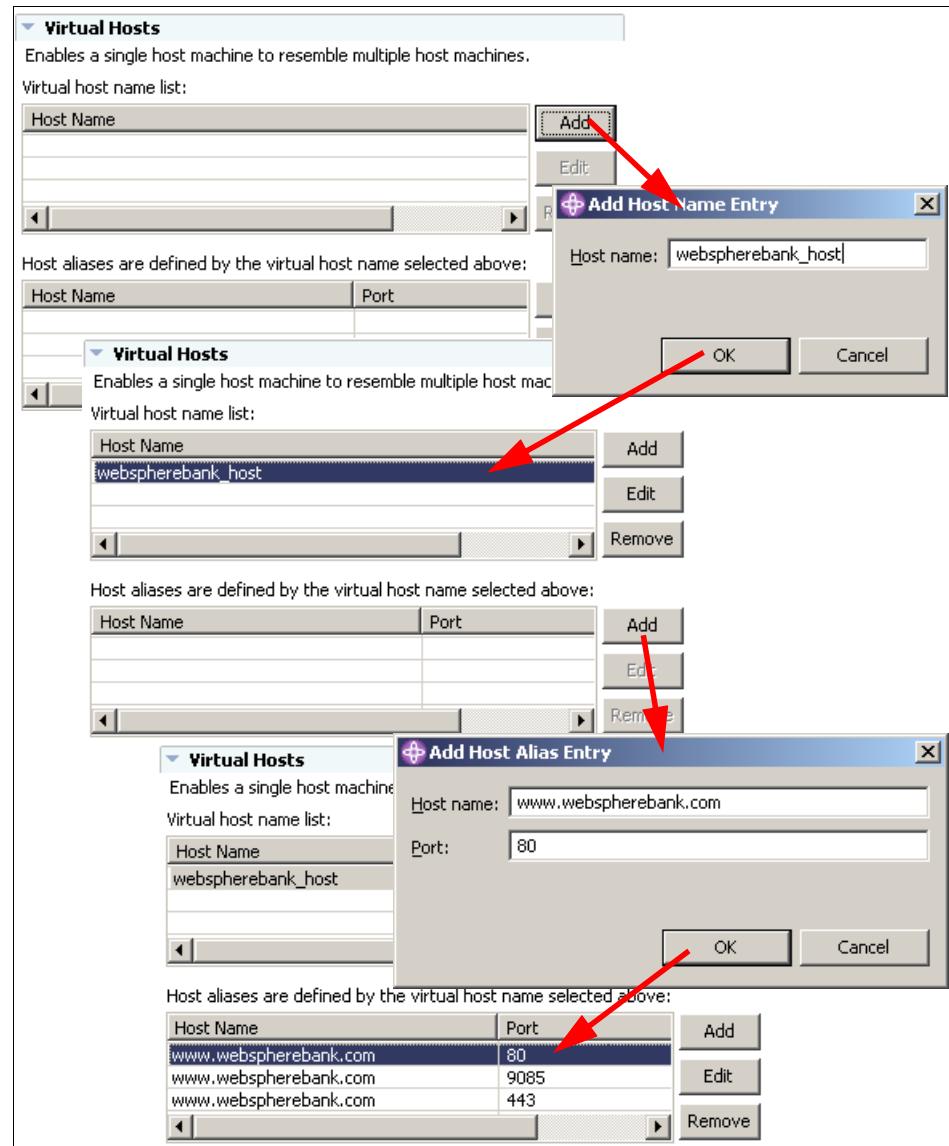


Figure 15-36 Configuring the virtual host for WebSphere Bank

5. 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, webspherebank_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, and so is the case also for the Web modules of the WebSphere Bank 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 webspherebank_host instead, do the following:

1. Expand **Dynamic Web Project** in the Project Explorer view.
2. Expand the **BankCMRQLEJB_WEB** project and double-click **Deployment Descriptor**.
3. Scroll to the bottom of the Overview page and replace default_host with webspherebank_host as shown in Figure 15-37.



Figure 15-37 Setting default virtual host for a Web module

4. Save the deployment descriptor by pressing **Ctrl-S** and then **close** it.
5. Repeat these steps for the other Web modules (BankCMRQLWeb, BankGallery and DepositJCAWeb) as well. Remember to save each Web module's deployment descriptor.

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 WebSphere cell configuration, as seen in Figure 15-38 on page 904. You can also see the scope level where each resource is configured.

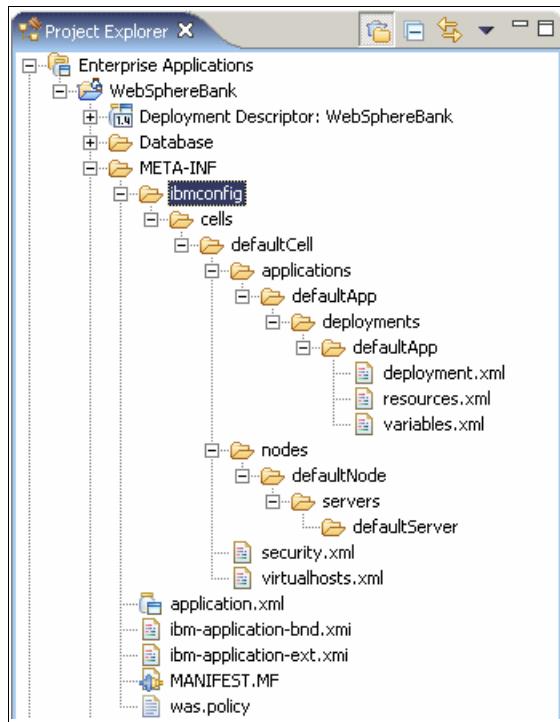


Figure 15-38 Enhanced EAR file contents

After you have re-packaged the application into an Enhanced EAR, export it as explained in “Exporting WebSphere Bank EAR file” on page 891.

At deployment time, WebSphere Application Server V6 uses this information to automatically create the resources.

15.11 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. This is what is done for the BankAdapterInterface.jar and WsaEJBDeployUtility.jar, which are used both by servlets and EJBs.
- ▶ Utility classes used by multiple applications can be placed on a directory referenced through a shared library definition.

See 14.4, “Learning class loaders by example” on page 835 for more details on how WebSphere finds and loads classes.



Deploying applications

In Chapter 15, “Packaging applications” on page 847, we discussed 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 Chapter 6, “Administration with scripting” on page 267. You might also want to take a look at Chapter 17, “WebSphere Rapid Deployment” on page 957 to see if using the new automatic application installation mode of the WebSphere Rapid Deployment features is appropriate for your environment.

WebSphere Application Server V6 supports the J2EE Deployment API Specification (JSR-88), which defines standard APIs to enable deployment of J2EE applications and standalone 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.

16.1 Preparing the environment

In this chapter we show you how to set up a fairly complete environment for the WebSphere Bank application and deploy the EAR file. Many times, however, you might not need or want to customize the environment as far 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 16.3, “Deploying the application” on page 930.

The steps in this section are performed typically by the application deployer. To deploy the WebSphere Bank application, do the following:

1. Create the DB2 database for WebSphere Bank. This step is required.
2. Create an environment variable for WebSphere Bank 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.
7. Configure WebSphere messaging. This step is required.

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. 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.

16.1.1 Creating the WebSphere Bank DB2 database

The WebSphere samples by default use Cloudscape as the database. However, in this chapter we will configure WebSphere Bank 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 16-1 on page 909.

Example 16-1 Creating the DB2 database

```
db2 create database bank
db2 connect to bank user <user_id> using <password>
db2 -tvf Table.ddl
db2 connect reset
```

The Table.ddl file is located in the BankCMRQLEJB_EJB\ejbModule\META-INF\backends\DB2EXPRESS_V82_1 directory of your Application Server Toolkit workspace. See “Creating a new database mapping and schema” on page 867.

16.1.2 Creating a WEBSPHEREBANK_ROOT environment variable

It is recommended 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 WEBSPHEREBANK_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 WEBSPHEREBANK_ROOT variable will need to be defined at the node level if all members of a cluster are on the same node, and at cell level if the cluster spans multiple nodes. Use the steps in 5.1.10, “Using variables” on page 179 to create a WEBSPHEREBANK_ROOT variable with a value of C:\apps\WebSphereBank.

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> etc. 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 <was_home>/installedApps directory and the logs files are written to the <was_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 WEBSPHEREBANK_ROOT variable before proceeding. If the directory is not created, the application server will not start.

16.1.3 Creating the WebSphere Bank 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 WebSphere Bank sample application.

Note: For a full discussion of application server properties, see 5.4, “Working with application servers” on page 190.

To create an application server, do the following:

1. Select **Servers** → **Application Servers**.
2. Click the **New** button and provide the information as shown in Figure 16-1 on page 911.

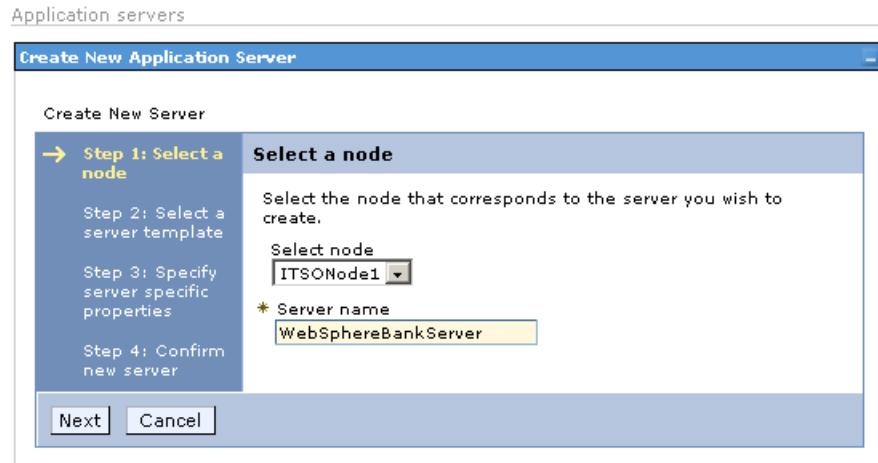


Figure 16-1 Creating the WebSphere Bank application server

– **Node**

Select the node on which the application server will be created.

– **Server name**

Enter the application server name, such as WebSphereBankServer.

Click **Next**.

3. In Step 2, select which server template to use as the base for this new application server. If you have not created any templates, your only choice is to 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, **WebSphereBankServer**, 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 `${USER_INSTALL_ROOT}` to `${WEBSPHEREBANK_ROOT}/workingDir`.
4. Click **OK**.

Note: The working directory will not be created if you use a composed path, such as C:/apps/WebSphereBank/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. These properties are discussed in detail in Chapter 9, “Problem determination” on page 417. 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:

```
 ${WEBSPHEREBANK_ROOT}/logs/SystemOut.log  
 ${WEBSPHEREBANK_ROOT}/logs/SystemErr.log
```

Click **OK**.

3. Select Diagnostic Trace.

Each component of the WebSphere Application Server is enabled for tracing with the JRs interface. This trace can be changed dynamically while the process is running from the Runtime tab, or added to the application server definition from the Configuration tab. As shown in Figure 16-2, 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 WEBSPHEREBANK_ROOT variable and select the **Log Analyzer** format.

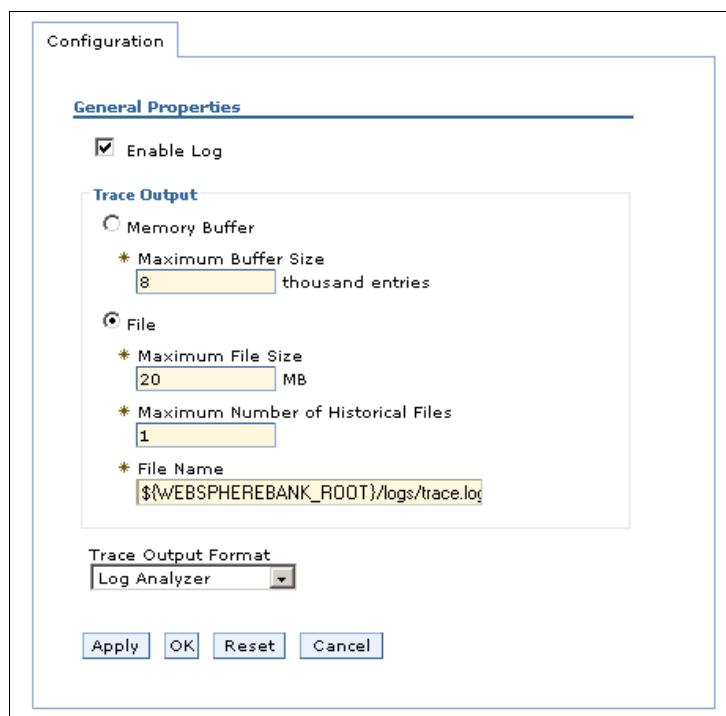


Figure 16-2 Specifying diagnostic trace service options

Click **OK**.

4. Select the 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:

```
 ${WEBSPHEREBANK_ROOT}/logs/native_stdout.log  
 ${WEBSPHEREBANK_ROOT}/logs/native_stderr.log
```

Click **OK**.

5. All log files produced by the application server are now redirected to the \${WEBSPHEREBANK_ROOT}/logs directory. **Save** the configuration.

Note: The rest of this example assumes a default HTTP port of 9085 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 **WebSphereBankServer**.
3. Select **Ports** in the **Communications** section.
4. Scroll down the page and note the port listed for **WC_defaulthost**.

16.1.4 Defining the WebSphere Bank 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 901.

Web modules need to be bound to a specific virtual host. For our sample, we chose to bind the WebSphere Bank Web module to a specific virtual host called webspherebank_host. This virtual host has the following host aliases:

- ▶ www.webspherebank.com:80
- ▶ www.webspherebank.com:9085
- ▶ www.webspherebank.com:443 (for SSL access)

Any request starting with <webspherebank_host_alias>/WebSphereBank, such as <http://www.webspherebank.com:9085/WebSphereBank>, is served by the WebSphere Bank application.

Tip: You can restrict the list of hosts used to access the WebSphere Bank Web application by removing hosts from the virtual host definition.

Imagine you want to prevent users from directly accessing the WebSphere Bank application from the WebSphere internal HTTP server when they invoke <http://www.webspherebank.com:9085/WebSphereBank>. In other words, you want to force all requests to go through the Web server plug-in. You can achieve this by removing www.webspherebank.com:9085 from the virtual host aliases list. The application server will still listen on that port, and that port will still be used in the plug-in configuration file. However, the application server will prevent any direct call to this port.

To create the webspheredbank_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, webspheredbank_host.
4. Click **Apply**.
5. Select **Host Aliases** in the Additional Properties section.
6. Add the three aliases shown in Figure 16-3 by clicking **New**, entering the values, and clicking **OK**.

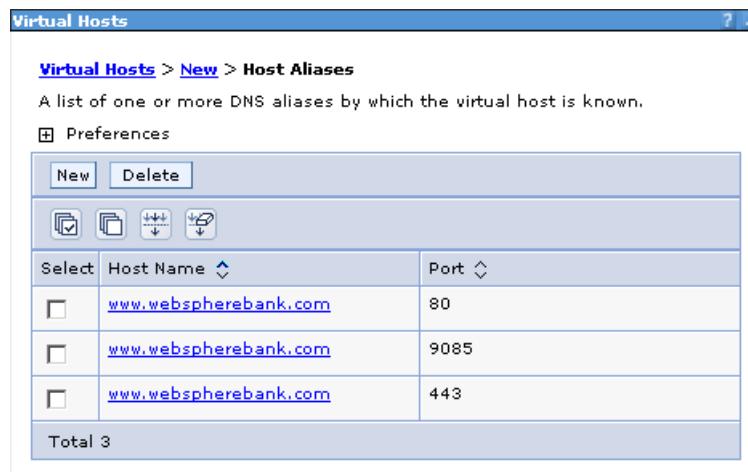


Figure 16-3 WebSphere Bank virtual host aliases

7. Click **OK**.
8. Save the configuration.

16.1.5 Creating the virtual host for IBM HTTP Server and Apache

Now that we have defined a webspheredbank_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 16-2.

Example 16-2 Using VirtualHost

```
<VirtualHost www.webspherebank.com:80>
    ServerAdmin webmaster@webspherebank.com
    ServerName www.webspherebank.com
    DocumentRoot "C:\WebSphere\IBMHTTPServer\htdocs\webspherebank"
    ErrorLog logs/webspherebank_error.log
    TransferLog logs/webspherebank_access.log
</VirtualHost>
```

If you want to have multiple virtual hosts for the same IP address, you must use the `NameVirtualHost` directive. See Example 16-3.

Example 16-3 Using the NameVirtualHost and VirtualHost directives

```
NameVirtualHost 9.42.171.190:80
```

```
<VirtualHost itso_server:80>
    ServerAdmin webmaster@itso_server.com
    ServerName itso_server
    DocumentRoot "C:\WebSphere\IBMHTTPServer\htdocs\itso_server"
    ErrorLog logs/itso_server_error.log
    TransferLog logs/itso_server_access.log
</VirtualHost>

<VirtualHost www.webspherebank.com:80>
    ServerAdmin webmaster@webspherebank.com
    ServerName www.webspherebank.com
    DocumentRoot "C:\WebSphere\IBMHTTPServer\htdocs\webspherebank"
    ErrorLog logs/webspherebank_error.log
    TransferLog logs/webspherebank_access.log
</VirtualHost>
```

The `www.webspherebank.com` and the `itso_server` hosts have the same IP address, 9.42.171.190. 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):

```
9.42.171.190 www.webspherebank.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.webspherebank.com`.

As you can see in Example 16-3, each virtual host has a different document root. Make sure that the directory you specify exists before you start the HTTP server. We recommend that you place an index.html file at the document root stating which virtual host is being called. This lets you easily test 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.webspherebank.com> or http://itso_server returns different HTML pages.

Configuring listening ports

If you want to use a different port rather than the default port 80, use the Listen directive to tell the Web server to listen to requests on that specific port:

1. Make a backup of the `<ihc_home>\conf\httpd.conf` file.
1. Start your favorite editor and edit this file.
2. Search for the line containing the `# Listen` string, and edit it as follows to make it listen on, for example, port 90:
`Listen 90`
3. Save the httpd.conf file.
4. Restart the HTTP server.

16.1.6 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 895.

The WebSphere Bank sample application uses a relational database, via entity beans, to store account information. To access this database, define a data source, which you then associate with the entity beans. The WebSphere Bank sample application, just like the other sample applications, is configured for Cloudscape by default. In Chapter 15, “Packaging applications” on page 847,

however, we also added a backend and EJB deployed code for the WebSphere Bank application to run against a DB2 database. We will now create the DB2 JDBC provider, datasource and JAAS authentication alias required to run against DB2.

For detailed information about JDBC providers and data sources, refer to 7.2, “JDBC resources” on page 321.

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 setup. 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 16-4.

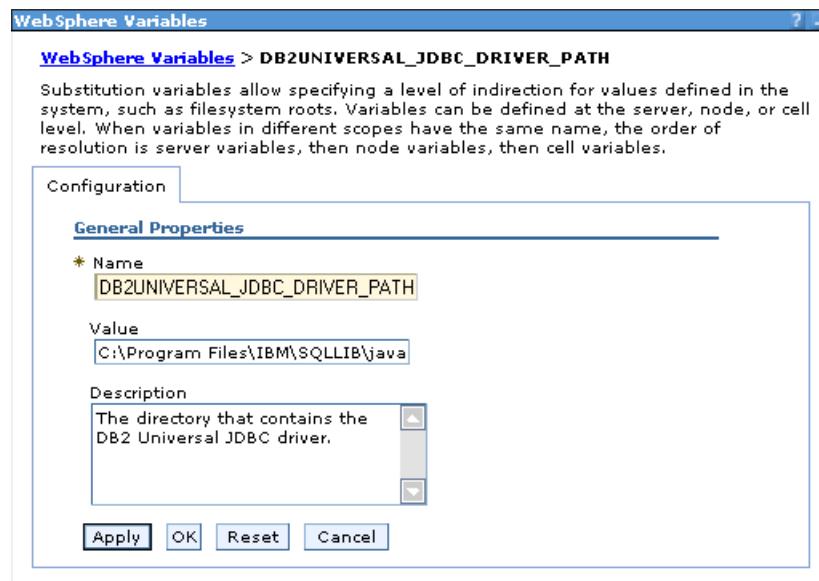


Figure 16-4 Configuring DB2 Driver Path

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** → **Global Security**. Expand the JAAS Configuration 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 16-5.
 - **Alias**
Enter the name of the security information alias, such as webspheredbank.
 - **User ID**
Enter a user ID with the proper authority to access the database.
 - **Password**
Enter the password for the user ID.



Figure 16-5 Creating WebSphere Bank JAAS authentication alias

3. Click **OK**.

Creating the WebSphere Bank 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 standalone server environment, it is sufficient to create the data source at the server level. Otherwise, define it at the node 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 16-6.

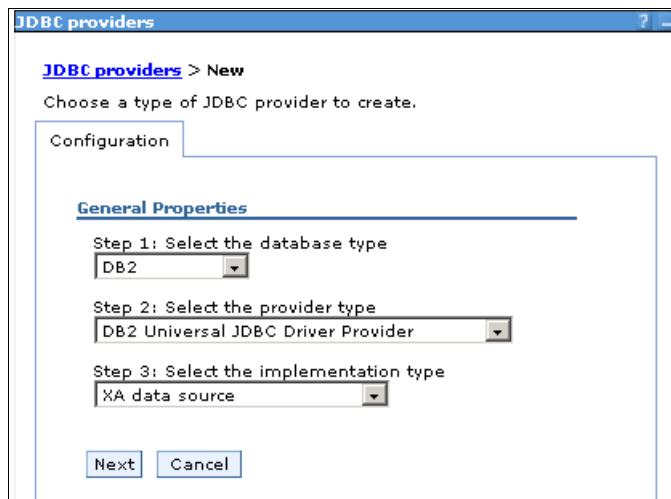


Figure 16-6 Creating a DB2 JDBC provider

- Database type: **DB2**
 - Provider type: **DB2 Universal JDBC Driver Provider**
 - Implementation type: **XA data source**
5. Specify the following information for the provider, as shown in Figure 16-7.

Note: We used the DB2 XA-capable JDBC Driver for the WebSphere Bank 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 MyJDBCXDriverXA.

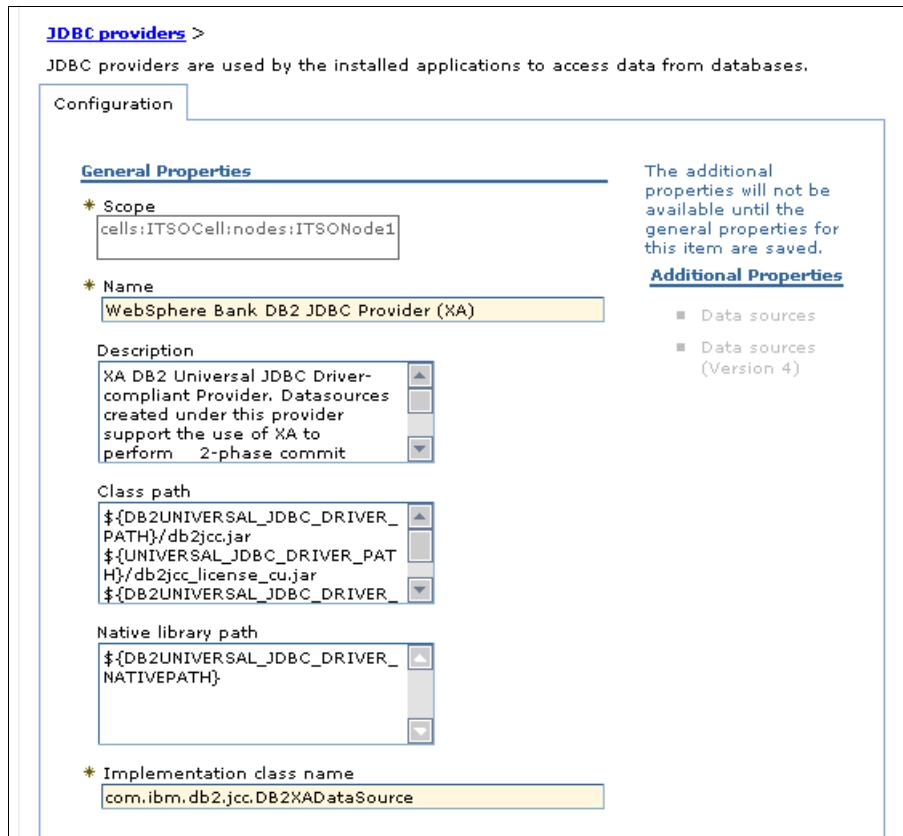


Figure 16-7 WebSphere Bank JDBC provider properties

– **Name**

Enter the JDBC provider name, such as WebSphere Bank DB2 JDBC Provider (XA).

– **Classpath**

Enter the path to the JAR/ZIP file that contains the JDBC provider code, for example, \${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar. We encourage you to use variables rather than hard-code the path to the JDBC drivers.

– **Native library path**

Enter the path to any native libraries required by the JDBC driver.

– **Implementation class**

Enter the Java class that provides the JDBC service, such as com.ibm.db2.jcc.DB2ConnectionPoolDataSource.

The classpath, native library path and the implementation class fields are all automatically completed for known JDBC providers.

6. Click **OK**.

Creating the WebSphere Bank data source

The next step is to create the data source for the WebSphere Bank DB2 database. To create a data source, do the following:

1. Select **Resources** → **JDBC Providers**.
2. Select the **WebSphere Bank DB2 JDBC Provider (XA)** and select **Data Sources** under Additional Properties.
3. Click **New** to add the new data source. See Figure 16-8 on page 923.

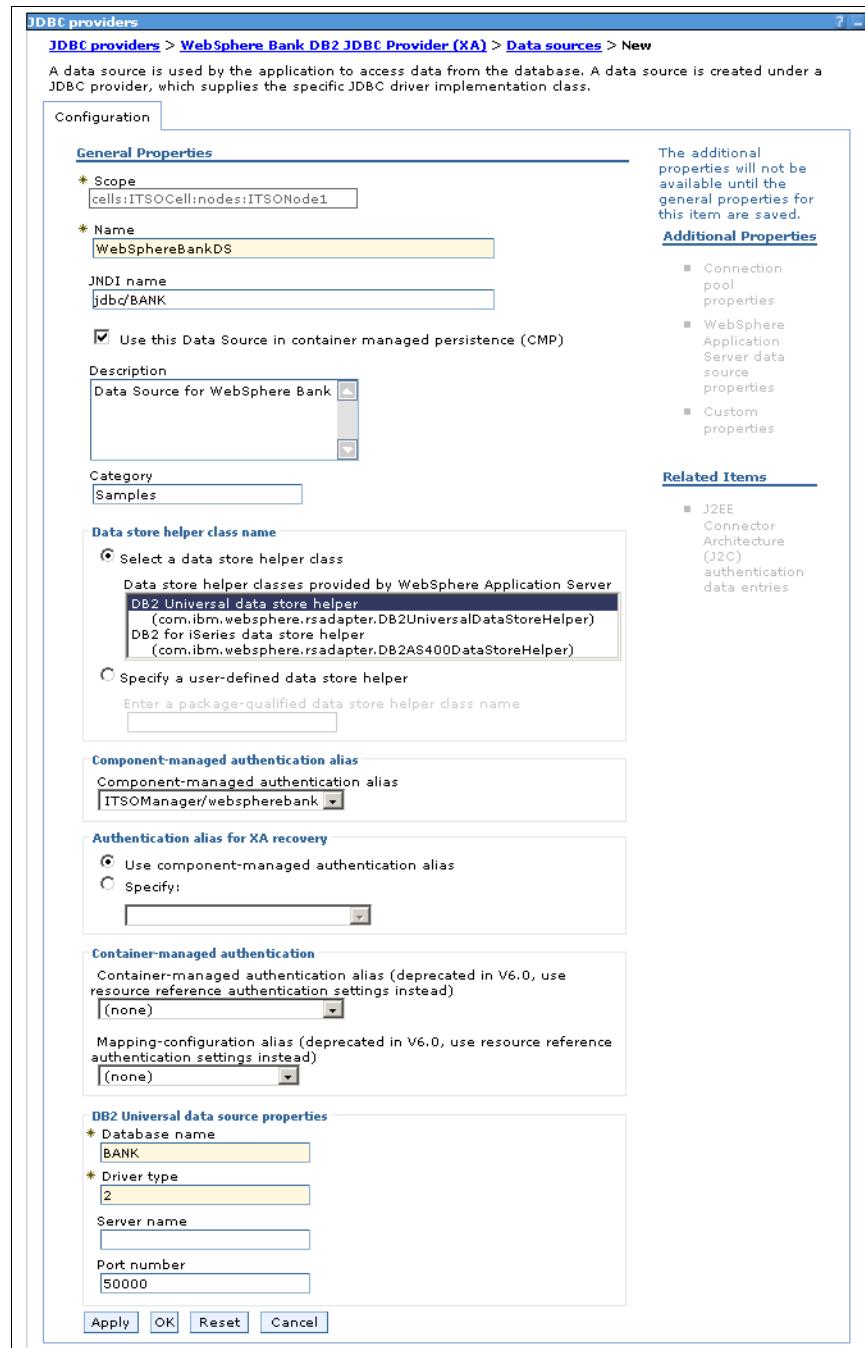


Figure 16-8 WebSphere Bank data source properties

– Name

Enter the data source name, which must be unique in the administrative domain or cell. It is recommended that you use a value indicating the name of the database this data source is targeting, such as “WebSphereBankDS”.

– 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 WebSphere Bank, set this field to jdbc/BANK. This value can be changed at any time after the data source has been created.

– Use this Data Source in container-managed persistence (CMP)

Check this option to indicate that the WebSphere Bank data source is used for persisting the application entity beans.

– Datasource helper class name

Select the name of the class used by the Relational Resource Adapter at runtime to implement the functionality of a specific database driver. Select **DB2 Universal data store helper**.

– Component-managed authentication alias

Enter the J2C alias used when connecting to this data source by selecting the authentication alias created previously, <cell name>/webspherebank.

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) is deprecated in WebSphere Application Server V6.

– DB2 Universal data source properties

Enter the database name, driver type, server name and port number required to access the database. In our environment the database is called BANK and is located on the same physical machine as our application server so we use the Type 2 driver.

For DB2, specify at least the database name and the driver type. For Oracle, provide the driver type (thin/oci8), and the database URL, or server name and listening port.

4. Click **OK**.
5. Save the configuration.

6. Test the connection by selecting the data source and clicking the **Test Connection** button.

16.1.7 Configuring the messaging resources

The messaging function of the WebSphere Bank consists of the following resources that need to be defined. The details of defining each are covered in the messaging chapters of this book.

- ▶ J2C authentication alias entry

Create a J2C authentication alias entry with the properties in Table 16-1.

Table 16-1 J2C authentication alias properties

Property	Value
Alias	<cell>/samples
User ID	samples
Password	s1amples

- ▶ Service integration bus

The service integration bus is the default messaging provider. By default, there is no bus defined. Define a bus with the the following property in Table 16-2.

Table 16-2 Service integration bus property

Property	Value
Name	SamplesBus

Let the rest of the properties default.

To see an example of how to create the service integration bus, see 11.8.2, “Creating a bus” on page 658.

Next, add the application server to the bus as a member. See 11.8.3, “Adding a bus member using a default data store” on page 660 for information and an example.

In this case, we are using the default data store in Cloudscape, so no action is needed to create the database.

- ▶ Service integration bus queue

Define a queue to the bus with the following properties in Table 16-3 on page 926.

Table 16-3 Service integration bus queues properties

Property	Value
Destination type	Queue
Identifier	BankJSConnFactory
Bus member	<node>:<server>

See 11.8.5, “Creating a queue destination” on page 666 for information and an example of how to do this.

► JMS provider:

WebSphere Bank uses the default messaging provider. This provider is preconfigured at all scope levels. For information about viewing and working with the default messaging provider, see 10.5.1, “Managing the default messaging JMS provider” on page 514.

► JMS connection factory

A JMS connection factory is required to create a connection to the service integration bus. The connection factory has the following properties in Table 16-4.

Table 16-4 JMS connection factory properties

Property	Value
Scope	Node
Connection factory name	BankJMSConnFactory
JNDI name	jms/BankJMSConnFactory
Bus name	SamplesBus
Component managed authentication alias	<cell>/samples

Let the rest of the properties default.

For information about JMS connection factory properties see “JMS connection factory properties” on page 527. To see an example of how to create this connection factory, see “JMS connection factory configuration” on page 534.

► JMS queue

A JMS destination is used to address a message to a specific destination on the JMS provider. For this application, the destination is a JMS queue and will reference a queue on the bus. Enter the properties in Table 16-5.

Table 16-5 JMS queue properties

Property	Value
Scope	Node
Connection factory name	BankJMSQueue
JNDI name	jms/BankJMSQueue
Bus queue name	BankJSQueue

Let the rest of the properties default.

For information about JMS queue properties see “JMS destination properties” on page 535. To see an example of how to create this JMS queue, see “JMS queue configuration” on page 539.

► Activation specification

The activation specification is used to associate a destination with a message-driven bean. In this case, a message-driven bean associated with this activation specification would be invoked when a message arrives on the BankJSQueue destination on the service integration bus, which is mapped to jms/BankJMSQueue. Enter the following properties in Table 16-6.

Table 16-6 Activation specification properties

Property	Value
Scope	Node
Name	BankActivationSpec
JNDI name	eis/BankActivationSpec
Destination type	queue
Destination queue name	jms/BankJMSQueue
Bus queue name	SamplesBus

Let the rest of the properties default.

For information about activation specification properties see “JMS activation specification properties” on page 545. To see an example of how to create this activation specification, see “JMS activation specification configuration” on page 549.

16.2 Generating deployment code

At some point, you will need to generate the deployment code for the Enterprise JavaBeans. You can do this in Rational Application Developer, in the Application Server Toolkit, from the command line, or at deployment time using the install panels in the WebSphere administrative console. The deployed code should match the version of the runtime target. If you plan to deploy an EAR file which was developed in WebSphere Studio version 5, we recommend you regenerate the deployed code to match WebSphere Application Server V6.

For information about how to generate the deployed code using the Application Server Toolkit, see “Creating a new database mapping and schema” on page 867.

16.2.1 Using EJBDeploy command line tool

You can generate the EJB deployed code using the EJBDeploy command-line tool. The syntax of the EJBDeploy command is shown in Example 16-4.

Example 16-4 EJBDeploy syntax

EJBDeploy (v6.0, dms0444.09)

```
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
                        CLOUDSCAPE_V5
                        DB2UDBOS390_V7  DB2UDBOS390_V8
                        DB2UDBISERIES   DB2UDBISERIES_V52 DB2UDBISERIES_V53
                        INFORMIX_V73    INFORMIX_V93      INFORMIX_V94
                        MSSQLSERVER_V7  MSSQLSERVER_2000
                        ORACLE_V9I      ORACLE_V10G      SYBASE_V1200      SYBASE_V1250
  -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
  -35                  Use the WebSphere 3.5 top-down mapping rules
  -40                  Use the WebSphere 4.0 top-down mapping rules
  -target               Set the server target, to one of:
```

	WAS510	WAS502	WAS501	WAS500
-trace	Trace progress of the deploy tool			
-sqlj	Use SQLJ instead of JDBC			
-OCCCColumn	Add a column for collision detection for WebSphere 6.0 or later release			

For a complete description of the EJBDeploy command and its parameters, see the Information Center. Search for *ejbdeploy*.

Example 16-5 shows a sample EJBDeploy run using the WebSphereBank EAR file.

Example 16-5 EJBDeploy sample run

```
C:\WebSphere\AppServer\bin>ejbdeploy c:\WebSphereBankEnhanced.ear  
c:\temp c:\WebSphereBankEnhanced_Deployed.ear -dbvendor DB2UDB_V82

Starting workbench.  
Creating the project.  
Building: /BankCMRQLEJB_EJB  
Deploying jar BankCMRQLEJB_EJB  
Validating  
[*Information]  
imported_classes/com/ibm/websphere/samples/bank/ejb/Customer.class(Method:  
getAccountsList(), Class: com.ibm.websphere.samples.bank.ejb.Customer):  
CHKJ2500I: java.util.Collection must be serializable at runtime (EJB 2.0:  
10.6.9).  
[*Warning] ejbModule/META-INF/ejb-jar.xml(BankCMRQLEJB): CHKJ2874W: Migrate  
this EJB module's default datasource binding to a default CMP Connection  
Factory binding.  
Generating deployment code  
Refreshing: /BankCMRQLEJB_EJB/ejbModule.  
Building: /BankCMRQLEJB_EJB  
Invoking RMIC.  
Generating DDL  
Generating DDL  
Building: /WsaEJBDeployUtility  
Building: /WsaEJBDeployUtility  
Writing output file  
Shutting down workbench.  
EJBDeploy complete.  
0 Errors, 1 Warnings, 1 Informational Messages
```

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.

16.3 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 **WebSphereBank.ear** file. Then click **Next**.

From the install panels 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 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.

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 16-7 on page 931. 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 16-7 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 one, 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 webspherebank_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 16.3.1, “Using a bindings file” on page 936.

4. Click **Next**.
5. The WebSphere Bank EAR file includes a was.policy file which grants the application access to certain resources, when WebSphere security is enabled. The installation panel warns you about this as a potential security exposure and asks you to decide whether to continue installing the application or abort. For the WebSphere Bank sample we are fine, so click **Continue**.

The rest of the wizard is divided into steps.

6. 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, whether you want to generate EJB deployment code, or enable reloadable classloaders for Web modules.

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 panel 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, avoiding the first user that hits 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.

Important: The WebSphere Bank application uses JMS resources, which are not supported by the Enhanced EAR. It is important that you deselect the Process embedded configuration check box.

Click **Next**.

7. 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 WebSphere Bank EAR file. In the Clusters and Servers box, select the **WebSphereBankServer**. Then click **Apply**. This assigns all modules to the WebSphereBankServer application server. If you deploy to a cluster, select the cluster instead of the single application server.

See Figure 16-9 on page 933.

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.

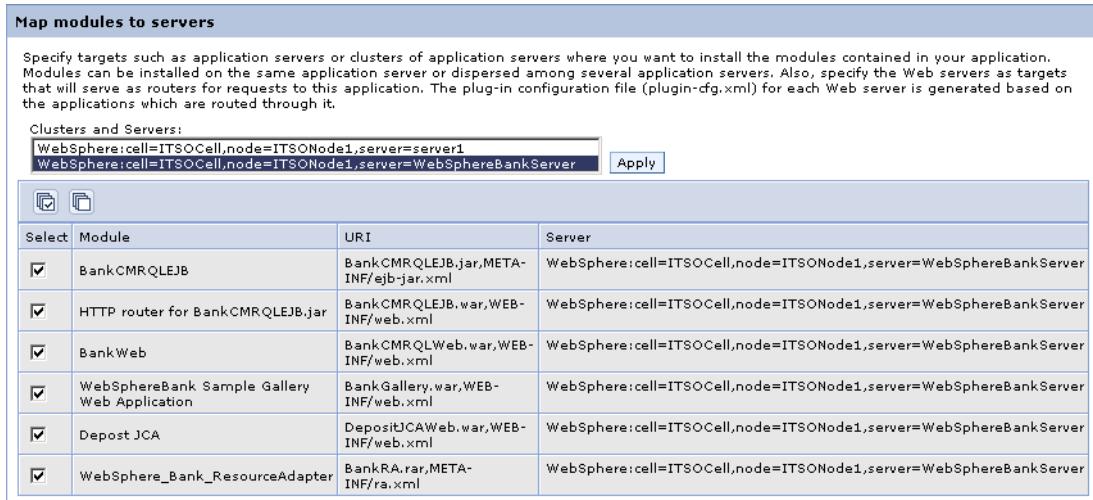


Figure 16-9 Mapping modules to application servers

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 12 if you like. See 17 on page 935.

8. 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 DB2EXPRESS_V82 because this is the version we are using.

9. Step 4: Provide listener bindings for message-driven beans

In this step you can select listener ports or ActivationSpecs to which to bind your message-driven beans. The WebSphere Bank application requires the bindings shown in Table 16-8.

Table 16-8 Listener bindings for message-driven beans

Message-driven bean	ActivationSpec
BankListener	eis/BankActivationSpec
BankATMListener	eis/com.ibm.websphere.samples.bank.adapter.BankMessageListener

We configured this in 15.3.3, “Binding the message-driven bean to an ActivationSpec” on page 864 so the defaults shown should be correct.

Click **Next**.

10. Step 5: Provide JNDI names for beans

Use this window to bind the enterprise beans in your application or module to a JNDI name. In 15.3.1, “Defining EJB JNDI names” on page 861, we defined these values in Table 16-9, so the defaults shown should be correct.

Table 16-9 Webbank enterprise bean JNDI names

EJB Name	JNDI Name
Sender session bean	ejb/Bank/Sender
Transfer session bean	ejb/Bank/Transfer
Account entity bean	ejb/Bank/Account
Customer entity bean	ejb/Bank/Customer

Click **Next**.

11. Step 6: Bind message destination references to administered objects

In this step, bind the message destination references to their JNDI names. Again, the values configured in the WebSphere Bank are fine.

Click **Next**.

12. Step 7: Map JCA resource references to resources

Each J2C object must be bound to a JNDI name. For the WebSphere Bank application, the ActivationSpec com.ibm.websphere.samples.bank.adapter.BankMessageListener should therefore be bound to a JNDI name.

In 15.3.3, “Binding the message-driven bean to an ActivationSpec” on page 864 we configured these values, which are fine.

Click **Next**.

13. Step 8: Provide 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 15.3.4, “Defining data sources for entity beans” on page 865 we defined the JNDI name for the EJBs in the BankCMRQLEJB module as jdbc/Bank. You see this in the window.

Click **Next**.

14. 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 7, (12 on page 934). We do not need to do anything here. Click **Next**.

15. Step 10: 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 15.3.2, “Binding EJB and resource references” on page 862.

Click **Next**.

16. Step 11: Map resource references to resources

Each resource reference defined in the application must be mapped to the corresponding resource. The EJB module BankCMRQLEJB has a resource reference to jms/BankJMSConnFactory, which is shown here.

Click **Next**.

At this step we now get an Application Resource Warning. What this tells us is that the resource we just referenced, jms/BankJMSConnFactory, is not defined for the specific application server to which we are installing our application. This is fine because the resource is defined at the node level instead.

Click **Continue**.

17. Step 12: Map virtual hosts for Web modules

For each Web module, select the virtual host we created for the application (webspheredbank_host).

Click **Next**.

18. Step 13: Map security roles to users and groups

Because the WebSphere Bank EAR file contains security roles, we need to map them to users and groups in our target environment. However, because we have not enabled J2EE security (global 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**.

19. 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**.

20. Step 15: Summary

The Summary window gives an overview of application deployment settings. If those settings are fine, click **Finish** to deploy the application.

21. Save the configuration.

If you are working in a distributed server environment, make sure you synchronize the changes with the nodes so they application is propagated to the target application server (s).

Deployment is now complete. 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.

You can now start the WebSphereBankServer application server and try the application by invoking <http://www.webspherebank.com/WebSphereBank>. Make sure that the host name is one of the three names you have added to the webspherebank_host definition. See 16.1.5, “Creating the virtual host for IBM HTTP Server and Apache” on page 915.

16.3.1 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 16-6 is an example showing a bindings file used to change the JNDI name of an EJB:

Example 16-6 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>
</dfltbndngs>
```

For more information about creating specific bindings file, refer to the Information Center.

16.4 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 runtime 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 16-10.

Table 16-10 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

Available functions	J2EE client	Thin client	Pluggable client	Applet client	ActiveX client
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
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

Available functions	J2EE client	Thin client	Pluggable client	Applet client	ActiveX client
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 panels 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.

16.4.1 Defining application client bindings

The WebSphere Bank sample application 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:

```
cell1/nodes/ITSONode1/servers/WebSphereBankServer/ejb/Bank/Customer
```

If you have created a cluster of application servers, use:

```
cell1/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.

For more information about JNDI and naming, please refer to Chapter 13, “WebSphere naming implementation” on page 769.

16.4.2 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 -profileName option is not required for running in a single profile environment or in an Application Clients installation. The default is default_profile. |
| -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 .ear file containing the client application.
The -CC properties are for use by the Application Client runtime. 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 <true false> to display additional informational messages. The default is false.
-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: <code>-CCproviderURL=corbaloc:iiop:myserver.mycompany.com:9810,:mybackupserver.mycompany.com:2809</code>

-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 16-7:

Example 16-7 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 6.0
J2EE Application Client Tool
Copyright IBM Corp., 1997-2004
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.

16.5 Updating applications

WebSphere Application Server V6 introduces new features that allow applications to be updated and restarted at a more fine-grained level than in earlier versions. In V6 it is possible to update only parts of an application or module and only the necessary parts are restarted. In V6 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.

Also introduced in V6 is enhanced support for managing applications in a cluster for continuous availability. The new 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.

16.5.1 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 panel, Select the **Full 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 panels and make any changes necessary. For information about the panels see “Deploying the application” on page 930. On the Summary panel click **Finish**.
5. When the application has been updated in the Master repository, select the **Save To Master Configuration** link.
6. If you are working 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.

16.5.2 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 panel, select the **Single module** option.
3. In the **Relative path to module** field, enter the relative path to the module to replace in the EAR file. For example, if you were to replace the HelloWeb.war Web module in the HelloApp EAR file, you would enter HelloWeb.war. If you enter a path that does not exist in the EAR file, the module 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 (e.g. HelloWeb) in the **Context root** field.
6. Click **Next**.
7. Proceed through the remaining panels and make any changes necessary. For information about the panels see “Deploying the application” on page 930. On the Summary panel 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 To Master Configuration** link.
9. If you are working 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.
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.

16.5.3 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 panel, select the **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 HelloWeb.war Web module, you would enter

`HelloWeb.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 panel, click **OK**.
6. When the application has been updated in the Master repository, select the **Save To Master Configuration** link.
7. If you are working 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.

16.5.4 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 in the list as shown in Figure 16-10 and click **OK**.



Figure 16-10 Removing a file from an application

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. In the Related items list on the right-hand part of the screen, click the link for Web modules, EJB modules or Connector modules, depending on the type of module the file to remove is.
3. Select the module and click the **Remove File** button.
4. In the Remove file dialog box, select the file to be removed in the list as shown Figure 16-10 and click **OK**.
5. Save the configuration.

16.5.5 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 V6. The uploaded file is analyzed and the necessary actions to update the application is 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 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 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 16-11.

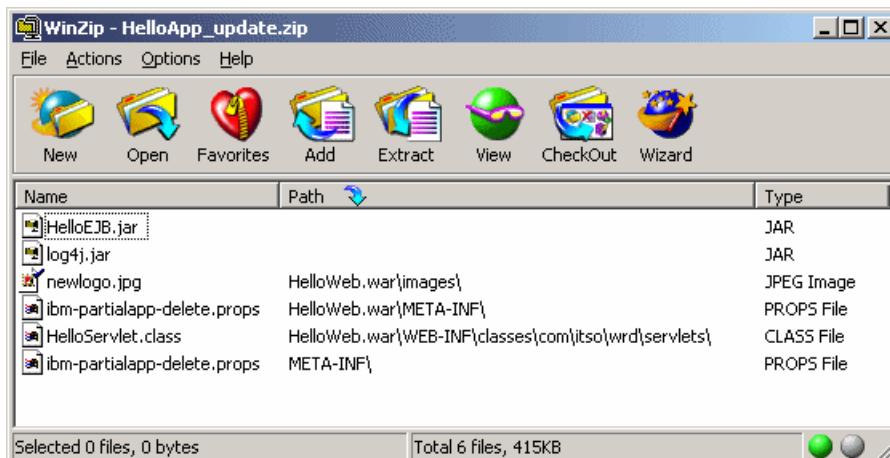


Figure 16-11 HelloApp_update.zip compressed file

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 HelloWeb_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 panel, 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 panel, 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.

16.5.6 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 panel, 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 **Full 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 panels and make any changes necessary. For information about the panels see “Deploying the application” on page 930. On the Summary panel click **Finish**.
5. When the application has been updated in the master repository, the status window shown in Figure 16-12 on page 952 is displayed:

Application HelloApp installed successfully.

If you want to do a rolling update of the application on the cluster(s) on which it is installed, then click Rollout Update. A rolling update will save all changes made in this session to the master configuration, then synchronize and recycle the cluster members on each node, one node at a time.

Rollout Update

To start the application, first save changes to the master configuration.

The application may not be immediately available while being started on all servers.

Save to Master Configuration

To work with installed applications, click the "Manage Applications" button.

Manage Applications

Figure 16-12 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 panel, 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 16-13 on page 953 is displayed in the status window.

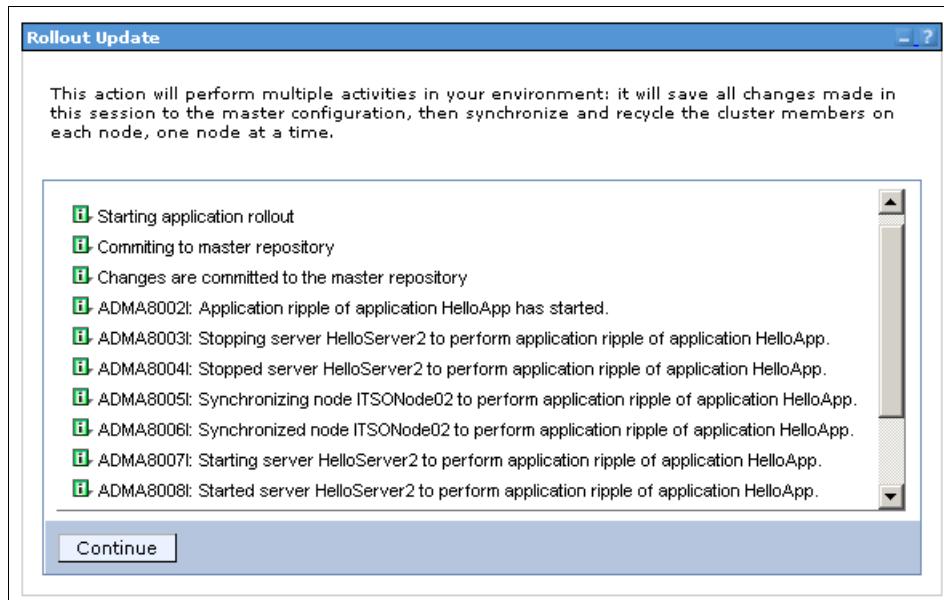


Figure 16-13 Rolling out an application

For each node, the cluster members are stopped, the application is distributed and they are restarted. When the rollout has completed, 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 rollout 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.

16.5.7 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 runtime 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 109. 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, it is recommended 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:

- ▶ Enable class reloading and Reloading interval for the enterprise application.

In order for application files to be reloaded automatically after an update, Enable class reloading must be enabled and the Reloading interval must be greater than 0.

Select **Applications** → **Enterprise Applications**. Double-click the application to open the configuration page. These settings can be found in the General Properties section.

- ▶ `reloadingEnabled` in the Web module extensions

A Web container reloads a Web module only when this setting is enabled.

This setting is found in the `ibm-web-ext.xmi` file and can be updated using the Application Server Toolkit. Open the Web module deployment descriptor and select the **Extensions** tab.

- ▶ Application class loader policy for the application server

The application 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**. Double-click the server to open the configuration page. The setting is found in the General Properties section.

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.



WebSphere Rapid Deployment

WebSphere Rapid Deployment is a collection of tools and technologies introduced in WebSphere Application Server V6 that make application development and deployment easier than ever before.

WebSphere Rapid Deployment consists of the following:

- ▶ Support for annotation-based (Xdoclet) programming
- ▶ Fine-grained application updates
- ▶ Rapid deployment tools

In this chapter we briefly discuss the concept of annotation-based programming and then show how to use the rapid deployment tools for the following two project types (modes):

- ▶ Free-form development projects
- ▶ Automatic application installation projects

Using free-form development, you can quickly develop small J2EE applications without using an integrated development environment. Automatic application installation is an easy way to manage your applications by letting WebSphere automatically install, update and uninstall your EAR files.

17.1 Annotation-based programming

Annotation-based programming is a method of speeding up application development by reducing the number of artifacts you need to develop and manage on your own. By adding metadata tags to your Java source code, the WebSphere Rapid Deployment tools can automatically create and manage the artifacts required to build J2EE compliant modules and applications.

For example, consider a stateless session EJB. With annotation-based programming, you simply create a single Java source file containing the bean implementation logic, add a few tags indicating that you want to deploy this class as an EJB, and indicate which methods should be made public on the EJB interface. Using this single artifact, WebSphere Rapid Deployment can create:

- ▶ The home and remote interface classes
- ▶ A stateless session implementation wrapper class
- ▶ The EJB deployment descriptor (ejb-jar.xml)
- ▶ The WebSphere-specific binding data
- ▶ All of the remaining artifacts required to produce a J2EE application

All you have to deal with is a single Java artifact, the other required artifacts are generated, as shown in Figure 17-1 on page 959.

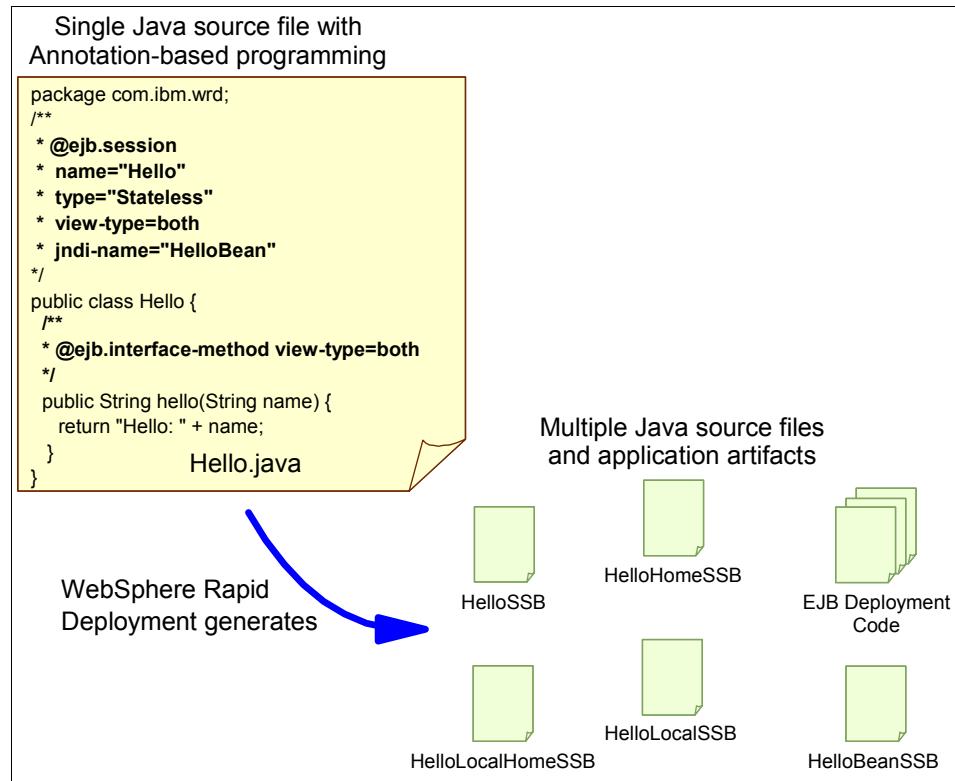


Figure 17-1 Artifacts generated by annotation-based class

WebSphere Rapid Deployment supports numerous tags to help you build your applications. These tags map directly to known J2EE artifacts and deployment descriptor elements. Specifically, WebSphere Rapid Deployment supports tags for the following artifact types and generation targets:

- ▶ EJBs
- ▶ Servlets
- ▶ Java classes
- ▶ Web services

WebSphere Rapid Deployment supports annotations using Javadoc-style comments in the package, class, field, or method declarations. The syntax uses XDoclet syntax, where it exists. Currently, WebSphere Rapid Deployment supports the syntax used by XDoclet for J2EE 1.3.

We do not go into the details of annotation-based programming in this book, but we do use it when developing the example application for showing the free-form development projects. For detailed information about annotation-based

programming, see the WebSphere Information Center. Search for *annotation* and resources on the Internet, for example:

<http://xdoclet.sourceforge.net/xdoclet/index.html>

17.2 Rapid deployment tools

Using the rapid deployment tools part of WebSphere Rapid Deployment you can:

- ▶ Package J2EE artifacts quickly into an EAR file.
- ▶ Deploy and test J2EE modules and full applications quickly on a server.
- ▶ Create a new J2EE application quickly without the overhead of using an integrated development environment (IDE).

For example, you can place full J2EE applications (EAR files), application modules (WAR files, EJB JAR files), or application artifacts (Java source files, Java class files, images, JSPs etc.) 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 perform the steps necessary to produce a running application on an application server.

There are two ways to configure the monitored directory, each performing separate and distinct tasks. You can specify the monitored directory as a free-form project or an automatic application installation project.

With the free-form approach, you can place in a single project directory the individual parts of your application, such as Java source files that represent servlets or enterprise beans, static resources, XML files and other supported application artifacts. The rapid deployment tools use your artifacts to place them automatically in the appropriate J2EE project structure, generate any additional required artifacts to construct a J2EE compliant application and deploy that application on a target server.

The automatic application installation project allows you to quickly and easily install, update, and uninstall J2EE applications on a server. If you place EAR files in the project directory, they are deployed automatically to the server. If you delete EAR files from the project directory, the application is uninstalled from the server. If you place a new copy of the same EAR file in the project directory, the application is reinstalled. If you place WAR or EJB JAR files in the automatic application installation project, the rapid deployment tool generates the necessary EAR wrapper, then publishes that EAR file on the server. For RAR files, a wrapper is not created. The standalone RAR files are published to the server.

The advantage of using a free-form project is that you do not need to know how to package your application artifacts into a J2EE application. The free-form project does the packaging part for you. The free-form project is suitable when you just want to test something quickly, perhaps write a servlet that performs a task.

An automatic application installation project simplifies management of applications and relieves you of the burden of going through the installation panels in the WebSphere administrative console, or developing wsadmin scripts to automate your application deployment.

Figure 17-2 describes the two modes.

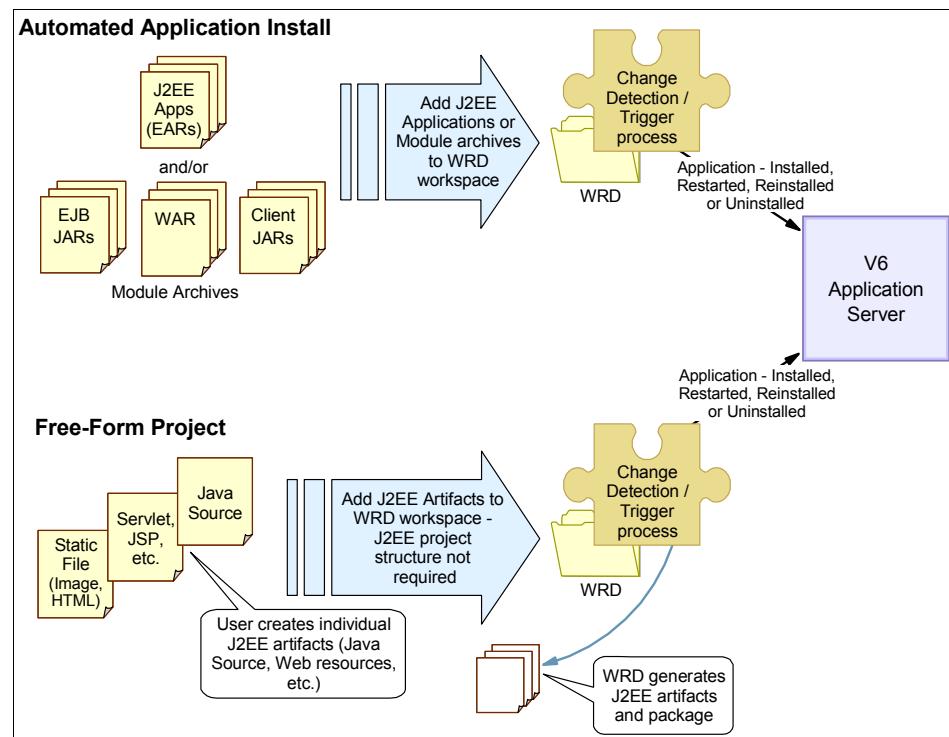


Figure 17-2 WebSphere Rapid Deployment modes

Note: The rapid deployment tools can be configured to deploy applications either onto a local or remote WebSphere Application Server.

17.3 Using rapid deployment commands

The rapid deployment tools are managed by two commands, **wrd-config** and **wrd**. Wrd-config is used to configure a workspace directory for a rapid deployment session, either free-form or automatic application installation mode. The wrd command is then used to tell the rapid deployment tools to start monitoring the directory and act on the changes you introduce to it.

17.3.1 wrd-config command

The **wrd-config** command is used to configure a workspace for either free-form development or automatic application installation mode.

The syntax for the **wrd-config** command is:

```
wrd-config.bat(sh) -project <project_name> -style <freeform| autoappinstall>  
[<optional parameters>]
```

Note: Before running the **wrd-config** command, you must set up the WORKSPACE environment variable to point to the workspace directory your rapid deployment session will use. This is done by executing:

- ▶ Windows: **set WORKSPACE=<workspace_root>**
- ▶ Linux: **export WORKSPACE=<workspace_root>**

Table 17-1describes the mandatory parameters.

Table 17-1 wrd-config mandatory parameters

Parameter	Description
-project <"project_name">	This is the name of the rapid deployment project that you want to create. The project name needs to be unique in your workspace.
-style <"freeform" "autoappinstall">	The deployment approach - either the free-form or automatic application installation project is used as the deployment style.

The free-form project allows you to create or drop in your J2EE artifacts (such as servlet, EJB or general Java source or class files, JSPs, static Web content, and all other generic files) into the free-form project. These resources are then automatically compiled (if necessary), placed in the appropriate location in the J2EE project structure and deployed on an application server. Optionally, the rapid deployment tools can also export an EAR file to a target directory.

The automatic application installation project creates a single project that listens for fully composed EAR or module files. If EAR files are placed inside this project, the EAR file is deployed automatically to the server. If the EAR file is deleted, then that application is uninstalled from the server. If you place WAR or EJB jar files in an automatic application installation project, the rapid deployment tool generates the necessary EAR wrapper, then publishes that EAR file on the server. For RAR files, a wrapper is not created. The standalone RAR files are published to the server.

The **wrd-config** command also takes the optional parameters described in Table 17-2. Not all parameters are applicable to the automatic application installation mode, however.

Table 17-2 wrd-config optional parameters

Parameter	Description
-rebuild	Clean and rebuildsthe contents of the rapid deployment project. Use this to start a new session,
-configure	Open an interactive console session to modify any available parameters.
-runtime <“was51” “was60”>	Targeting your runtime will configure which JRE library to use, based on the runtime location, and configure the project's classpath to contain the WebSphere runtime libraries. Specify as an identifier if either WebSphere Application Server v5.1 (“was51”) or WebSphere Application Server v6.0 (“was60”) is used as the target run-time environment for rapid deployment processing. If this parameter is not specified, the default setting is WebSphere Application Server v6.0. This parameter is only used with -runtimePath parameter.
-runtimePath <“was_home”>	Was_home is the directory where WebSphere Application Server is installed (for example, c:\WebSphere\AppServer).
-j2eeVersion <“1.3” “1.4”>	Specify either J2EE version 1.3 or 1.4 is used for development and deployment.
-configPath “x:\filename.xml”	Specify the destination file path for the configuration file, where x is the temporary directory. This file persists the configuration data to an XML file, which can later be used to drive other rapid deployment configuration sessions. If this path is not specified, the default location is in the root of the rapid deployment workspace.

Parameter	Description
-configData "x:\filename.xml"	The path of an existing XML configuration file that is used to drive the configuration session. If this path is not specified, the default location of the XML configuration file is in the root of the rapid deployment workspace with the naming convention projectName_headlessconfig.xml, where projectName is the value specified in the -project parameter.
-listStyles	List the available deployment styles (AutoAppInstall and FreeForm) and their descriptions.
-listServers	List the available runtime server targets.
-properties	List the properties for a given deployment project. This parameter is only used with -project.
-buildMode	Specify to disable all console outputs. This is useful for silent builds.
-usage	Display the optional and required parameters for this command.

Note: The runtimePath is not used for targeting the server to publish against, but, rather, specifying the WebSphere libraries you want your application to compile against. If the runtime and runtimePath are not specified, the default runtime environment used is WebSphere Application Server v6.0.

When running **wrd-config**, an interactive Parameter Configuration Settings dialog window is displayed as shown in Example 17-1:

Example 17-1 Parameter Configuration Setting dialog window

Parameter Configuration Settings

Press ENTER to accept defaults
The * symbol denotes required input

Enter the server name* (server1) :
Enter the server JMX host name* (localhost) :
Enter the server JMX port number* (8880) :
Enter a path for containing the exported EAR (--) :
Enter your server username (--) :
Enter your server password (--) :

Table 17-3 describes the settings that should be configured in the dialog window. The order listed is the way they are presented by the dialog window.

Table 17-3 wrd-config Parameter Configuration Settings

Option	Description
serverName	Use the name of the server process you want to publish your application. For example, server1. For WebSphere Application Server Network Deployment, the server name is in the form <cell_name>/<node_name>/<server_name>. Note: WebSphere Rapid Deployment cannot publish to a cluster.
serverJMXHost	Use the host name of the machine containing the server to which you want to make a connection. For example, localhost. For WebSphere Application Server Network Deployment, type the host name of the Network Deployment Manager.
serverJMXPort	Use the server administrative port number, also known as the SOAP connector port. This port is used for making JMX connections with the server. For example, 8880. In a distributed server environment, use the SOAP port number of the deployment manager. the default is 8879.
earExportPath	This is an optional field to specify the directory location of the output EAR file that is created by rapid deployment tools and contains the generated classes required for deployment. For example: c:\temp Note: When you specify the directory for the output EAR file and then run the wrd-config command, any existing output EAR file of the same name will be overwritten without warning. In addition, the EAR file gets overwritten as additional J2EE artifacts are added to the free-form project to reflect the new changes.
username	(Optional) If security is enabled, specify the user name for current active authentication settings defined in the server configuration.
password	(Optional) If security is enabled, specify the password for current active authentication settings defined in the server configuration.

Example 17-2 shows some examples of how to use the wrd-config command.

Example 17-2 wrd-config examples

To create a new free-form project called MyProject:

```
wrd-config.bat -project "MyProject" -style "freeform"
```

To create a new automatic application installation project called MyProject:

```
wrd-config.bat -project "MyProject" -style "autoappinstall"
```

To persist configuration data to an XML file:

```
WRD-config.bat -project "MyProject" -style "freeform" -runtime "was60"  
-runtimePath "c:\WebSphere\AppServer" -configPath  
"c:\configData\myHeadlessConfig.xml"
```

To create a new rapid deployment project using an existing XML configuration file:

```
wrd-config.bat -configData "c:\configData\myHeadlessConfig.xml"
```

To clean and rebuild an existing rapid deployment project:

```
wrd-config.bat -project "MyProject" -rebuild
```

To modify available deployment parameters:

```
wrd-config.bat -project "MyProject" -configure
```

To query properties of an existing rapid deployment project:

```
wrd-config.bat -project "MyProject" -properties
```

To query available rapid deployment styles and run-time targets:

```
wrd-config.bat -listStyles -listServers
```

Note: A `projectName_headlessconfig.xml` file is generated in the root of your rapid deployment workspace. Use this file later to configure a project with the same configuration, without having to be prompted again for these parameters. Use the optional `-configData` parameter to complete this task.

17.3.2 wrd command

The `wrd` command is used to configure the rapid deployment tools to start monitoring a project directory, configured by the `wrd-config` command, for changes.

The syntax for the `wrd` command is:

```
wrd.bat(sh) [<optional parameters>]
```

The wrd command has no mandatory parameters, but takes the optional parameters described in Table 17-4.

Table 17-4 wrd optional parameters

Parameter	Description
-monitor	Enable console feedback from the rapid deployment tool.
-project <"project_name">	Use the name of the rapid deployment project that you want to target to run in batch mode. This -project parameter is only used in conjunction with -batch
-batch	Enable batch mode on a specified target project. Batch mode runs a full build on a rapid deployment workspace and then shuts down the process.
-usage	Display the optional parameters for this command.

Note: Before starting a WebSphere Rapid Deployment session, make sure the application server that will host the application is started. Otherwise, you will see the following error message in the console output:

```
[04:31:32 PM] Publishing HelloApp to server_510658053  
[04:31:32 PM] ERROR! Failed to make connection to WebSphere Application Server.
```

17.4 Free-form projects

Free-form projects provide a simple way to develop small J2EE applications quickly, without having to use a full-blown integrated development environment. However, when developing larger applications, you almost always want the benefits of an integrated development environment, such as Rational Application Developer for WebSphere software to give you features like application modeling, version control, debugging, profiling, and so forth.

Also, if your application cannot conform to the assumptions and defaults made by the rapid deployment tools, you need to use an integrated development environment to give you total control over how your application is packaged.

When using free-form projects, the rapid deployment tools make certain assumptions about the J2EE application being created. The J2EE project structure created will hold one EJB module, one EJB client JAR, one Web module and one utility JAR, all encapsulated in one enterprise application (EAR file). There will be, at most one of each type of module. A module is created only

when necessary. So, unless you have introduced any EJB code in your project, no EJB module is created. Table 17-5 shows the naming conventions used for naming the modules.

Table 17-5 Naming conventions for free-form project contents

Naming conventions for the project folder	Project type
<i>project_name</i>	The free-form project
<i>project_nameApp</i>	A single enterprise application project
<i>project_nameEJB</i>	A single EJB module project
<i>project_nameEJBCClient</i>	A single EJB client jar project
<i>project_nameWeb</i>	A single Web module project
<i>project_nameUtility</i>	A single utility Java project

Project_name is the name given the project when setting up the environment and is the directory being monitored.

The rapid deployment tools will configure and update the manifest files and project references to maintain the following default settings:

- ▶ The EJB module project will have reference and visibility to the EJB client jar project.
- ▶ The Web module project will have reference and visibility to the EJB client jar project.
- ▶ The Web module project, EJB module project, and EJB client jar project will have reference and visibility to the utility project.
- ▶ The enterprise application project will contain the Web module project, EJB module project, and EJB client jar project.
- ▶ The utility Java project is added as a utility JAR inside the enterprise application project.

Each of the projects, excluding the enterprise application project, contains an imported_classes directory. The imported_classes folder is set as an exported entry on the classpath of its containing project. As a result, the directory and any contents within it, is visible to any projects referencing that project.

The initial configuration of the free-form project creates the following subdirectories in the project directory:

- ▶ The gen folder is a repository of non-Java artifacts that are generated by the rapid deployment tools.

- ▶ The `src` folder is a subdirectory of the `gen` folder and is a repository of Java artifacts generated by the rapid deployment tools. For example, if you drop an annotated session EJB, the remote, home and local interfaces are generated in `src` directory.
- ▶ The `bin` directory contains the class files that were compiled for any Java source files dropped into the free-form project.

When Java source files are dropped into a free-form project, they are compiled into this project and the generated class files are copied into the target J2EE project's `imported_classes` folder. The source files are not copied into any of the J2EE project structures. The deployed application contains only class files.

If a classification of a resource changes from one artifact type to another that results in changes to its target J2EE module project, the old mapped locations are purged. For example, if `Hello.java` implemented a servlet, its class files map to the Web module project. If the content of `Hello.java` changes to implement an enterprise bean, its class files now map to the EJB module project. The dangling class in the Web project is removed.

Any resources that cannot be classified by the rapid deployment tools are mapped to the utility Java project, and their folder structure is preserved as created and dropped into the free-form project. For example, if the following resource exists in the free-form project, `/MyFreeForm/data/myproperties.props`, this file is mapped to `/MyFreeFormUtility/data/myproperties.props`, preserving the `data` folder. In addition, if the class files cannot be classified by the rapid deployment tools as other than a class file, the tool maps these files to the `imported classes` folder in the utility Java project.

When removing artifacts from the free-form project, the rapid deployment tools remove the artifacts from their mapped J2EE project structure location and, if applicable, also remove the deployment descriptor entries. However, the rapid deployment tools keep any project folders created.

The application generated can be published and deployed either on a local or remote WebSphere Application Server. When publishing to a local server, the server runs the application using the resources (.class files) within the workspace. As a result, the server itself does not maintain the resources for the application on the local server. To have the server maintain the resources within the application, use remote publishing, even though you are using a local server. This way, you can later modify server-specific configurations using the WebSphere administrative console.

To use remote publishing on a local server to keep resources in an application, do the following:

1. Open the headless.props file in the workspace/.plugins/.metadata/com.ibm.ws.rapiddeploy.core directory, where workspace is the directory where your rapid deployment project resides.
2. Set the parameter value of TREAT_SERVER_AS_REMOTE to true. This enables the WebSphere Administrative console to work with the application.

If the free-form project is configured to a local server (on the same machine running the rapid deployment session), the application synchronizes with the server after every change. If the free-form project is configured to a remote server, changes are batched together in one minute intervals to limit file transfers over the network for every change. A publish is triggered every minute, only when a change occurs.

The rapid deployment tools only manage the J2EE part of your application, not any additional resources (such as JDBC Providers, DataSources, JMS configuration, and so on) your application might also need. You must manually configure these resources using the WebSphere administrative console before your application can run.

17.5 Free-form development example

For the purpose of showing how to develop an application using a free-form project we will create a very very simple J2EE application. When completed, the application will contain the following artifacts:

- ▶ A stateless session EJB, called HelloEJB
- ▶ A servlet, called HelloServlet
- ▶ A JSP, called hello.jsp
- ▶ A GIF image, called logo.gif in a directory called images

The EJB has only a remote interface and only a single method, getGreeting, returning a String value. When invoked, HelloServlet looks up HelloEJB and calls its getGreeting method. It then forwards to the JSP, which displays the message along with the GIF image.

The EJB and servlet are developed using annotation-based programming to reduce the number of artifacts that need to be managed.

The WebSphere Rapid Deployment session will be set up to automatically produce an EAR file with the contents of our application.

The scenario we will use contains the following steps:

1. Create a workspace directory

2. Configure a free-form project in the workspace.
3. Launch a WebSphere Rapid Deployment session. This step configures the rapid deployment tools to monitor the project directory for changes.
4. Copy the EJB source code into the project directory.
5. Copy the servlet source code into the project directory.
6. Copy the JSP source code into the project directory.
7. Copy the GIF image into the project directory.
8. Verify that the code has been compiled, packaged it into a J2EE application and is installed on the application server.
9. Test the application.
10. Examine the EAR file exported.

17.5.1 Setting up the environment for free-form development

Before starting out with our free-form example project, we need to set up the environment. Do the following:

1. Open a command prompt. This will be used for running the rapid deployment tools in headless, nongraphical, mode. All output from the tools will be written to this window. We, therefore, also refer to it as the console window.
2. Create a workspace directory, for example c:\wrd, by typing:

```
mkdir c:\wrd
```

The workspace directory holds the projects with which you work.

3. Set the WORKSPACE environment variable by typing:

```
set WORKSPACE=c:\wrd
```

Tip: To list all environment variables currently set, type **set** in Windows, or **export -n** in Linux and press Enter.

4. Configure the rapid deployment tools to create a free-form project called **Hello** in the workspace directory by typing:

```
cd <profile_home>\bin  
wrd-config.bat -project "Hello" -style "freeform"
```

<profile_home> is the root directory of the application server profile you are using.

This brings up the interactive Parameter Configuration Settings dialog window. Enter the information as requested in Example 17-3 on page 972.

Example 17-3 Parameter Configuration Settings

```
-----  
Parameter Configuration Settings  
-----  
Press ENTER to accept defaults  
The * symbol denotes required input  
  
Enter the server name* ( server1 ) : ITSOCell/ITSONode1/server1  
Enter the server JMX host name* ( localhost ) :  
Enter the server JMX port number* ( 8880 ) : 8879  
Enter a path for containing the exported EAR ( -- ) : c:\temp  
Enter your server username ( -- ) :  
Enter your server password ( -- ) :  
  
Configuring the workspace...  
Building the workspace...  
WebSphere Rapid Deployment configuration completed.
```

When creating this example, we were running in a distributed server environment and, therefore, used the JMX port (8879) for our deployment manager. Our target application server is called server1 and exists on node ITSONode1 in the cell called ITSOCell.

If you were running on a standalone server, you would use the JMX port number, default 8880, for the standalone application server, default server1. In our environment, the deployment manager and the application server are located on the same physical machine so we used the default JMX hostname localhost. Also, we did not have WebSphere global security enabled, so we did not need to supply a username or password.

We entered c:\temp as the exported EAR target directory. This means that the WebSphere Rapid Deployment tools will produce an EAR file for us in this directory in addition to deploying the application to our server.

Note: Make sure the application server that the application will be published on (ITSOCell/ITSONode1/server1 in our example) is started before proceeding to the next step.

When the **wrd-config** command has finished, the directory structure shown in Figure 17-3 on page 973 is created in the workspace directory.

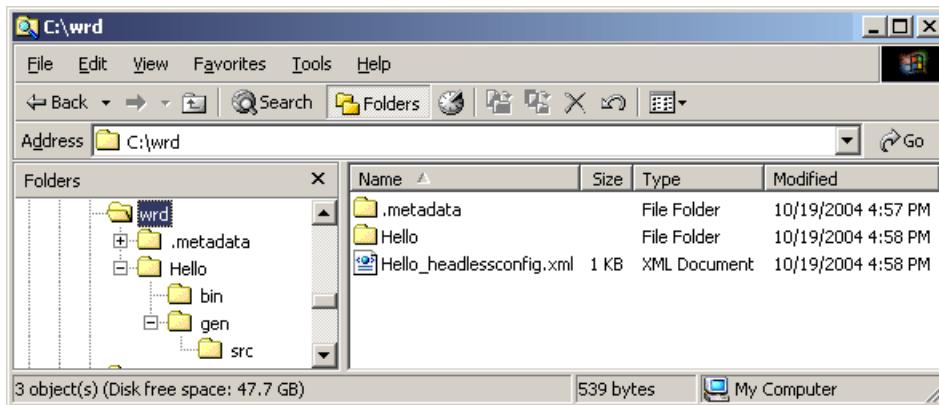


Figure 17-3 Initial free-form project structure

The Hello directory is the project directory and will be monitored by the tools. You can have multiple projects in a workspace as long as their names are unique. In the root of the workspace, the Hello_headlessconfig.xml file has been created. This file holds the configuration for our project.

5. Launch the WebSphere Rapid Deployment session by typing:

```
wrd.bat -monitor
```

This tells the rapid deployment tools to start monitoring the project directory for changes and display messages on the console.

Note: The `wrd` command itself is used to tell the rapid deployment tools to start monitoring the project directory and the optional `-monitor` switch enables the console output messages.

The `wrd` command displays the messages in Example 17-4 in the console window:

Example 17-4 Launching WebSphere Rapid Deployment

```
Launching WebSphere Rapid Deployment. Please wait...
Starting Workbench...
```

```
WebSphere Rapid Deployment ready for e-business...
```

```
Type 'q', 'quit', or 'exit' to shut down WebSphere Rapid Deployment processes.
```

To terminate a WebSphere Rapid Deployment session, type **Q**, **quit** or **exit** and press **Enter** in the console window. This stops the monitoring of the project directory. You can continue working where you stopped by simply running the **wrd** command again.

17.5.2 Adding application source code

When the WebSphere Rapid Deployment session is configured and monitoring of the project directory is activated, we can introduce the Java files and the other artifacts that make up our application. We will use annotated Java source code and let the rapid deployment compile it for us, but we could also have copied .class files directly to the project directory.

Adding EJB source code

To add EJB source code, do the following:

1. Create a directory called `com\itso\wrd\ejbs` in the `c:\wrd\Hello` project directory. The reason for doing this is that the Java source file must be placed in the correct directory structure, matching the class package name.
2. Using a text editor, add the EJB code shown in Example 17-5 and save the file as `HelloEJBBean.java` in the `c:\wrd\Hello\com\itso\wrd\ejbs` directory:

Example 17-5 Source code for HelloEJBBean.java

```
package com.itso.wrd.ejbs;
import javax.ejb.*;
/**
 * Bean implementation class for Session Bean: HelloEJB
 *
 * @ejb.bean
 * name="HelloEJB"
 * type="Stateless"
 * jndi-name="ejb/com/itso/wrd/ejbs/HelloEJBHome"
 * view-type="remote"
 * transaction-type="Container"
 *
 * @ejb.home
 * remote-class="com.itso.wrd.ejbs.HelloEJBHome"
 *
 * @ejb.interface
 * remote-class="com.itso.wrd.ejbs.HelloEJB"
 *
 */
public class HelloEJBBean implements SessionBean {
    private SessionContext mySessionCtx;
    /**
     * @ejb.interface-method view-type=remote
```

```
/*
public String getGreeting() {
    return "Hello WebSphere!";
}
public SessionContext getSessionContext() {
    return mySessionCtx;
}
public void setSessionContext(SessionContext ctx) {
    mySessionCtx = ctx;
}
public void ejbCreate() throws CreateException { }
public void ejbActivate() {}
public void ejbPassivate() {}
public void ejbRemove() {}
}
```

After a few seconds, the introduced change is picked up by the rapid deployment tools and the console window displays the contents of Example 17-6 on page 976.

Example 17-6 Adding the HelloEJBBean.java

```
[11:14:20 AM]  [/Hello/com/itso/wrd/ejbs>HelloEJBBean.java] Added
[11:14:20 AM]  [/Hello/bin/com/itso/wrd/ejbs>HelloEJB.class] copied to project
[HelloEJBClient]
[11:14:20 AM]  [/Hello/bin/com/itso/wrd/ejbs>HelloEJBBean.class] copied to
project [HelloEJB]
[11:14:20 AM]  [/Hello/bin/com/itso/wrd/ejbs>HelloEJBHome.class] copied to
project [HelloEJBClient]
[11:14:28 AM]  Publishing IBMUTC to server_1617843630
[11:14:30 AM]  Installing New Application: IBMUTC
...
...
[11:15:45 AM]  Publishing HelloApp to server_1617843630
[11:15:46 AM]  Installing New Application: HelloApp
...
...
[11:16:05 AM]  ADMA5013I: Application HelloApp installed successfully.
[11:16:41 AM]  Installation Completed Sucessfully: HelloApp
[11:16:41 AM]  Starting Application: HelloApp
[11:16:42 AM]  Application Started Sucessfully: HelloApp
[11:16:43 AM]  Publishing HelloApp to server_1617843630
[11:16:43 AM]  Updating Application.
[11:16:44 AM]  Update is not required.
```

When the annotated EJB source code is introduced in the monitored directory, the necessary EJB files such as home and remote interfaces are generated and compiled. They are packaged in an EJB module which is, in turn, packaged into an application called HelloApp. HelloApp is deployed onto the application server.

If the rapid deployment does not find the Universal Test Client (UTC) on the application server, it will be automatically installed. The Universal Test Client can be used for testing for example EJBs and can be accessed at

<http://localhost:9080/UTC>.

If you open the WebSphere administrative console and select **Applications** → **Enterprise Applications** → **HelloApp** you can verify the application configuration as shown in Figure 17-4 on page 977.

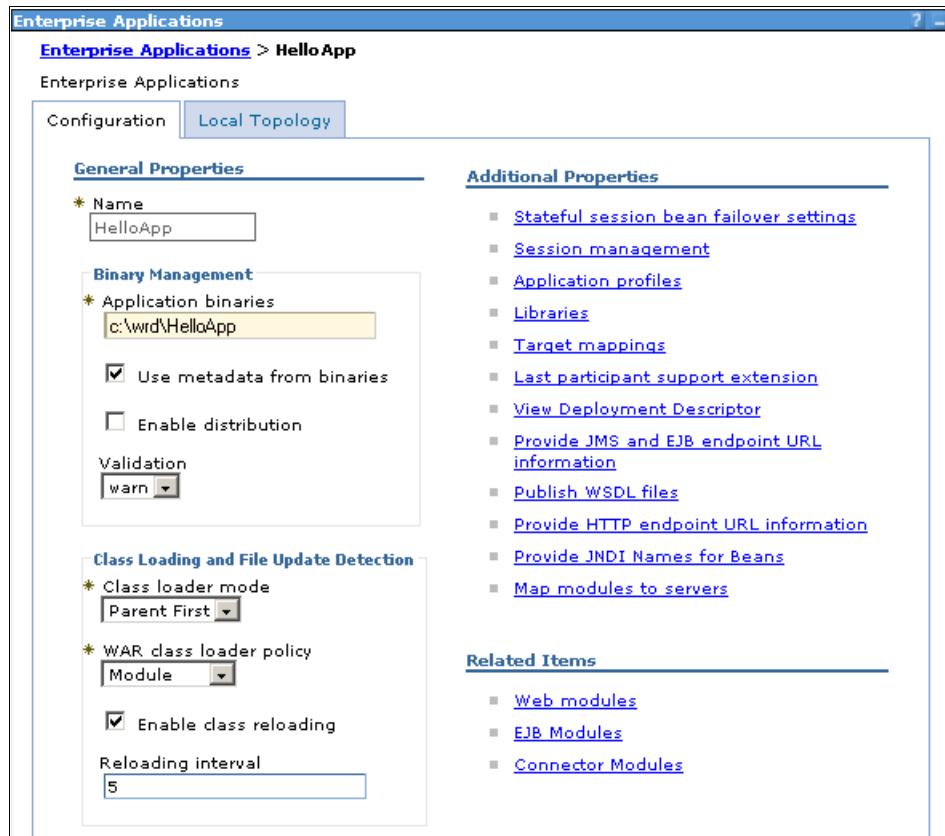


Figure 17-4 HelloApp deployed to WebSphere Application Server

The application resources (binaries) are pointing at the project directory, c:\wrd\HelloApp, indicating that the application server is executing the application using the resources directly from the project directory. This is possible because we are publishing to a local server. If we would have run against a remote server, the rapid deployment tools would have packaged the HelloApp.ear file, distributed it to the remote server and deployed it there. Running against a local server is, of course, faster.

If you click the EJB Modules, you see that the application has one EJB module, HelloEJB.jar, defined. But there are no Web modules yet.

Note: It is important that the Java source files you add to the project directory are created in the correct directory structure, matching the class files package name. If you simply drop the HelloEJBBean.java class into the c:\wrd\Hello directory instead of the c:\wrd\Hello\com\its0\wrd\ejbs directory, you see a compilation error such as:

```
[05:15:43 PM] 'The declared package does not match the expected package ' in  
resource 'HelloEJBBean.java' on line number 1
```

If you are dropping whole EJB modules (ejb.jar files) into a free-form project, you need to generate the deployed code yourself. The EJBDeploy tools do not automatically run on these already packaged ejb.jar files. For information about the EJBDeploy tool, see 16.2.1, “Using EJBDeploy command line tool” on page 928.

If you drop an entity bean, by default the rapid deployment tools will generate a bean-managed persistence (BMP) entry in the deployment descriptor. If you want a container-managed persistence (CMP) bean, specify this using an annotated source file.

A naming convention must be followed for the Java source, compiled class, or annotated source files for each enterprise bean. For example, if an enterprise bean implementation class is named HelloEJBBean.java, follow this convention for any bean interfaces that are defined:

- ▶ The remote interface class must be named HelloEJB.java.
- ▶ The home interface class must be named HelloEJBHome.java.
- ▶ The local interface class must be named HelloEJBLocal.java.
- ▶ The local home interface class must be named HelloEJBLocalHome.java.

The primary key class for entity beans can be named anything because the rapid deployment tools logically locate the correct class name by introspecting either the remote home or local home interface.

These rules do not imply that both remote and local view types must be created. These are just naming conventions when creating source for a set of bean interfaces.

A new EJB entry to the deployment descriptor is only added when a minimum set of resources are available in the free-form project. For example, if the bean class is placed in the free-form project, its compiled class maps to the EJB project, but no deployment descriptor entry is created until the rapid deployment tools locate either a set of remote view type classes or local view type classes. For entity beans, a primary key class is also required. If the minimum resources are removed, the bean descriptor entry is removed.

Adding servlet source code

Using the same technique as for the EJB, now add the servlet code by doing the following:

1. Create a directory called com\itso\wrd\servlets in the c:\wrd\Hello project directory.
2. Add the servlet code shown in Example 17-7 and save the file as HelloServlet.java in the c:\wrd\Hello\com\itso\wrd\servlets directory:

Example 17-7 Source code for HelloServlet.java

```
package com.itso.wrd.servlets;
import java.io.*;
import javax.naming.*;
import javax.rmi.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.itso.wrd.ejbs.*;
/**
 * Servlet implementation class for Servlet: HelloServlet
 *
 * @web.servlet name="HelloServlet" display-name="HelloServlet"
 *
 * @web.servlet-mapping url-pattern="/HelloServlet"
 *
 */
public class HelloServlet extends javax.servlet.http.HttpServlet
    implements javax.servlet.Servlet {

    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        String greeting = null;
        try {
            InitialContext ctx = new InitialContext();
            Object objectHome =
                ctx.lookup("ejb/com/itso/wrd/ejbs>HelloEJBHome");
            HelloEJBHome helloEJBHome = (HelloEJBHome)
                PortableRemoteObject.narrow(objectHome, HelloEJBHome.class);
            HelloEJB helloEJB = helloEJBHome.create();
            greeting = helloEJB.getGreeting();
        } catch (Exception e) {
            e.printStackTrace(System.out);
        }
        req.setAttribute("greeting", greeting);
        RequestDispatcher dispatch = req.getRequestDispatcher("/hello.jsp");
        dispatch.forward(req, resp);
    }

    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
```

```
        throws ServletException, IOException {
    doGet(req, resp);
}
```

After a few seconds, rapid deployment tools have picked up the new changes and Example 17-8 is shown in the console window.

Example 17-8 HelloServlet.java added

```
[11:23:21 AM]  [/Hello/com/itso/wrd/servlets>HelloServlet.java] Added
[11:23:21 AM]  [/Hello/bin/com/itso/wrd/servlets>HelloServlet.class] copied to
project [HelloWeb]
[11:23:21 AM] Publishing HelloApp to server_1617843630
[11:23:22 AM] Updating Application.
[11:23:23 AM] Servlet added to web.xml: HelloServlet
[11:23:23 AM] Servlet mapping added. URL is: [HelloWeb/HelloServlet]
...
...
```

Using the WebSphere administrative console, you can now see that there is a Web module called HelloWeb.war added to the HelloApp application. The context root for the Web module is the same as its name, HelloWeb. Also, a servlet mapping HelloWeb/HelloServlet is added to the web.xml file.

Note: In our very simple example application, our EJB publishes only a remote interface and our servlet looks it up in the JNDI namespace directly without using an EJB reference. We could have used the @web.ejb-ref tag if we wanted to use an EJB reference instead of looking it up directly. To access an EJB's local interface, you must use the @web.ejb-local-ref tag.

Adding JSP source code

Using the same procedure as before, add the JSP code shown in Example 17-9 and save the file as hello.jsp in the c:\wrd\Hello directory:

Example 17-9 Source code for hello.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<TITLE>Greeting</TITLE>
</HEAD>
<BODY>
<H1><%= request.getAttribute("greeting") %></H1>
```

```
<IMG border="0" src="images/logo.gif" width="211" height="44">
</BODY>
</HTML>
```

When the JSP is added, Example 17-10 on page 981 is shown in the console window.:

Example 17-10 Adding hello.jsp

```
[11:24:58 AM] [/Hello/hello.jsp] Added
[11:24:58 AM] [/Hello/hello.jsp] copied to project [HelloWeb]
[11:24:58 AM] JSP entry added to web.xml: hello
[11:24:58 AM] Servlet mapping added. URL is: [HelloWeb/hello]
...
...
```

As you can see in the console window, the web.xml file is updated also with a servlet mapping for the JSP.

Adding a GIF image

Because we want rapid deployment tools to maintain the directory structure for our static resources, we first need to create a directory to hold our image before we can add it to the project.

1. Create a directory called images in c:\wrd\Hello
2. Copy logo.gif to c:\wrd\Hello\images directory

The following is shown in the console window:

```
[11:26:05 AM] [/Hello/images/logo.gif] Added
[11:26:05 AM] [/Hello/images/logo.gif] copied to project [HelloWeb]
```

17.5.3 Terminating the WebSphere Rapid Deployment session

We have added the source code and resources that make up our example application and can now terminate the WebSphere Rapid Deployment session. Type **Q** and press **Enter** in the console window to do this. This stops monitoring of the project directory.

17.5.4 Verifying results

We are now ready to see what has been generated for us and to test our simple application.

The application we have developed can now be accessed at

<http://localhost:9080>HelloWeb/HelloServlet>.

The output is shown in Figure 17-5 on page 982.



Figure 17-5 *HelloServlet* output

Note: The rapid deployment tools do not automatically regenerate the HTTP server plug-in configuration file. You must manually regenerate the plug-in configuration file.

In the c:\temp directory, the rapid deployment tools have also produced an EAR file called HelloApp.ear. The contents of this EAR file are shown in Figure 17-6. As you can see, it uses the default names for the project modules.

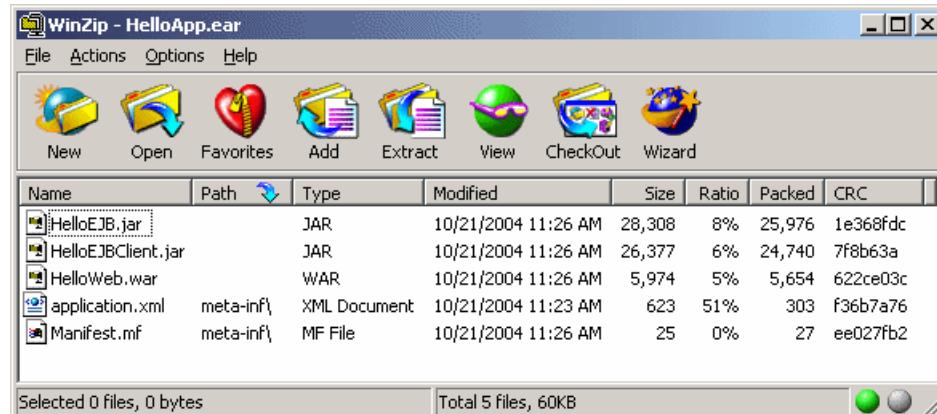


Figure 17-6 *HelloApp.ear* file produced

17.6 Automatic application installation projects

Automatic application installation mode allows you to set up a directory and have it monitored for either fully composed EAR files or application modules such as WAR files, EJB JAR files, or standalone resource adapter archive (RAR) files.

If you drop EAR files in the monitored directory, the EAR file is automatically deployed to the server. If you delete the EAR file from the monitored directory, that application is uninstalled from the server. If you drop an updated EAR file in the monitored directory, the application is updated on the server. If you drop WAR or EJB JAR files in the monitored directory, the rapid deployment tools generate the necessary EAR wrapper and then publishes that EAR file on the server. For RAR files, a wrapper is not created. The standalone RAR files are published to the server.

The automatic application installation mode only manages your applications, not any resources, such as data sources, they might require. You need to create and manage these resources yourself by using the WebSphere administrative console or wsadmin scripts, before the application can run.

Note: If you drop an EJB module, or an EAR file containing an EJB module, which does not contain EJB deployed code, the rapid deployment tools generate the deployment code with the default EJBDploy settings, for example, the backend option will be set to DB2UDB_V81. If you want to set a different backend option, run the EJB deployment tools (ejbdeploy.bat) on the EAR file before dropping it into the automatic application installation project. See 16.2.1, “Using EJBDploy command line tool” on page 928.

To use the automatic application installation mode you must:

1. Create a workspace directory.
2. Configure an autoappinstal1 project in the workspace directory.
3. Configure the rapid deployment tools to monitor the directory for changes.
4. Drop EAR files or application modules into the directory.

We will show how to use the automatic application installation mode using a simple example.

17.7 Automatic application installation example

For the purpose of showing how the automatic application installation mode works, we will use the Hello application we developed in the “Free-form development example” on page 970. In this example, the rapid deployment tools

produced an EAR file, HelloApp.ear, for us. We will use this EAR file in the following scenario:

- ▶ Install application
- ▶ Update the application
- ▶ Uninstall the application

17.7.1 Setting up an automatic application installation session

Using the automatic application installation mode requires you to set up a directory on your system and configure WebSphere to monitor the directory for changes. Do the following:

1. Open a command prompt. This will be used for running the rapid deployment tools in headless, non-graphical, mode. All output from the tools will be written to this window. We therefore also refer to it as the console window).
2. Create a workspace directory, for example c:\wrd, by typing:

```
mkdir c:\wrd
```

3. Set the WORKSPACE environment variable by typing:

```
set WORKSPACE=c:\wrd
```

Tip: To list all environment variables currently set, you can type **set** in Windows or **export -n** in Linux and press Enter.

4. Configure the WebSphere Rapid Deployment tools to create an automatic application installation project called `install_server1` by typing:

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

```
wrd-config.bat -project "install_server1" -style "autoappinstall"  
<profile_home> is the root directory of the application server profile.
```

This brings up the interactive Parameter Configuration Settings dialog window, Example 17-11. Enter the information as requested:

Example 17-11 Parameter Configuration Settings dialog window

```
-----  
Parameter Configuration Settings  
-----
```

```
Press ENTER to accept defaults  
The * symbol denotes required input
```

```
Enter the server name* ( server1 ) : ITSOCell/ITSONode1/server1  
Enter the server JMX host name* ( localhost ) :  
Enter the server JMX port number* ( 8880 ) : 8879
```

```
Enter your server username ( -- ) :  
Enter your server password ( -- ) :  
  
Configuring the workspace...  
Building the workspace...  
WebSphere Rapid Deployment configuration completed.
```

When creating this example, we were running in a distributed server environment and used the JMX port (8879) for our deployment manager. Our target application server is called server1 and exists on node ITSONode1 in the cell called ITSOCell1.

If you were running on a standalone server, you would use the JMX port number (default 8880) for the standalone application server (default server1). In our environment, the deployment manager and the application server were located on the same physical machine, so we used the default JMX hostname localhost. Also, we did not have WebSphere global security enabled. As a result, we did not need to supply a username or password.

When the **wrd-config** command has finished, a directory called `install_server1` is created in the `c:\wrd` directory. The `install_server1` directory is the project directory and is the directory that will be monitored by the tools. In the root of the workspace the `install_server1_headlessconfig.xml` file has been created. This file holds the configuration for our project.

Note: We named our automatic application installation project `install_server1` because the rapid deployment tools then create the `c:\wrd\install_server1` monitored directory for us and this name explains well what the directory is used for. We could have given the project any name, however, because it is only used to name the monitored directory.

The real name of the application server that the application is installed on is configured in the Parameter Configuration Settings dialog window (ITSOCell/ITSONode1/server1).

5. Launch the WebSphere Rapid Deployment session by typing:

```
wrd.bat -monitor
```

This tells the rapid deployment tools to start monitoring the project directory for changes and to display messages on the console.

Note: The **wrd** command itself is used to tell the rapid deployment tools to start monitoring the project directory and the (optional) `-monitor` switch enables the console output messages.

The wrd command displays Example 17-12 in the console window:

Example 17-12 Launching wrd.bat -monitor

```
Launching WebSphere Rapid Deployment. Please wait...
Starting Workbench...
```

```
WebSphere Rapid Deployment ready for e-business...
```

```
Type 'q', 'quit', or 'exit' to shut down WebSphere Rapid Deployment processes.
```

To terminate a rapid deployment session, type **Q**, **quit** or **exit** and press **Enter** in the console window. This stops the monitoring of the project directory. You can start monitoring the directory by simply running the **wrd** command again.

17.7.2 Managing applications

When the WebSphere Rapid Deployment session is configured and monitoring of the project directory is activated, we can use it for installing, updating and uninstalling our application.

Installing an application

To install an application, simply copy the EAR file to the monitored directory:

- ▶ Copy the HelloApp.ear file to the `c:\wrd\install_server1` directory. The application is automatically installed on the application server, as shown in the console output window in Example 17-13.

Example 17-13 Installing the HelloApp.ear

```
[01:09:12 PM]  [/install_server1>HelloApp.ear] Added
[01:09:12 PM] !INSTALL_EAR_FILE HelloApp.ear!
[01:09:12 PM] Publishing HelloApp to server_1617843630
[01:09:14 PM] Installing New Application: HelloApp
[01:09:18 PM] ADMA5016I: Installation of HelloApp started.
...
...
[01:09:19 PM] ADMA5013I: Application HelloApp installed successfully.
[01:09:35 PM] Installation Completed Sucessfully: HelloApp
[01:09:37 PM] Starting Application: HelloApp
[01:09:38 PM] Application Started Sucessfully: HelloApp
```

You can access the application at

<http://localhost:9080>HelloWeb/HelloServlet>.

The output is shown in Figure 17-5 on page 982.

Note: The rapid deployment tools do not automatically regenerate the HTTP server plug-in configuration file when an application has been installed, updated or uninstalled. You must manually regenerate the plug-in configuration file.

Updating an application

To show what happens when an application is updated, we changed the getGreeting method of the HelloEJB to return another greeting message. We then exported the application as a new HelloApp.ear file and copied it into the c:\wrd\install_server1 directory, replacing the earlier version of the HelloApp.ear file. When the change is picked up by the WebSphere Rapid Deployment tools, Example 17-14 is shown in the console window:

Example 17-14 Updating the HelloEJB

```
[01:26:57 PM]  [/install_server1>HelloApp.ear] Modified
[01:26:57 PM]  !INSTALL_EAR_FILE HelloApp.ear!
[01:26:57 PM]  Publishing HelloApp to server_1617843630
[01:26:58 PM]  Reinstalling Application.
[01:27:18 PM]  ADMA5017I: Uninstallation of HelloApp started.
...
...
[01:27:18 PM]  ADMA5013I: Application HelloApp installed successfully.
[01:27:34 PM]  Application Updated Successfully. HelloApp
[01:27:39 PM]  Starting Application: HelloApp
[01:27:39 PM]  Application Started Sucessfully: HelloApp
```

Access the application again at

<http://localhost:9080>HelloWeb/HelloServlet>

See the new greeting message as shown in Figure 17-7 on page 988.



Figure 17-7 Updated HelloApp running

Uninstalling an application

To uninstall the application, simply delete the EAR file from the project directory. The rapid deployment tools will automatically uninstall it from the application server. The console window shows Example 17-15.

Example 17-15 Uninstalling HelloApp.ear

```
[01:31:50 PM] [/install_server1>HelloApp.ear] Deleted
[01:31:50 PM] !DELETE_EAR_WRAPPER HelloApp.ear!
[01:31:50 PM] Uninstalling HelloApp:server_1617843630
[01:31:59 PM] ADMA5017I: Uninstallation of HelloApp started.
...
[01:31:59 PM] ADMA5106I: Application HelloApp uninstalled successfully.
[01:32:35 PM] Application Uninstalled: HelloApp
```

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 991. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *WebSphere Application Security V6 Security Handbook*, SG24-6316
- ▶ *WebSphere Application Server V6 Scalability and Performance Handbook*, SG24-6392
- ▶ *WebSphere Application Server V6: High Availability Solutions*, REDP-3971
- ▶ *WebSphere Application Server V6 Migration Guide*, SG24-6369
- ▶ *WebSphere Application Server V6: Web Services Development and Deployment*, SG24-6461
- ▶ *IBM WebSphere Application Server V5.1 System Management and Configuration*, SG24-6195

Other publications

These publications are also relevant as further information sources:

- ▶ *Transaction Processing: Concepts and Techniques* (Jim Gray, Andreas Reuter), Elsevier Science & Technology Books, ISBN 1-55860-190-2
- ▶ *Enterprise Messaging Using JMS and WebSphere* (Kareem Yusuf), Prentice Hall, ISBN: 0-13-146863-4
- ▶ *Java Message Service* (Monson-Haefel, Chappell), O'Reilly, ISBN: 0-596-00068-5
- ▶ *Professional JMS* (Grant, Kovacs, et al), Wrox Press Inc., ISBN: 1861004931
- ▶ *Enterprise JavaBeans, Fourth Edition* (Monson-Haefel, Burke, Labourey), O'Reilly, ISBN: 0-596-00530-X
- ▶ *EJB Design Patterns* (Marinescu), Wiley, ISBN: 0471208310

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ *Service Data Objects*, John Beatty, Stephen Brodsky, Raymond Ellersick, Martin Nally, Rahul Patel
<ftp://www6.software.ibm.com/software/developer/library/j-commonj-sdowmt/Commonj-SDO-Specification-v1.0.doc>
- ▶ WebSphere Application Server home page
<http://www.ibm.com/software/webservers/appserv/was/>
- ▶ WebSphere Application Server system requirements
<http://www.ibm.com/software/webservers/appserv/doc/latest/prereq.html>
- ▶ WebSphere Application Server support (Fix Packs, fixes, and hints and tips)
<http://www.ibm.com/software/webservers/appserv/support.html>
- ▶ Java Community Process home
<http://www.jcp.org/en/jsr/all>
- ▶ *Java 2 Platform Enterprise Edition Specification, v1.4*
http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf
- ▶ *Java 2 Platform Enterprise Edition, v 1.4 API Specification* at:
<http://java.sun.com/j2ee/1.4/docs/api/index.html>
- ▶ WebSphere Application Server Information Center
<http://www.ibm.com/software/webservers/appserv/infocenter.html>
- ▶ *MBeanInspector for WebSphere Application Server*
<http://www.alphaworks.ibm.com/tech/mbeaninspector>
- ▶ *Sample Scripts for WebSphere Application Server Versions 5 and 6*
<http://www-106.ibm.com/developerworks/websphere/library/samples/SampleScripts.html>
- ▶ Tcl Developer Xchange
<http://www.tcl.tk/>
- ▶ IBM WebSphere Developer Technical Journal
<http://www-106.ibm.com/developerworks/websphere/techjournal/>
- ▶ 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>
- ▶ IBM alphaWorks emerging technologies
<http://www.alphaworks.ibm.com>
- ▶ IBM developerWorks
<http://www.ibm.com/developerworks/>
- ▶ Worldwide WebSphere User Group
<http://www.websphere.org>
- ▶ *An Introduction to Java Stack Traces*
<http://java.sun.com/developer/technicalArticles/Programming/Stacktrace/>
- ▶ *Apache HTTP Server Log Files*
<http://httpd.apache.org/docs/logs.html>
- ▶ IBM HTTP Server documentation library
<http://www.ibm.com/software/webservers/httpservers/library/>
- ▶ *WebSphere MQ Using Java*
<http://www-306.ibm.com/software/integration/mqfamily/library/manuals/crosslatest.html>
- ▶ Java Message Service (JMS)
<http://java.sun.com/products/jms>
- ▶ *Persistent Client State HTTP Cookies*
http://home.netscape.com/newsref/std/cookie_spec.html
- ▶ 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

\$ 314
\${DB2UNIVERSAL_JDBC_DRIVER_PATH} 921
\${DRIVER_PATH} 330
\${LOG_ROOT} 423
\${SERVER_LOG_ROOT} 423
\${USER_INSTALL_ROOT} 423
\$AdminApp 275–276
 edit 307
 editInteractive 307
 install 305–306
 installInteractive 305
 list 297–298
 options 305
 uninstall 306
\$AdminConfig 274, 276, 281–282, 285
 attributes 285
 create 302, 308
 create Server 304
 createUsingTemplate 314
 defaults 284
 getid 287, 302, 304, 308, 310, 314
 list 284, 307, 310
 listTemplates 314
 modify 302, 308, 310
 parent 283
 queryChanges 308
 remove 305
 reset 290
 save 302, 306
 show 286, 308
 showAttribute 287, 308
\$AdminConfig types 282
\$AdminControl 274, 277, 293, 296
 queryNames 277–278, 298
 refreshRepositoryEpoch 101–102
 stopServer 294
\$AdminTask 275–276, 281, 288
 createApplicationServer 303
 createCluster 291, 310
 createClusterMember 313
 createSIBus 290–291
 deleteClusterMember 313

deleteServer 304
exportServer 263
exportWasprofile 263
help 289
\$Help 275

A
access intent 875, 878, 880
 application profile 881
 policies 878
access intents
 application profiles 210
 tracing 885
activation specification 927
activation.jar 61, 355
ActivationSpec 50
ActivationSpec JavaBean 489, 491, 493–494, 507, 509
Active Server Pages 938
ActiveX to EJB Bridge application client 938, 940
ActivitionSpec 32
activity session service 211
activity.log 421, 449
ActivitySession Service 15
addNode 220–221, 223, 225
admin_host 38, 240
AdminControl 279
administration services 185, 212
administrative console 68
 changing the session timeout for the adminconsole application 165
 home page 166
 logging in 164
 scope 171
 securing 182
 starting 163
administrative console port 126, 133
administrative console secure port 126, 133
administrative service 29, 33
AdminServer 157
AdminService 93–94
AffinityCookie 405
Aged Timeout 337–338

alias destination 602, 634, 668
ALL_AUTHENTICATED 815–816
analyze 449
annotation-based programming 957–958
Apache 915
apache 459
Apache Server 12
apachectl 396
applet application client 938
application
 deploying 930
 editing with wsadmin 307
 exporting 246
 finding the URL 253
 installation 242, 244, 305, 986
 listing installed 297
 multiple updates 948
 preventing from starting 247
 preventing startup 307
 removing files 947
 single file update 946
 single module update 946
 starting 247
 starting order 247
 starting with wsadmin 298
 stopping 247
 stopping with wsadmin 298
 uninstalling 246, 988
 uninstalling with wsadmin 306
 updating 944, 987
 viewing 248
 viewing EJB modules 250
application class loader 827, 832, 839
application classloader policy 207
application client 937
 deployment 937
 launching 941
application client bindings 940
application client container 26, 28
application extensions class loader 841
application module
 updating 945
Application Profiling 15
application profiling service 210
Application Response Time (ARM) agents 36
application server 24, 69
 clustering 24
 creating 192, 303, 910
 logs and trace 912
 modifying with wsadmin 309
 removing 304
 restarting 202
 runtime attributes 203
 starting 197, 295
 stopping 200–201, 296
Application Server Facilities 484
application server profile 114–117, 119–121, 130, 149, 151, 153, 163, 191, 197, 200, 259, 261, 263–264, 271, 316, 389, 971, 984
Application Server Toolkit 10, 17, 76, 78, 456, 850, 856, 858, 864, 903, 909, 928, 930
application.xml 108, 857
application-client.xml 857
applications
 deployment
 dynamic reload 887
 hot deployment 887
 starting and stopping 178
Asynchronous Beans 15
asynchronous beans 320
asynchronous messaging 465–466
attributes 280, 282, 285
authentication 51, 54–55
 component managed 374
 component-managed 333–334, 352
 container 375
 container-managed 333–334
 resource 374
authentication alias 662, 682
authorization 55
auto reload 887
automatic application installation 957, 960, 963, 983–985
automatic file synchronization 953
autoRequestEncoding 888
autoResponseEncoding 888

B

backend ID 245, 866–867, 869, 933
backupConfig 259–260
bean managed activity session 765
bean managed transaction 765–766
bean managed transactions 505
Bean Scripting Framework (BSF) 69, 268
bean-managed transaction 31
binding 860
 application client bindings 940

compound name 774
configured 770
corbaname 774
CorbaObjectNameSpaceBinding 790
data sources 865
EJB JNDI names 861
EJB references 862
EjbNameSpaceBinding 790
IndirectLookupNameSpaceBinding 790
name 771
overriding defaults 930
simple name 773
StringNameSpaceBinding 790

bindings
 configured 35
 Name bindings 34

bindings connection 552–553

bindings file 936

BMP 877–878

boot class path 184

bootstrap 785, 791–794, 800, 802–804, 808–809, 822

bootstrap class loader 822, 825

bootstrap endpoint 682

bootstrap server 577, 579, 588–589, 616, 628

BOOTSTRAP_ADDRESS 219

BootstrapBasicMessaging 617

bootstrapnoderoot 796

bootstrapped client 588

BootstrapSecureMessaging 617

bootstrapserverroot 796

BootstrapTunneledMessaging 617

BootstrapTunneledSecureMessaging 618

Built-in Mail Provider 61, 355, 357

bus member 47, 595, 600, 603, 644–645, 648, 654
 adding to the service integration bus 664

C

cache 210, 721, 738, 754–755, 759, 870, 874
 EJB 871–873

cache disk offload 32

cache ID 716–717

cache identifier 715

cache replication 31

cached connection handles 533

cached handles 341, 353

cacheGroups 285

Caching Proxy 10, 78

CCI 343

cci.jar 344

cell 97, 106, 162
 definition 23

cell persistent root 770, 776, 779, 781, 784, 790, 794–796, 798–800, 807–808
 definition of 776

cellroot 796

CICS 42, 344

class loader 211, 822, 825, 827–828, 831–832, 835, 839
 Java 2 class loaders 822
 policies 828
 WebSphere class loaders 825
 RCP directory 826
 RE directory 826
 RP directory 826

class loader policy 830

class loading 184

class loading mode 207

class path 184

class preloading 833–834

classloader 931

classloader policy 207

ClassNotFoundException 821

Class-Path 838–839

classpath 857
 JDBC provider 329
 protocol provider 359
 resource adapter 348
 URL provider 365

client caching 42

client connection 552–554

client/server replication 725

client/server topology 37

client-server topology 727, 733

Cloudscape 12

cluster 17–18, 23–24, 47, 49–50, 64–65, 75, 167, 191, 235, 241, 244, 597, 625, 638, 640, 643–644, 651–652, 654, 755, 870, 872, 909, 932, 941, 944, 951, 953
 add a server using wsadmin 313
 adding messaging engines 670
 create with wsadmin 310
 creating 236
 definition 24
 managing with wsadmin 299
 message-driven beans 651
 messaging engine 640, 646

messaging engines 583
restarting servers 202
starting 299
starting and stopping 239
stopping 299
viewing topology 238
workload management 66

cluster.xml 106
CMP 876, 878, 883–884
CMP 2.0 enterprise bean 345
cmpConnectionFactory 345
Collector tool 418, 451
com.ibm.scripting.host 270
com.ibm.websphere.rsdadapter 324
com.ibm.ws.rsdadapter.cci 324
com.ibm.ws.rsdadapter.jdbc 324
com.ibm.ws.rsdadapter.spi 324
com.ibm.ws.scripting.connectionType 270
com.ibm.ws.scripting.defaultLang 270
com.ibm.ws.scripting.traceFile 270
com.ibm.ws.scripting.traceString 270
com.sun.jndi.ldap.LdapCtxFactory 773
Common Client Interface (CCI) 343
Common Secure Interoperability 53
communication channel 97
communication settings 212
compiled language debugger 456
completeObject 301
completeObjectName 278–279, 292, 295
component managed authentication 374
component managed authentication alias 574
component-managed authentication 333–334, 352
component-managed authentication alias 556, 663, 924
CompoundClassLoader 839
concurrency control 877
concurrent message consumers 484
configuration ID 283
configuration reload 657, 659
configuration repository 69–70
connection factory 344–345, 472, 572, 579, 581, 641
 bindings 931
 CMP 332
 data source 325
 J2C 349, 351
 JCA 344
 resource adapter 345
 WebSphere MQ 552
connection factory bindings 245
connection handles 341
connection management 486
connection management contract 342
connection manager 30
Connection object 475
connection pool 758–759
connection pooling 321, 335, 342
connection proximity 583–584
Connection Timeout 336–337
ConnectionFactory object 475
ConnectionWaitTimeoutException 336–337
connectors
 JMX 87
console
 See administrative console
container managed activity session 765
container managed authentication 375
container managed transaction 765–766
container managed transactions 504
container-managed authentication 333–334
container-managed authentication alias 334
container-managed persistence 332, 878
 See also CMP
container-managed relationship 884
container-managed transaction 31
context root 253–255, 946
cookies 38, 699, 703, 705–708, 715
CORBA 33, 791, 811, 815
 naming service groups 817
 naming service users 816
 URL 791
corbaloc 35, 770, 777, 779, 781, 784–786, 791–792, 795–796, 801–804, 806, 811
corbaname 35, 770, 773–774, 785–787, 789–790, 793, 800–801
core group 67, 185, 193, 219, 223
core group bridge service 213
core group policy 641, 672
core group service 213
correlation ID 449
CosNaming 33–34, 769, 771, 787, 789, 791, 794, 797, 811
 CORBA 797
 CORBA interface 772
 distributed 787
 INS 785
 JNDI plug-in 794
 security 815

CosNamingCreate 815
CosNamingDelete 816
CosNamingRead 815
CosNamingWrite 815
Covalent Enterprise Ready Server 12
capplication client module 74
createSubcontext 815
createUsingTemplate 314
CSIV2 51, 57–58, 77
CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRE SS 219
custom profile 115, 117, 120, 138, 143, 145–147, 263
custom services settings 185
custom user registry 54
CVS 73

D

data centric 464
data replication service 698, 724–725, 728
Data Replication Service (DRS) 36
data source 59–60, 72, 662, 922, 924
 binding using the AAT 865
 creating 326, 331
 version 4 325
 version 5 323
data source connection factory 849
data sources
 mapping to CMP beans 934
DataAccessFunctionSet 324
database mapping 867
database mapping editor 869
database persistence 721, 742
database reauthentication 340
DataDirect Technologies JDBC Drivers for Web-Sphere Application Server 10
datagram 468
datasource helper classname 333
DataSource object 59, 321–322
DataStoreHelper 324, 333
DB2 7, 10, 12
DB2UNIVERSAL_JDBC_DRIVER_PATH 918
DDL 244
debug adapter
 JavaScript 456
 WebSphere 456
debugging service 213
default core group 219
default data source mapping 934
default error page 888
default messaging provider 62–63
default node group 129, 219
default profile 117–118, 122–123, 130, 139, 383–384, 409
Default URL Provider 62
default virtual host 208, 245
default_host 240, 257–258
default_host virtual host 37
DefaultCoreGroup 185
DefaultNodeGroup 23
defaults 284
delegation 822, 831, 839, 841
deployment descriptor 97, 107, 109, 344, 858, 860, 865, 870, 875
 application 97
 EJB module
 IBM extensions 885
 viewing 251, 254
deployment manager 10, 23–24, 33, 68–70, 78, 129–130, 133, 143
 starting 155, 163, 187–188, 293
 stopping 187, 189–190, 293
deployment manager node 129
deployment manager profile 115–116, 120–121, 151, 153, 183, 185, 187–188, 191, 264
deployment.xml 108
DES 730
destination 465–466, 473, 602
 configuration 574
destinations 48
destroySubcontext 816
diagnostic trace service 437
diagnostic trace service settings 185
directory browsing 888
discovery address 95
distributed discovery 94
distributed server configuration 34
distributed server environment 82, 84, 92, 106, 119, 378, 382, 936, 945, 950, 972, 985
distributed transaction 484
dmgr 162
 server 779
 starting 163
dmgr_profile_home 104
DNS domain 708
dumpNameSpace 453, 808
durable subscription 471, 493, 529, 532, 547

dynamic cache 29, 31, 210, 728
dynamic cache replication 36
Dynamic Query 15
dynamic reload 887, 954–955
DynamicCache object type 284–285

E
Edge Side Include caching 32
editInteractive 307
EIS 343–344, 346
eis/datasourcename_CMP 332
EJB
 binding 773
 EJB client 66, 798
 EJB collaborator 58
 EJB container 26, 28, 36, 65–66, 72, 209, 870
 EJB Deploy Tool 928
 EJB home 34, 770–771, 789, 797–798, 800–801
 EJB module 74
 deployment descriptor
 IBM extensions 885
 viewing 250
 EJB references
 mapping to beans 935
 EJB security collaborator 57
ejbCreate 501
EJBDeploy 929
EJBHome 742
ejb-jar.xml 857–858, 860
EJBLocalObject 767
EjbNameSpaceBinding 799
EJBObject 742, 767
ejbRemove 500, 502
embedded JMS provider 17
embedded JMS Server 637
end of servlet service 735–736, 747–748
Enhanced EAR 17, 74, 847, 891–894, 903, 908, 932
enterprise applications
 extensions sharing session context 889
 See also application
enterprise beans
 isolation level attributes 877
Enterprise Edition management API 85
Enterprise Information Systems (EIS) 341
Enterprise JavaBeans
 using EJBDeploy 928
Enterprise Service Bus (ESB) 17

enterprise services 42
enterprise Web services 41, 43
EntirePool 338
entity beans 870, 872–873, 877–878, 882, 884–885, 917
 caching options 870
environment entry 184, 211
environment variables 909
 See also variables
epoch 103
EventProvider 91
EventType 438
EVERYONE 815–816
exception destination 625–626
Extended JTA Support 15
extensions class loader 822, 825–827, 841
external caching 32

F
factoryClassname 372–373
fail back 642–643, 672, 674, 677
FailingConnectionOnly 338
failover
 critical services 67
 deployment manager 66
 EJB container 66
 hot 724
 HTTP server 66
 JMS messaging 67
 messaging engine 598
 node agent 66
 stateful session beans 17
 Web container 66
federated name space 34, 770–771, 774, 776–777, 779, 781, 784, 788, 798, 808
FFDC 452
file permission mode mask 184
File Serving Enabled 405
file serving servlet 886
file servlet 886
file synchronization service 99, 227
file transfer 70
findByPrimaryKey 880–881, 884
finder method 880
fire and forget 468
First Failure Data Capture
 See FFDC
First Steps 123, 128–130, 134, 136–137, 142, 200

foreign bus 606–607, 609–612, 627, 634–635, 657, 678, 680, 682–683
 service integration bus 678
 WebSphere MQ 629
foreign bus link 678
foreign destination 603, 692
free-form development 957, 959
free-form project 961–962, 968–970, 978
free-form projects 967
frequency settings 735
Full Resynchronize 103

G

garbage collection 184, 755
generic JMS provider 62–63
generic server 25
GenPluginCfg 408–409
getAttribute 280
getConnection() 322, 325
getid 283
getLocalHost() 96

H

Handle 742
heap size 184, 211, 755, 910
help 181
high availability 50, 66
 messaging 645, 647, 651
 messaging engine 638
high availability domain 67
High Availability Manager 18
high-volume Web sites 758
HomeHandle 742
host alias 239, 257, 915
host aliases 240, 258
hosts file 916
hot deployment 887, 954
hot failover 724
hot standby 67
 transaction log 67
htpasswd 399
HTTP session persistence 36
HTTP session tracker servlet 454
HTTP transport 208
HttpServletRequest
 getHeader 715
 getSession 706, 751
HttpServletResponse

encodeRedirectURL 709
encodeURL 709
HttpSession 713, 735–738, 741–742, 745–747, 749–751, 756, 766
 getAttribute 747
 getId 715
 removeAttribute 745
 setAttribute 745, 747

I

IBM HTTP Server 10, 12, 26, 399, 404, 915
 admin password 399
 administration server port 398
 listen ports 917
 logs 440
 NameVirtualHosts 916
 remote administration 397
 Startup errors 917
 version 459
 VirtualHosts 915
ibm.websphere.preload.classes 834
ibm-application-bnd.xmi 108
ibm-application-ext.xmi 108
ibmconfig directory 858
ibm-ejb-access-bean.xmi 858
ibm-ejb-jar-bnd.xmi files 858
ibm-ejb-jar-ext.xmi 858
ibm-partialapp-delete.props 949
IBMSession
 isOverFlow 713
 sync 735–736
IBMTckerDebug 454
IBMTckerDebug servlet 454
IIOP 785, 791, 793
 URL 791
IMAP 61, 355–356, 363
immediate stop 201
IMS 42
inbound user ID 680, 684
InboundBasicMessaging 614
InboundBasicMQLink 614
InboundSecureMessaging 614
InboundSecureMQLink 615, 633
index.html 917
indirect foreign bus 637
IndirectLookupNameSpaceBinding 799
InetAddress.getLocalHost() 96
Informix Dynamic Server 12

initial context 769–771, 791, 794–796, 801, 809
 default 799–800
initial context factory 791–793, 797, 800–801, 809
InitialContext 791
in-memory buffer 432, 438
install 306
installed optional packages 834
installedApps directory 109
installInteractive 305
inter-engine authentication alias 659
interface centric 465
Internationalization Service 15
Interoperable Naming Service (INS) 35, 769–770
invoke 281, 294
invoker servlet 887
isolation level 875–877
isolation level attributes 877
IsolationLevelChangeException 877

J

J2C 344, 349
J2C authentication alias 925
J2C authentication data 919
J2C connection factory 349
J2EE 1.4 85
J2EE application client 937
J2EE client 941
J2EE Connector Architecture (JCA) 30
J2EE Management 77
J2EE security 53
J2SE 1.4 85, 456
JAAS 29, 51, 55, 77
JAAS authentication alias 849, 894
JACC 51, 55, 77
Jacl 272–273, 296–297, 302, 306
JAF 77, 355
JAR manifest file 838
Java 2 Platform, Enterprise Edition (J2EE) 4
Java 2 security 29, 51–52, 55, 108
Java 2 Standard Edition (J2SE) 4
Java Activation Framework (JAF) 355
Java and process management settings 184
Java API for XML Messaging (JAXM) 498
Java API for XML Registries (JAXR) 13
Java API for XML-based RPC (JAX-RPC) 13
Java Authentication and Authorization Services (JAAS) 52
Java Authorization Contract with Containers (JACC) 18
Java Build Path 854–855
java comp name 813
java comp/env 369
Java Contract for Containers (JACC) 53
Java DataBase Connectivity 321
 See also Resource providers JDBC
Java logging interface 456
Java Management Extension (JMX) 78
Java Management Extensions (JMX) 85
Java Native Interface (JNI) 322
Java Secure Socket Extension (JSEE) 53
java.net.URLConnection 364
java.net.URLStreamHandler 364–365
JavaBeans Activation Framework (JAF) 61, 355–356
JavaMail 59–61, 77, 354–357, 360–361
 Built-in Mail Provider 61
JavaScript debug adapter 456
JavaServer Faces (JSF) 15
JavaServer Pages 888
javax.ejb.EJBHome 741
javax.ejb.EJBObject 741
javax.ejb.MessageDrivenBean 499
javax.ejb.MessageDrivenContext 503
javax.jms.JMSException 582
javax.jms.MessageListener 500
javax.jms.Queue 850
javax.naming.Context 741
javax.naming.directory 773
javax.naming.ldap 773
javax.naming.ObjectFactory 372
javax.servlet.http.HttpSessionActivationListener 750
javax.servlet.http.HttpSessionAttributeListener 750
javax.servlet.http.HttpSessionListener 750
javax.transaction.UserTransaction 741
javax.xml.messaging.ReqRespListener 500
JAXM 500
JAXP 77
JAXR 41, 77
JAX-RPC 41, 43–44, 48, 77
JCA 12, 341, 343, 349
 CCI implementation 60
 connection manager 60
 resource adapter 61
 services 29–30
JCA CCI 323–324
JCA connection manager 323, 343

JCA connector 50
JCA resource adapter 342
JCA resource reference 934
JCA Web services 42
JDBC 77
JDBC driver 59, 918
JDBC provider 59, 662, 849, 920
 configuring with wsadmin 313
 creating 326
JDBC resource provider 59
JDT 456
JMS 8, 12, 17, 77
JMS activation specification 509–510, 515, 518, 545, 547, 549–551, 571
JMS activation specification. See also Activation-Spec JavaBean
JMS administered object 550
JMS administered objects 472, 490, 523, 526
JMS client 49, 576, 580–581, 588, 616
JMS connection 475
JMS connection factory 472, 515, 527, 534, 849, 926
JMS destination 32, 473, 491, 926
 generic JMS provider configuration 574
JMS domains 471
JMS exception 482
JMS message 476–477
JMS message selector 478
JMS provider 59, 62, 471, 514, 926
 default messaging 514, 519, 529, 539, 541, 547, 580
 generic 522–523, 572
 WebSphere MQ 519, 552
JMS queue 850, 926
JMS server 25, 785
JMS session 476
JMX 36, 67, 69, 77, 83, 87, 270, 276–277, 972, 985
 agent 87
 architecture 86
 connectors 87
 enabled management application 87
 ObjectName 92
JMX connector 185
JMX MBeans 58
JNDI 33–34, 64, 72, 75, 245, 771, 789, 797, 799–800, 802–804, 806–809, 811, 813, 815
 APIs 775
 caching 797
 client 791
Context.list() 810
EJB Home 789
initial context 809
initial context factory 800–801
javax.naming package 772
javax.naming.provider.url 791
JMS 473–474
objects registered by dmgr server 779
over CosNaming 771
provider URL 35, 770
service provider 800
 using to federate name space 797
JNDI bindings 847
JNI 184, 828, 938
JRas 913
JSESSIONID 38, 705, 707, 716
JSP
 finding the URL 253
JSP precompile 931
JspBatchCompiler 932
JSR 101 41
JSR 109 41, 43, 79
JSR-003 85
JSR-077 35
JSR-109 48
JSR-77 85
JSR-88 907
JTA 77
JTA XAResource API 484
Jython 273

L

Last Participant Support 15
launchClient 941–942
LDAP 51, 54, 77, 773, 797
legacyRoot 779, 781, 784, 799, 808
Light Weight Third Party Authentication (LTPA) 55
lindex 307
listener port 50, 509–510, 554, 568–570, 572, 933
listTemplates 314
Load Balancer 10, 78
load balancing 413
Location Service Daemon (LSD) 96
log 186, 198, 201, 396, 418
 activity 421, 428, 443
 activity.log 429
HTTP 382
IBM HTTP Server 440, 442–443

JVM 420–421, 425–426
merging 449
native 420
native_stderr.log 421, 428
native_stdout.log 421, 428
process 420
service 421, 428, 443
standard 421
startServer 189–190, 198–199
stderr 184
stdout 184
stopServer.log 202
System.err 421
System.out 421
SystemErr.log 420
SystemOut.log 420, 423, 458–459
Web server 401
Web server plug-in 439

Log Analyzer 78, 418, 421, 430, 443, 448, 913
 Merging activity logs 449
 starting 444
 Symptom database 443
 Updating the symptom database 450

log and trace settings 186

logs

- file formatting 424
- file rotation 424
- profile creation 128, 134, 142
- startServer.log 129, 135
- SystemOut.log 129, 135

loopback address 96

loose coupling 464

Lotus Domino Enterprise Server 12

LTPA 51, 55, 77

M

mail from 362

mail provider 356

mail store 362

mail transport 362

mail.jar 61, 355

managed application server 191

managed node 25, 378, 381–382, 386, 388–389

managed objects 85, 91

managed process 82, 772, 776–777, 781

managed server 16, 97

ManagedConnectionFactory 325, 345

manifest 838

MANIFEST.MF 844

master configuration 165

master repository 97

match criteria 642, 675

Max Connections 336–337

maximum failed deliveries 625

maximum in-memory session count 699

Mbean extensions 185

MBean proxy 89

MBean server 88–89

mbeanIdentifier 278

MBeans 86–87, 91, 93–94, 279, 292
 server 86

TraceService 301

mbList 277

mediation 50, 606, 659

memory-to-memory replication 699, 723, 725, 728–729

memory-to-memory session persistence 40

message consumer 468, 476, 479, 481, 610, 640

message consumer pattern 468

message consumers 645

message endpoint 488, 494–496

message endpoint proxy 496

message listener 29, 32, 481, 483, 489, 505

message listener service 212

message order 513

message point 603

message producer 468, 476, 478, 605–606, 610, 640

message producer pattern 468

message selector 546

message store 48

message-driven bean 50, 75, 77

message-driven beans 212, 245, 488, 493, 495, 497–502, 504–505, 507, 509–513, 515, 545–549, 551, 568, 572, 605, 640, 649, 651, 848, 861, 927, 933

- binding 864
- life cycle 501

MessageEndpointFactory 488

messaging bus 46

messaging client 636

messaging engine 47, 49, 581, 595–597, 619, 638, 644, 649

- data store 601, 620–621, 623, 660
- failover 598
- name 600
- policy type 642

preferred server 672
secure communications 619
messaging provider 464, 466
meta-data 285
metadata 111
METHOD_READY state 874–875
Microsoft Internet Information Services 12
Microsoft SQL Server 12
MIME 61, 241, 355
Min Connections 337
missing transaction context 341, 353
monitored directory 960, 983
monitoring policy 212
multi-broker domain 729
multibroker.xml
 733–734
multicast 95–96
multicast address 95
multi-row persistent session management 759
multi-row schema 743–744
multi-row session support 757
multithreaded access detection 340
multi-threaded garbage collection 755

N

name bindings 771
name server 34, 770
name service (JNDI) 29
name space 34, 770
 viewing contents 453
name space bindings
 configuring 811–812
NameService 787, 792, 794, 796
NameServiceCellPersistentRoot 795
NameServiceCellRoot 795
NameServiceNodeRoot 795
NameServiceServerRoot 781, 784, 792–793, 795
namestore.xml 107
naming clients 771
naming service
 name bindings 34
nanny process 157
native library path 329
native path 349
native_stderr.log 421, 428
native_stdout.log 421, 428
nboundBasicMQLink 632
nhanced 858

NoClassDefFoundError 825
node
 adding 220
 See also addNode
 clustering 25
 definition 23
 managing 217
 removing 225
 See also removeNode
 restoring 260
 starting 230
 stopping 156, 228, 230, 233
 synchronization 227
 See also syncNode
node agent 33–34, 69, 89, 92, 95–97, 99, 106, 129, 143, 146, 151, 156, 162, 198, 221, 277, 295, 770
 definition 70
 restarting 233
 starting 144, 155, 198, 230, 294
 stopping 144, 231, 294
node group 23, 129, 219, 223
node groups 16
node persistent root 770, 776, 781, 784, 790
 definition of 776
non-durable subscription 471, 652
nonpersistent message reliability 530
nonpersistent MQ messages 689
non-serializable J2EE objects 741

O

Object Pools 15
object pools 320
Object Request Broker (ORB) 32
Object Request broker (ORB) 29, 32
onMessage 501
operating system security 52
operations 280
optimistic 877
optimistic concurrency 878
OptimisticPredicate 882
Option A EJB caching 870
Option B EJB caching 871
Option C EJB caching 872
Oracle 12
ORB 58
ORB service 185, 211
ORB_LISTENER_ADDRESS 218
outbound user ID 680, 684

OutboundBasicMQLink 618
OutboundSecureMQLink 618
outputFilename 430
overwrite session management 699

P

parent 282
PARENT_FIRST 831, 836, 839, 841
PARENT_LAST 831, 839, 842
partition ID 716
partitioned queues 639
passivation 209, 874
peer-to-peer mode 724
peer-to-peer replication 725
peer-to-peer topology 37, 725–726, 733
preferred servers only 675
Performance Monitoring Infrastructure (PMI) 35
Performance Monitoring Instrumentation (PMI) 78
Performance Monitoring Interface (PMI) 29
performance monitoring service 206, 212
persistence manager 324, 875, 880, 883, 885
Persistence Resource Adapter 324
persistent area 34
persistent message reliability 530
persistent MQ messages 689
persistent partition 770
persistent session 711, 713, 741
persistent session database 757
persistent store 710, 870
pessimistic 877
PessimisticUpdate 880
pluggable application client 938
pluggable authentication module 54
plug-in configuration file
 automated propagation 412
 automatic regeneration 410
 propagating 411
 regenerating 406, 408
 viewing 407
plugin-cfg.xml 404
PMI request metrics 36
PMI service 35
Point-to-Point domain 482
Point-to-Point messaging model 466–467, 514, 519, 610
poison message 513
pooled connection 59
POP3 61, 355–356, 363

port
 admin_host virtual host 38
 bootstrap 785, 788, 791, 804, 809
 CELL_DISCOVERY_ADDRESS 95
 default host 37
 NODE_DISCOVERY_ADDRESS 95
 NODE_MULTICAST_DISCOVERY_ADDRESS 95
 SSL 240
pre-compile JSP 932
prefer local 237
preferred server 642, 647, 672, 677
preferred servers only 642–643, 647, 672, 674
Principals 55
print 425
print() 421
println() 421, 425
printStackTrace() 421
Private UDDI Registry 79
process definition 211
process execution 184, 211
process group assignment 184
processor partitioning 184
product information
 viewing 206, 457
profile 16
 deleting 263
 exporting and importing 262
Profile creation wizard 121, 123, 128–130, 134, 136, 138, 142, 151, 153–154, 158, 191
profile registry 151
profile_home 104, 118
ProfileCreator 121
profileRegistry.xml 151
profiles
 about 114
 creating 121
 types 116
Programming Model Extensions 15, 29
programming model extensions 210, 320
project directory 960
protocol adapters 87
 JMX 87
protocol provider 61, 354, 359
protocol switch rewriting 710
provider endpoint 578, 628, 641, 682
provider URL 789–790, 793
 javax.naming.provider.url 791
pseudo-synchronous messaging 469

publication point 604
publish/subscribe broker profile 633
publish/subscribe domain 471, 482
Publish/Subscribe messaging model 466–467, 514, 519, 612
publish/subscribe profile 689
pull mode 468, 480
Purge Policy 338
push mode 468, 480
Python 273

Q

quality of service 50
queryNames 277–278, 292
queue destination 515, 535, 537, 539, 554, 562–563, 602–603
 creating 666
queue destinations 561
queue manager 611, 616, 630, 632, 634, 683, 689
queue point 603
QueueConnectionFactory object 474

R

ra.xml 344, 349, 857
Rational Application Developer 10, 17, 43, 73–75
Rational ClearCase 73
Rational Rose 73
Rational Web Developer 10, 75
Rational XDE 73
RCF directory 826
RE directory 826
read ahead 539
read-ahead 883–884
Reap Time 337–338
Reap Timeout 337–338
receiver channel 631–632, 688–689
Redbooks Web site 991
 Contact us xxi
referenceable 64, 370–371, 373
RegenerateKey 730
relational resource adapter 60, 323–324, 333
reloadingEnabled 955
removeNode 225–226
replica 731
replication client 727
replication domain 237, 726, 728–730
replication entry 238
replication mode 733

replication server 727
replicationType 286
replicator entry 728–729
repository 84, 95–97, 104–106
 application data 107
 application execution 109
 saving work to 180
request routing 379
request-reply 468
request-reply pattern 469
reset 290
resource adapter 59, 74, 343–346, 349, 488, 649, 827
 deployment descriptor 489
 installation 345
 lifecycle management 486
 message inflow management 487
 packaging 489
 service integration bus 514, 517, 649
 transaction inflow management 487
 WebSphere Relational Resource Adapter 345
work management 486

Resource Adapter Archive (RAR) 344

resource environment entry 64
resource environment provider 59, 64, 370–371
resource provider
 J2C 341, 343, 345, 349
 JavaMail 356
 JDBC 326, 331
 URL 364, 368
 Configuring URLs 366
resource providers 58
resource references 935
resources.xml 326, 345, 370
res-sharing-scope 341
restoreConfig 259–260
ripplestart 239
RMI 218, 222, 270
RMI connector 87
RMI/IIOP 66, 222
Rollout Update 25, 944, 951–953
rollout update 244
round robin 414
round robin routing policy 66
RP directory 826

S

SAAJ 77

sample applications 137
SAP 344
SAS interceptor 58
scalability 50
Scheduler Service (Timer Service) 15
Schedulers 320
scope 107, 171–172, 176
security 18, 164, 182
 console 85
 session 751
 session management 751
 Web services 41
 WebSphere Application Server 29, 37, 51,
 56–57, 77
 WebSphere UDDI Registry 44
security collaborator 57
security contract 343
security management 486
security roles 246, 935
security server 56
sender channel 630–632, 688–689
serialize session access 699
serve servlets by class name 887
server
 starting 156
 status 156
 stopping 156
server ID 717
server root 770, 790
server root context 792–795, 798, 809
server weight 237
server weighted routing policy 66
server.xml 221
ServerCloneID 415
serverindex.xml 95, 97
serverStatus 135, 156, 163, 194–195
Service Data Object (SDO) 77
Service Data Objects (SDO) 14
service integration 17
service integration bus 17, 45–46, 48–50, 106, 149,
 168, 212, 218, 221, 223, 515, 577, 594, 849, 925
 architecture 594
 bus member. See bus member
 clustering 638
 configuration 655
 connecting to 576
 controlling messaging engine selection 579
 creating 658
 creating with wsadmin 290
data store 620
destination 602, 667
exception destination 626
foreign bus. See foreign bus
foreign link 680
JMS activation specification 509, 518, 545–546
JMS connection factory 515, 527–528
JMS destination 536
JMS queue 537, 540
JMS topic 542
link 627–629
load balancing bootstrapped clients 588
mediation 606
message-driven beans 649, 651
messaging engine. See messaging engine
quality of service 530
reliability 604, 606
resource adapter 509, 514, 516, 518
runtime components 612
scalability 600
security 652
SIB service. See SIB service
topic destination 543
topologies 645–646
transport chain 614
WebSphere MQ addressing 633–634
 XA recovery 533
service integration bus queue 850
service provider 355
Service Provider Interface (SPI) 343, 772
Servlet 2.2 API 718
servlet caching 208
servlets
 finding the URL 253
 serve by class name 887
servlets by class name 405
session
 performance 752
session administrative object 356
session affinity 698, 703, 705, 715, 755
session beans 873
 caching options 873
 stateful 874
 stateful EJB timeout 874
session cache 710
session cleanup settings 738
session context 701
session ID 702, 705, 710, 715–716
session identifier 706, 709

session invalidation time 738
session management 27, 38, 208
 affinity 715, 717–718
 cleanup schedule 749
 DB2 page sizes 742
 HTTP 698
 invalidating sessions 749, 755
 JSESSIONID 38
 last access time 735
 local 710
 maximum in-memory session count 712
 multi-row schemas 743
 overflow 713
 overflow cache 711–712, 754
 persistent 719–721, 734
 properties 699–700
 row type 743
 security ID 751
 security integration 714
 serializable requirements 741
 serialize session access 714
 session affinity 717
 session cache size 754
 session listeners 749
 session object size 753
 session size 753
 session timeout 755
 session tracking mechanism 703
 single-row schemas 743
 single-row to multi-row migration 744
 SSL ID tracking 704–705
 SSL session identifiers 38
 time-based write frequency 737
 write contents 745–746
URL rewriting
See also cookies
session management properties
 application 700
 application server 700, 761–762
 Overwrite Session Management 701
 Web module 700
session manager 711, 724, 728, 731, 751, 755, 757–759
 overflow 711
session object size 753
session persistence 39, 698, 705
session scope 700
session store 717
session write interval 737

SessionBeanTimeoutException 875
SESSIONS table 743
setAttribute 280, 300–301
setCharacterEncoding 889
setMessageDrivenContext 500, 502
setupCmdLine 221
shared libraries 834, 842
shared library class loader 832
shared session context 701
showall 288
showAttribute 287
Showlog 421
showlog 430
SIB JMS Resource Adapter 517, 649
SIB service 577, 612–615, 641, 648, 655–657, 682
sib.client.ssl.properties 619
SIB_ENDPOINT_ADDRESS 578, 614–615, 656
SIB_ENDPOINT_SECURE_ADDRESS 614–615, 656
SIB_MQ_ENDPOINT_ADDRESS 614
SIB_MQ_ENDPOINT_SECURE_ADDRESS 615
SIB_MQ_ENDPOINT_ADDRESS 615, 689
SIB_MQ_ENDPOINT_SECURE_ADDRESS 615
sibDDLGenerator 623, 662
Simple WebSphere Authentication Mechanism 751
Simple WebSphere Authentication Mechanism (SWAM) 54, 77
single sign-on 55
Singletons 833
SMTP 61, 355–356
SNMP 67
snoop 258
SOAP 79, 222
SOAP connector por 212
SOAP connector port 101, 126, 133
SOAP with Attachments API for Java (SAAJ) 13, 41
SOAP_CONNECTOR_ADDRESS 217, 220, 222
SOAP_CONNECTOR_PORT 218
special header 414
SPI 324, 343, 772
spilling 601
SQL92 conditional expression syntax 478
SSL 44, 216, 619, 629, 631, 705
SSL ID tracking 699
SSL session ID 703, 705
SSL session identifiers 38
SSLV3TIMEOUT 705
standalone application server 191
standalone server environment 83, 106, 119, 384

Standard Java security 52
startApplication 298
startManager 130, 155, 160, 163, 188, 190
startNode 144, 155, 198
startServer 136, 156, 198, 200, 202
startserver 396
startServer.log 189–190, 198–199
Startup Beans 15
stateful 873
stateful session bean failover 698, 729, 760–762, 764
stateful session EJB persistence 36, 40
StateManageable 91
statement cache size 340
StatisticsProvider 91
stderr 420, 423, 428
stdout 420, 423, 428
sticky bean managed unit of work 766
stopApplication 298
stopManager 190
stopNode 144, 156, 228, 231–232
stopServer 137, 156, 201–203, 293
stopServer.log 202
storeUsingOCC 882
stream 71
Stream Handler Class 365
Subject class 55
subscription durability 547
Sun Java System Web Server 12
sun.misc.Launcher\$AppClassLoader 822
SWAM 51, 55, 751
Sybase Adaptive Server Enterprise 12
symptom database 450
Symptom database *See* Log Analyzer
synchronization 70, 98, 100, 227–228, 389
 forced 99
 scheduling 99
synchronize 97, 103
synchronous messaging 465
syncNode 103, 228–229
system class loader 822, 825
System.err 420–421, 423, 425, 427
System.out 420–421, 423–425, 427
SystemErr.log 420
SystemOut.log 155, 301, 420, 423, 458–459

T

target groups 585

target mappings 247
target server 402, 960
template 147, 153, 192, 314
 application server (creating) 194
 Web server 393
thin application client 938
thread
 ID 427
thread pool 213
tightly coupled 464
time-based write 737
time-based writes 737
timeout
 aged 337–338
 connection 336–337
 ConnectionWaitTimeoutException 336
 reap 337–338
 session 713, 749, 755
 unused 337
Tivoli Access Manager 53
Tivoli Access Manager (TAM) 18
Tivoli Access Manager Servers for WebSphere Application Server 11
Tivoli Directory Server for WebSphere Application Server 11
Tivoli Performance Viewer 36, 78
topic destination 515, 535–538, 541, 555, 561–562, 566
topic mappings 633
topic space destination 602–603
 creating 668
topic space mappings 629, 682, 689
topic subscriber 471
trace 418, 430–431, 437, 913
 enabling using wsadmin 300
 output 438
 starting for a running server 437
 strings 433
 viewing output 438
 Web server plug-in 439
trace specification string 433
trace string 437
TraceService 300–301
transaction
 bean managed 505
 commit 870
 container managed 504
 isolation level 876–877
 message-driven beans 502

viewing 205
transaction management 29, 486
transaction management contract 342
transaction service 30, 205, 210
transient area 34
transient partition 770
transport chain 26, 186, 208, 216, 578, 583, 612, 614–616, 619, 629, 631–632, 659, 688
TRIPLE_DES 730
types 282

U

UDDI 41, 46
UDDI Registry 44
UnauthorizedSessionRequestException 751
Unified Modeling Language (UML) 73
unique ID 654
unit of work 447, 449, 765
Universal Description, Discovery and Integration (UDDI) 13
Universal Test Client 976
unmanaged node 378, 381, 386, 389
unmanaged Web server node 26
UriGroup Name 405
URL provider 59, 62, 364, 368
URL rewriting 38, 698–699, 703, 705, 709–710
user registry 53
user rights 127, 133
utility JAR 838–840

V

V5 default messaging provider 63
variable 179–180, 909–910, 918
variables 330
variables.xml 107
versionInfo 458
virtual host 75, 405, 901, 903, 917, 935
 admin_host 38, 240
 and Web server plug-in 241
 architectural overview 239
 architecture 37
 binding 931
 creating 240
 creating with wsadmin 308
 default_host 37, 240, 257–258
 example 914–915
 finding the URL for a servlet or JSP 253, 256–257

host alias 257
host aliases 240, 258
IBM HTTP Server 916
in a cluster 241
managing 239
map to Web modules 246
mapping 256
MIME settings 241
modifying with wsadmin 309
scope 240
virtual hosting 915
Virtual hosts 915
 See also IBM HTTP Server VirtualHosts

W

WAR class loader 827, 832, 839
WAR classloader 837
was.policy 108, 931
was_home 117
WAS_USER_SCRIPT 383
waslogbr 444
wasprofile 118, 151–153, 263–264
WASService 157–159, 194
WC_defaulthost 914
Web container 26, 57, 64–66, 71, 207, 889
 See also J2EE Web container
 overview 26
Web container inbound chain 27
Web container inbound transport chain 379
Web container transport chain 186
Web module 74–75, 886, 889
 auto reload 887
 default error page 888
 directory browsing 888
 file serving servlet 886
 serve servlets by class name 887
Web security collaborator 57
Web server 16, 25, 57, 932, 945, 950
 adding 392
 configuration file 400
 logs 401
 logs and traces 439
 starting and stopping 395
 viewing status 394
Web server plug-in 10, 26–27, 32, 57, 64–65, 71, 75, 378–380, 945, 951
 overview 27
 regenerating 406

request routing 412
Web server plugin 27, 240–241, 293, 715, 718–719, 755
generating 409
log 439
regenerating with wsadmin 300
trace 439
Web server plug-in configuration service 410
Web service client 43
Web service endpoint 48
Web service provider 43
Web services 8, 50
WebSphere Application Server support 40
Web Services Description Language (WSDL) 40
Web services engine 27
Web Services Explorer 43
Web services for J2EE (JSR 109) 79
Web services for Java 2 Platform, Enterprise Edition 13
Web Services Gateway 14, 17, 44, 51, 79
Web Services Security (WS-Security) 13
Web Services-Interoperability (WS-I) Basic Profile 13
web.xml 369, 857, 860
WebSphere
security server 56
Version 457
WebSphere Application Server 9
WebSphere Application Server - Express 8
architecture 20
WebSphere Application Server Enterprise 320
WebSphere Application Server Network Deployment 9
architecture 21
WebSphere Business Integration Event Broker 519
WebSphere Business Integration Message Broker 519
WebSphere Business Integration Server Foundation 8, 320
WebSphere Enhanced EAR 74
WebSphere MQ 8, 17, 49, 62–63, 519, 608, 633
link 630
WebSphere MQ client 634
WebSphere MQ client link 637
WebSphere MQ connection factory 552, 558, 571
WebSphere MQ JMS provider 62–63
WebSphere MQ link 630–635, 685
WebSphere Rapid Deployment 17, 75–77, 242, 957, 959–960, 967, 970, 973–974, 981, 985–986
WebSphere Relational Resource Adapter 62, 345
WebSphere Studio 42
WebSphere UDDI Registry 44
Windows service 126, 133, 150, 157, 159, 194, 223
WLM 698, 870
work area partition service 211
work area service 211
WorkArea Service 15
working directory 911
workload management 64, 67, 760
EJS WLM 870
messaging 644–645, 647–648, 651
messaging engine 638, 644
WebSphere Application Server 78
workspace 168
wrd 962, 966, 985
wrd-config 962–966, 972, 985
ws.ext.dirs 826–827
wsadmin 68, 268
definition 268
getid 283
help 269
interactive 271
list 284
profile 271, 273
properties 270, 274
script files 272
show 286
showattribute 287
starting session 268
WSCallerHelper 324
WSDL 44–46, 79
WS-I Basic Profile 79
WSIF 42, 44
WSIL 79
wsinstance 114
wsOptimisticRead 879–880
wsOptimisticUpdate 879
wsPessimisticRead 879
wsPessimisticUpdate 879
wsPessimisticUpdate-NoCollision 883
wsPessimisticUpdate-WeakestLockAtLoad 881
WS-Security 41

X

XAResource 30
XAResources 31
XDoclet 959

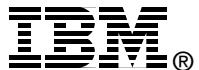
Xdoclet 957

IBM



WebSphere Application Server v6: System Management and

(1.5" spine)
1.5" <-> 1.998"
789 <-> 1051 pages



WebSphere Application Server V6 System Management and Configuration Handbook



**Read this book and
others in the
WebSphere
Handbook Series**

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

**Learn to design and
administer your own
system**

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 book, we provide a detailed exploration of the WebSphere Application Server V6 runtime environments and administration process.

**Customize profiles,
scripts and
applications**

**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**