WebSphere. software

IBM

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

ibm.com/redbooks

Archived

Petitioner has added romanettes to pages (i)-(iii) and numbers to pages 1013-1014.
Otherwise, it leaves the original page numbering.

SAP AMERICA, INC.
EXHIBIT 1026



International Technical Support Organization

WebSphere Application Server V6: System Management and Configuration Handbook

February 2005

Archived

SG24-6451-00

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

Archived

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

© Copyright IBM Corp. 2005. All rights reserved.

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

Archived

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

Archived

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

Archived

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

Archived

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

Archived

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

Archived

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

Archived

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.

© Copyright IBM Corp. 2005. All rights reserved.

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:

Archived

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

Archived

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.

© Copyright IBM Corp. 2005. All rights reserved.

Trademarks

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

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

Archived

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

© Copyright IBM Corp. 2005. All rights reserved.

Archived

1

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.

© Copyright IBM Corp. 2005. All rights reserved.

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.

Archive

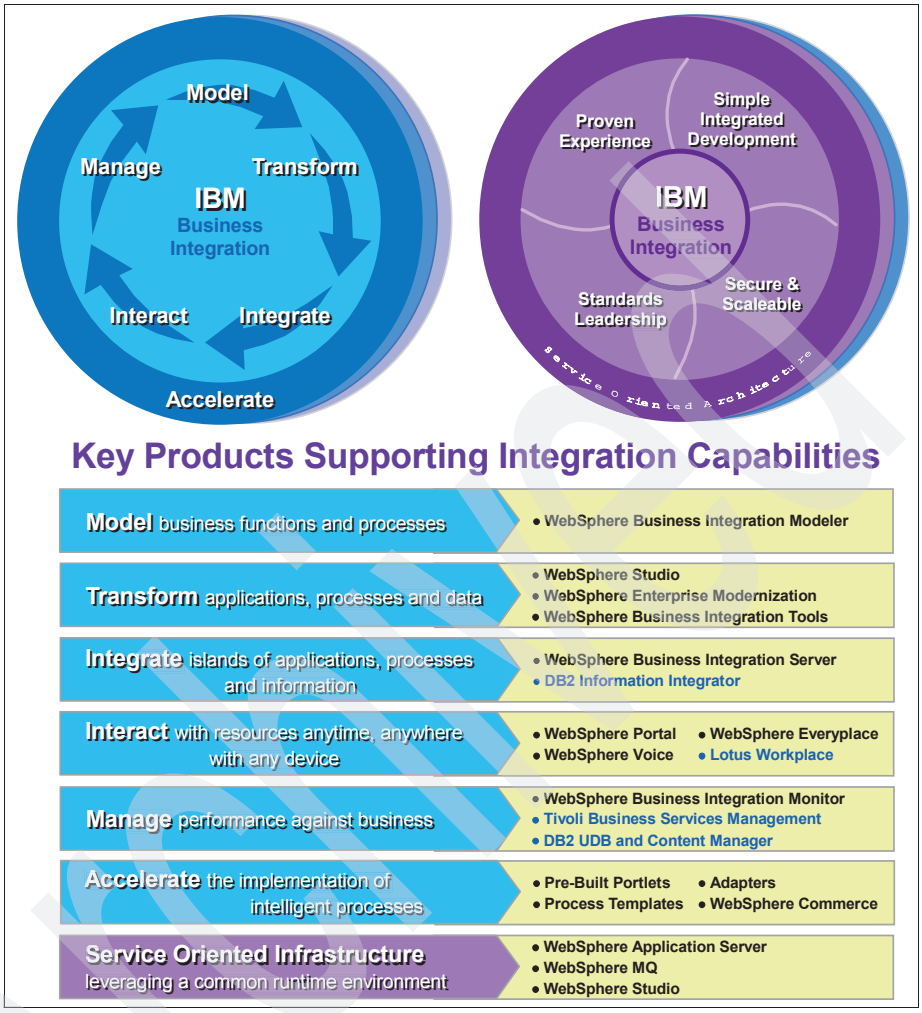


Figure 1-1 WebSphere Product family

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.

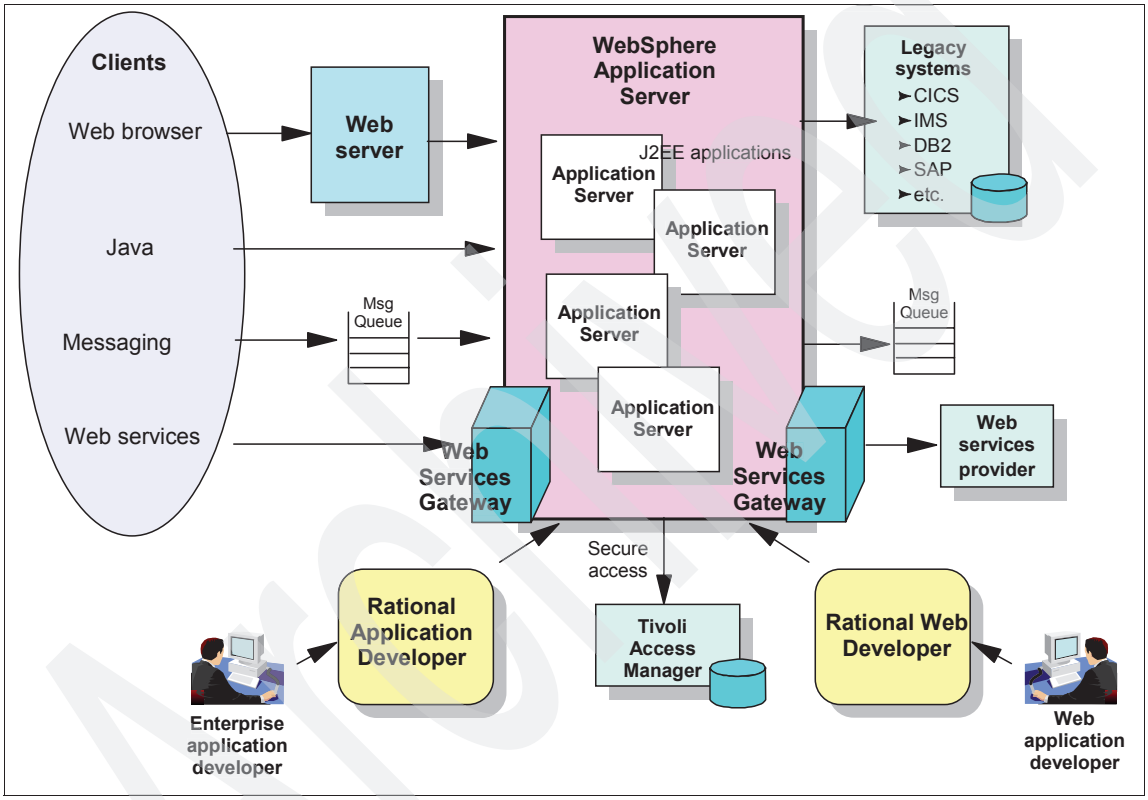


Figure 1-2 WebSphere Application Server product overview

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

Archived

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.

Archived

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

Abridged

- 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.

Archived

- 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 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.

Archived

- 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.

Archived

2

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*

© Copyright IBM Corp. 2005. All rights reserved.

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.

Archived

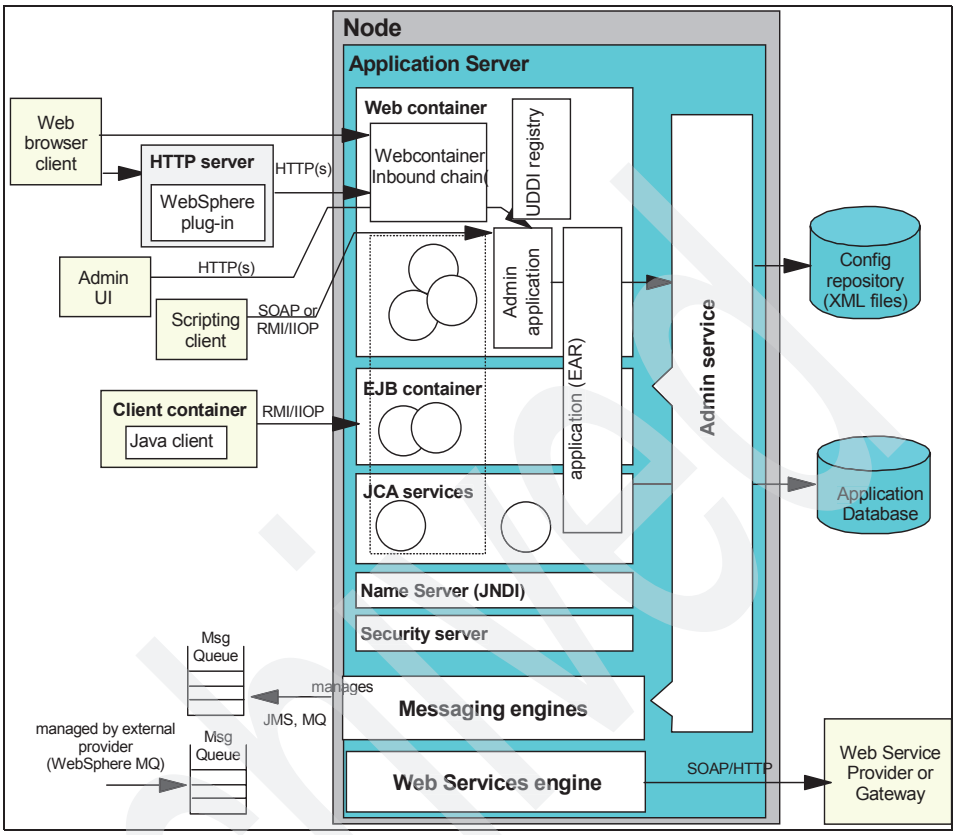


Figure 2-1 Architectural overview for a stand-alone server

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.

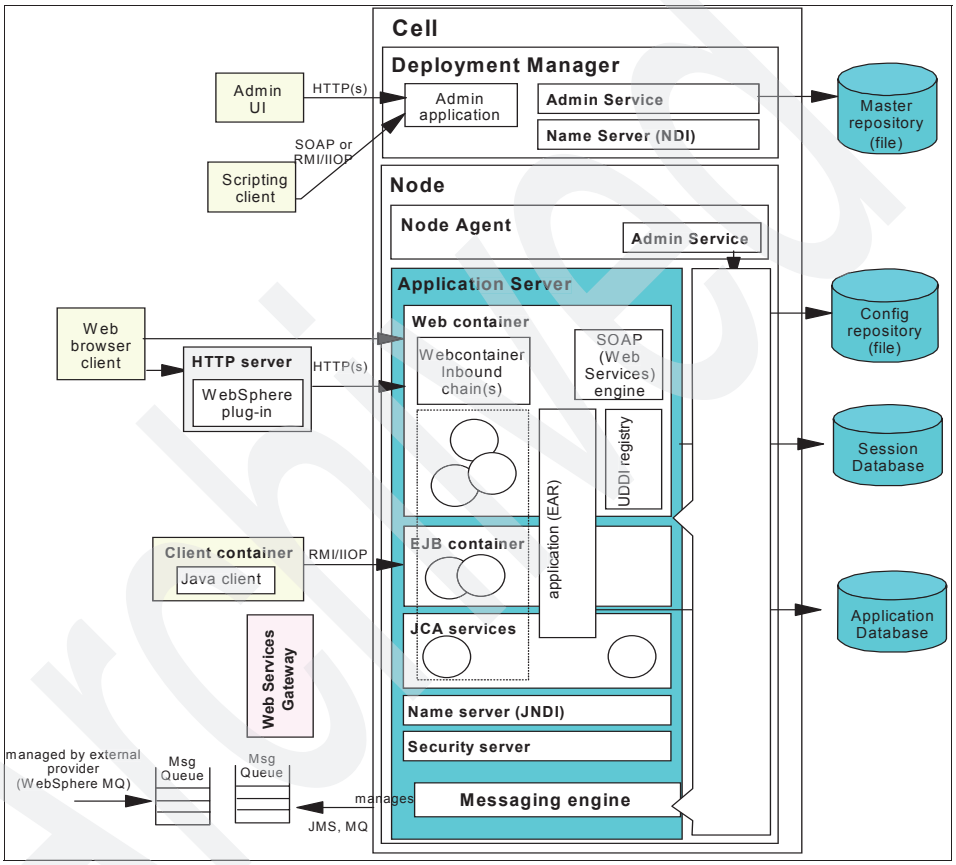


Figure 2-2 Distributed server environment

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.

Archived

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.

Archived

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

Archived

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 ActivitionSpec 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).

Archived

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.

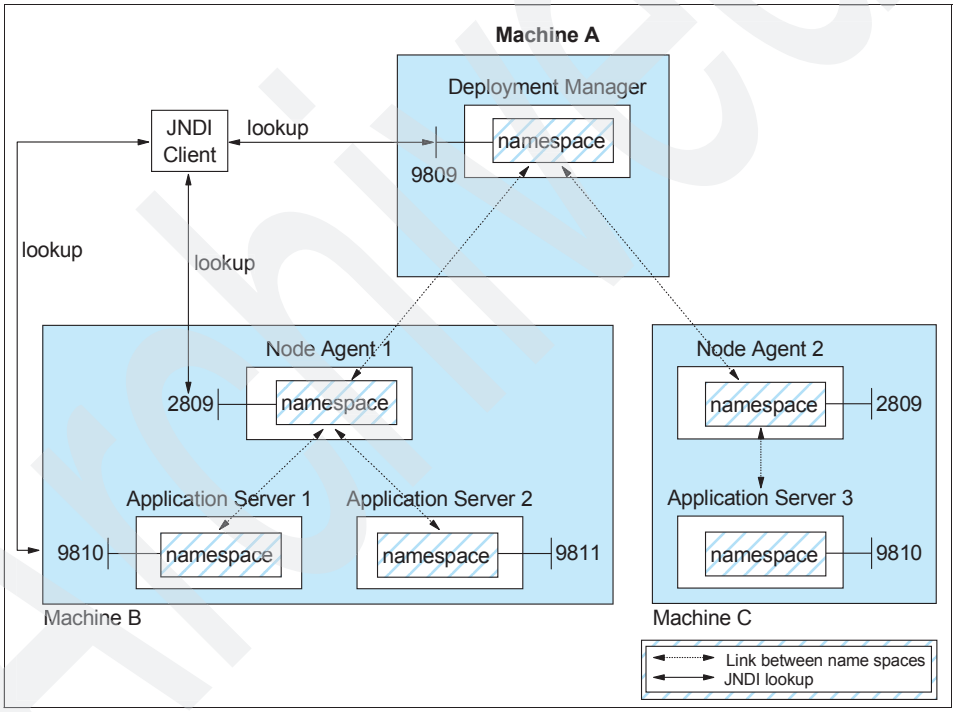


Figure 2-3 Naming topology

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
  ```

Archived

- 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

Archived

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

Archived

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`

Archive

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.

Archived

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.

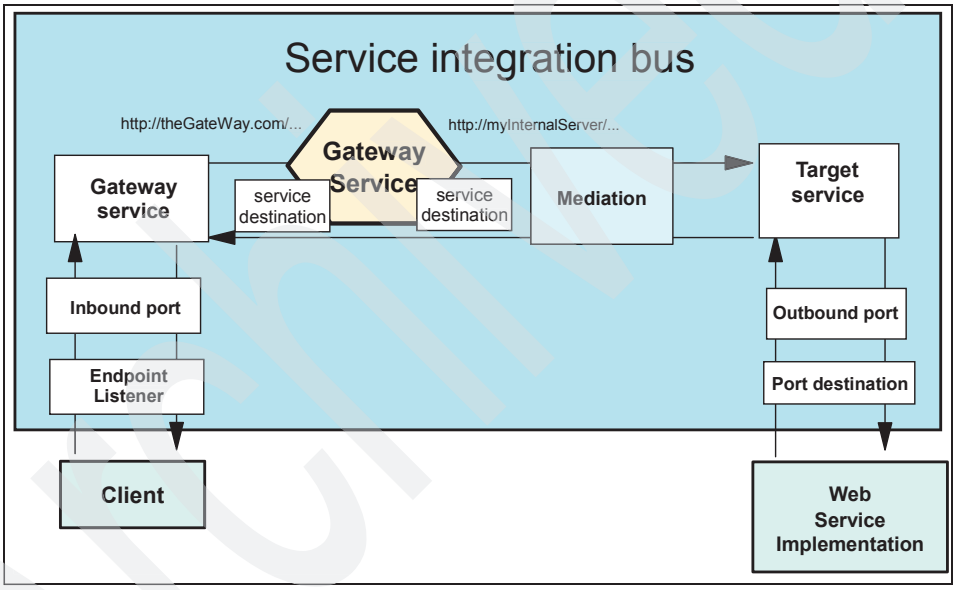


Figure 2-4 Exposing Web services through a gateway

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.

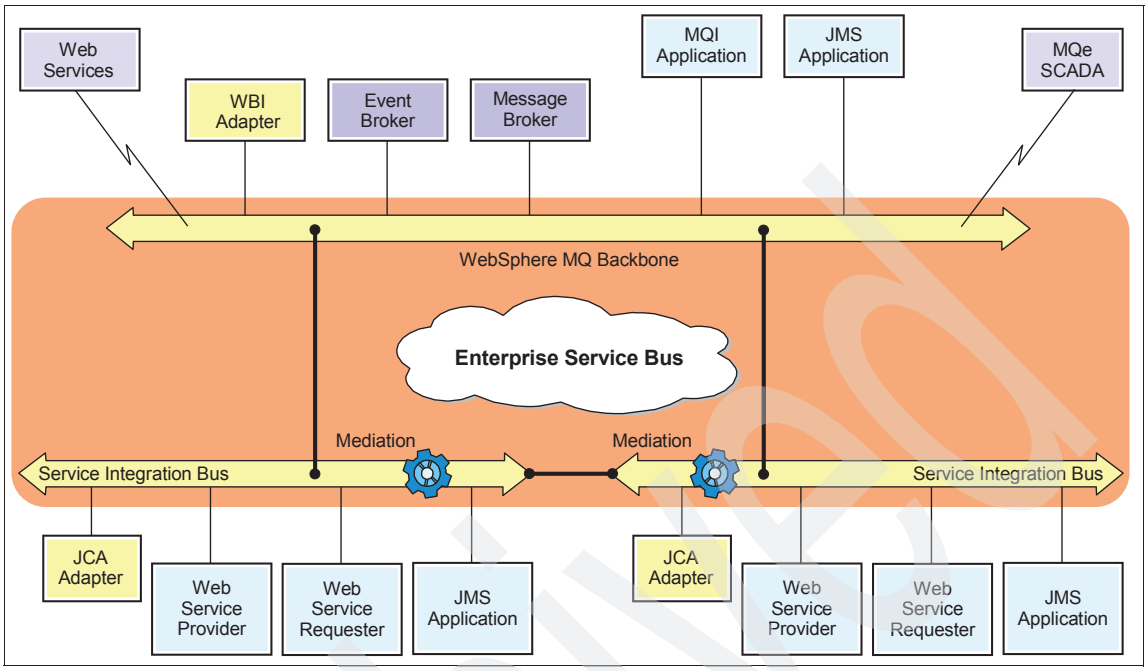


Figure 2-5 The Enterprise Service Bus

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 from where to 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.

Archived

- 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.

Archive

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

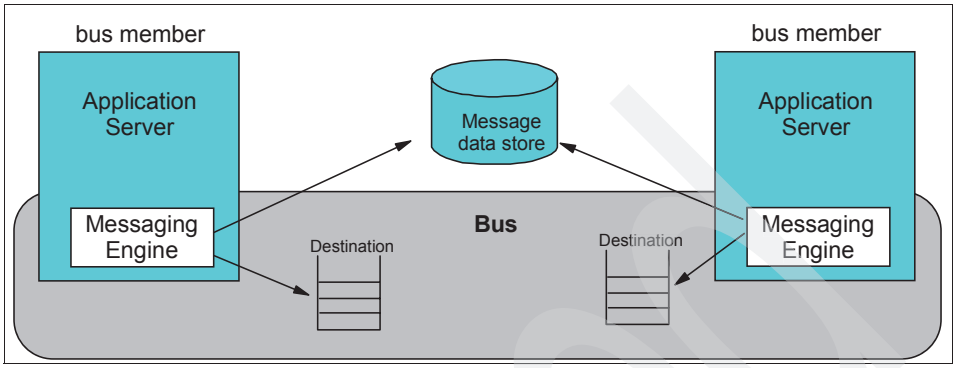


Figure 2-6 Service Integration Bus

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

Archived

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.

Archived

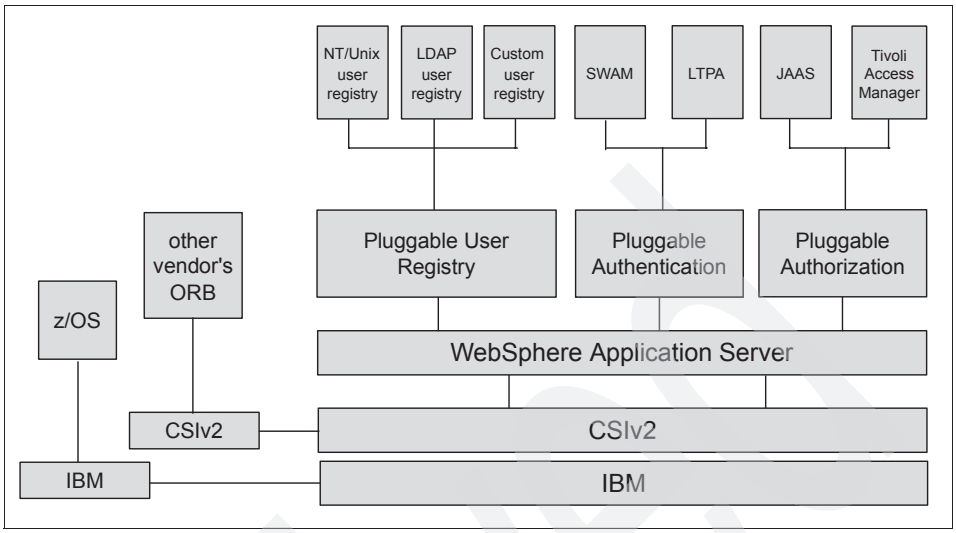


Figure 2-7 WebSphere Application Server security architecture

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 inactivate 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 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 SSLand 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.

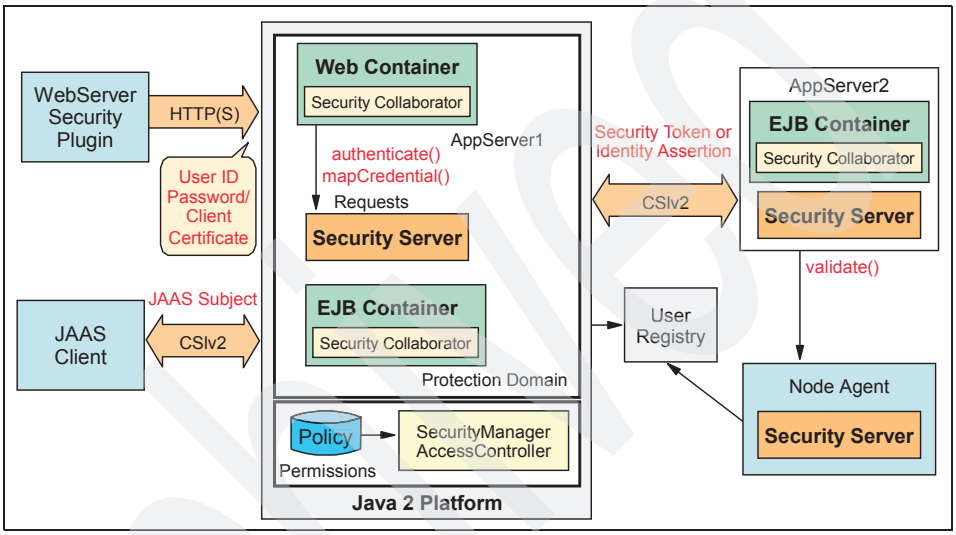


Figure 2-8 WebSphere Application Security components

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 (CSIv2) 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 CSIv2 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.

Archived

5. 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:

1. The administration client generates a request that reaches the server side ORB and JMX MBeans. The JMX MBeans represent managed resources.
2. 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:

1. A Java client generates a request that reaches the server side ORB.
2. The CSIv2 or IBM SAS interceptor performs authentication on the server side on behalf of the ORB, and sets the security context.
3. The server side ORB passes the request to the EJB container.
4. After submitting a request to the access-protected EJB method, the EJB container passes the request to the EJB collaborator.
5. The EJB collaborator reads the deployment descriptor from the EAR file and reads the user credentials from the security context.
6. Credentials and security information are passed to the security server, which validates user access rights and passes this information back to the collaborator.
7. 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.

Archived

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.

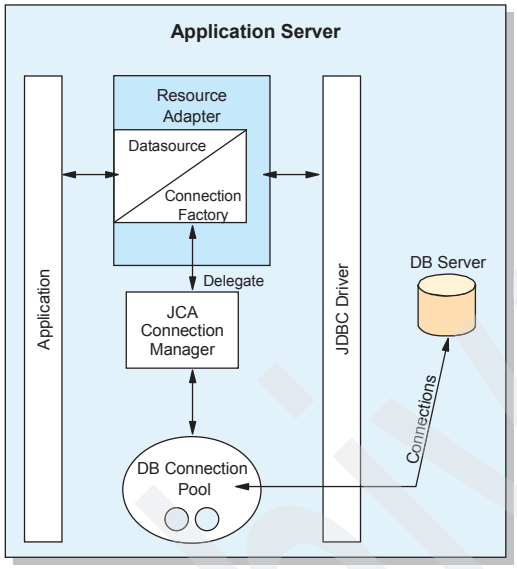


Figure 2-9 Resource adapter in J2EE connector architecture

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

Archived

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.

Archive

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.net.URLConnection classes. It enables communication between the application and a URL resource that is served by that particular protocol.

A URL provider named Default URL Provider is included in the initial WebSphere configuration. This provider 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.

Archived

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.

Archived

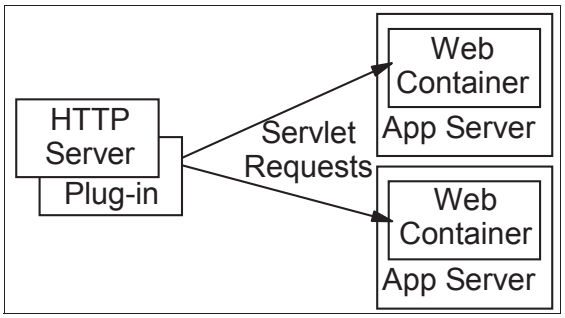


Figure 2-10 Plug-in (Web container) workload management

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:

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

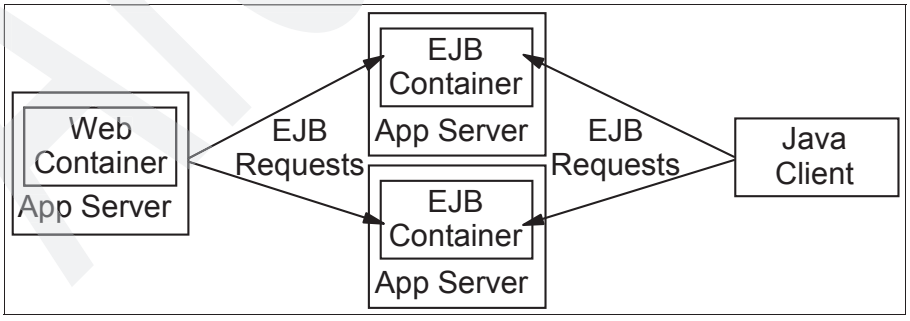


Figure 2-11 EJB workload management

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/IIOP, 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.

Archived

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.

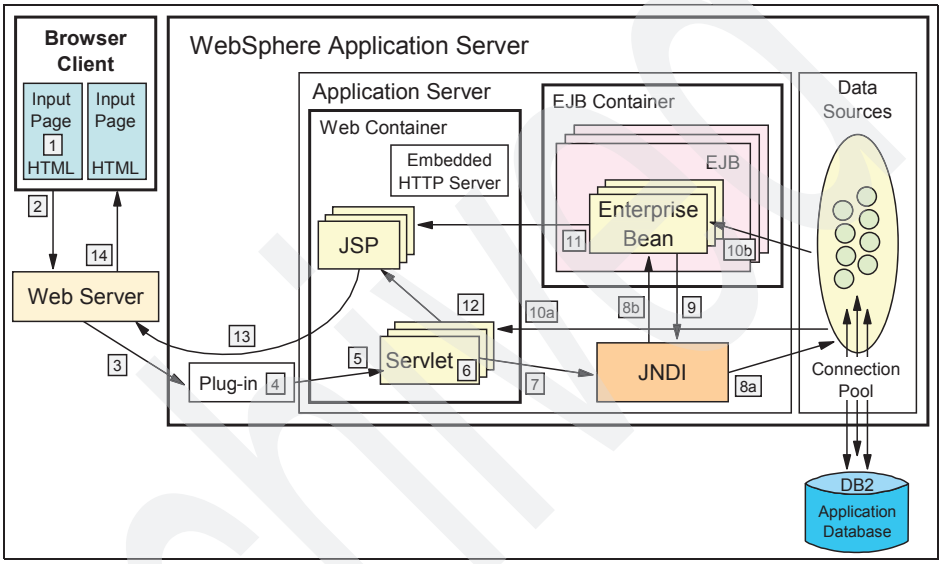


Figure 2-12 Application flow

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.

Archived

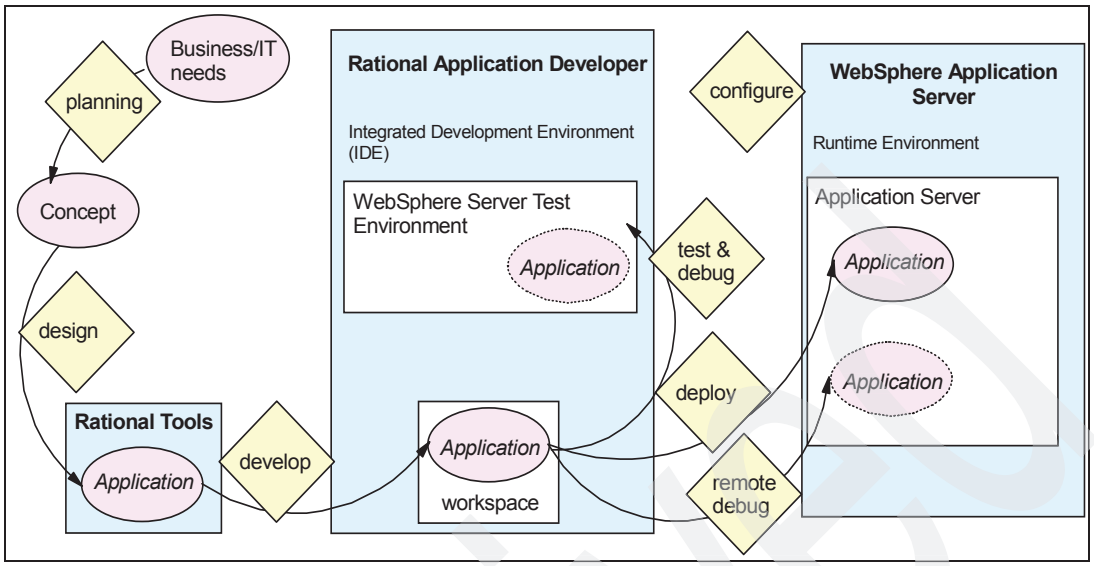


Figure 2-13 Develop and deploy

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

Archived

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

Archive

Feature	Base and Express V6	Network Deployment V6
J2EE 14. programming support[1] ► 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 ► 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 ► 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

Archived

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 ► 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

Abstract

Feature	Base and Express V6	Network Deployment V6
Web services ► 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]		
► 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.

Archived

Archived

3

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

© Copyright IBM Corp. 2005. All rights reserved.

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.

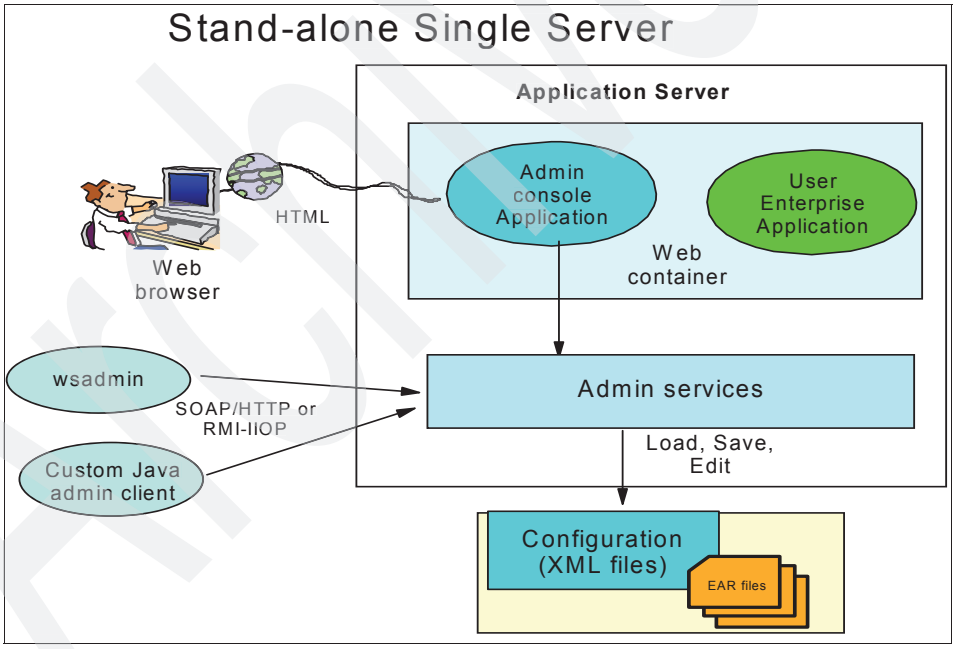


Figure 3-1 Managing a single-server installation

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.

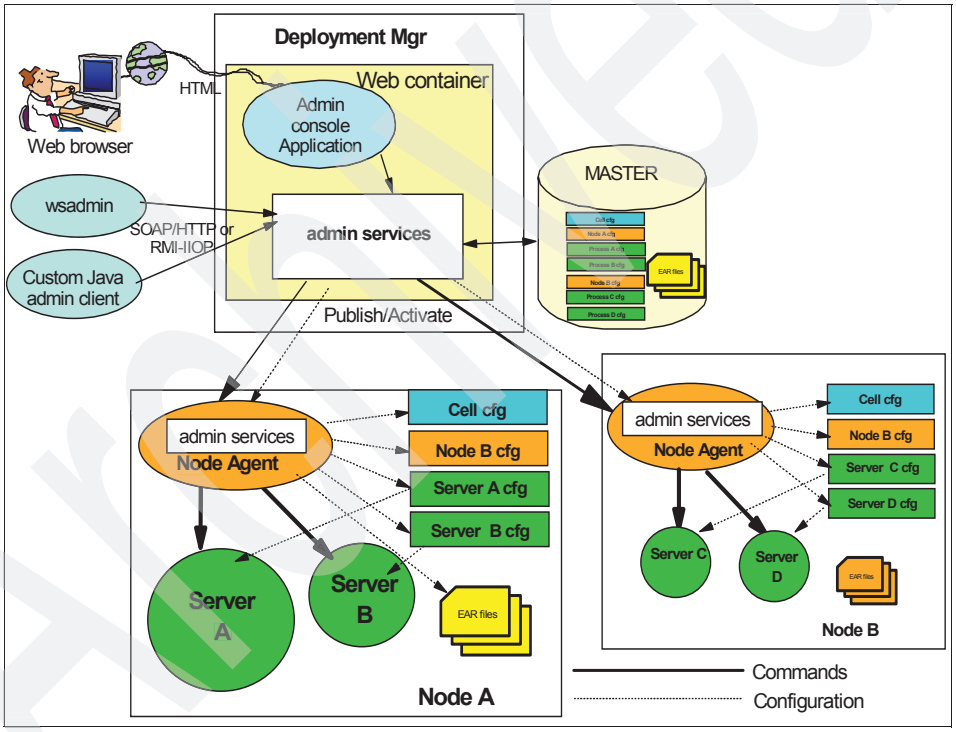


Figure 3-2 Managing a multi-server installation

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 managable from third party management products that delivers J2EE management capabilities.

Archived

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.

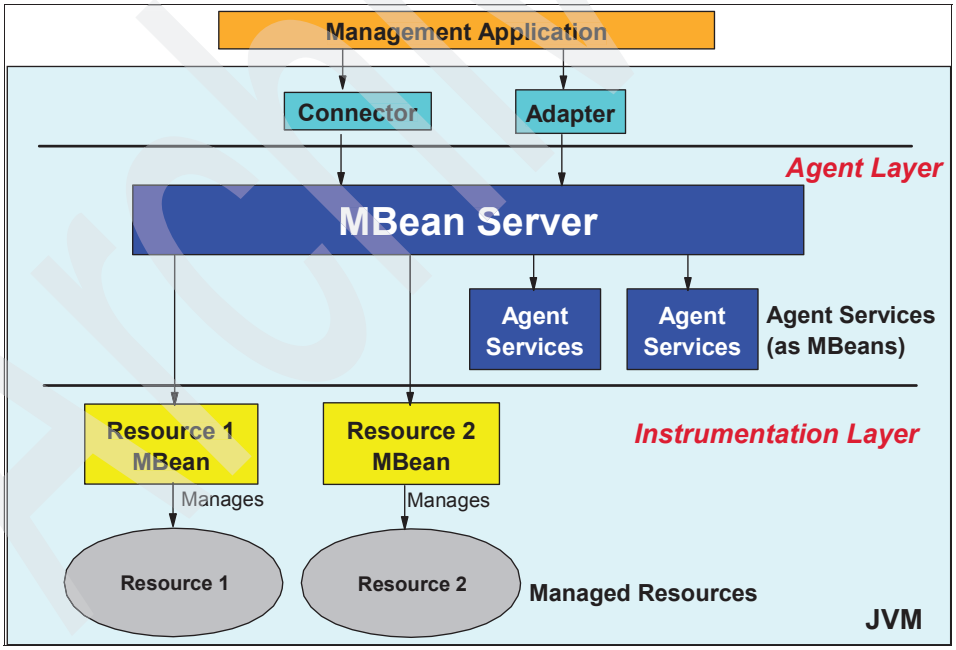


Figure 3-3 JMX architecture

Archived

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.

Archived

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.

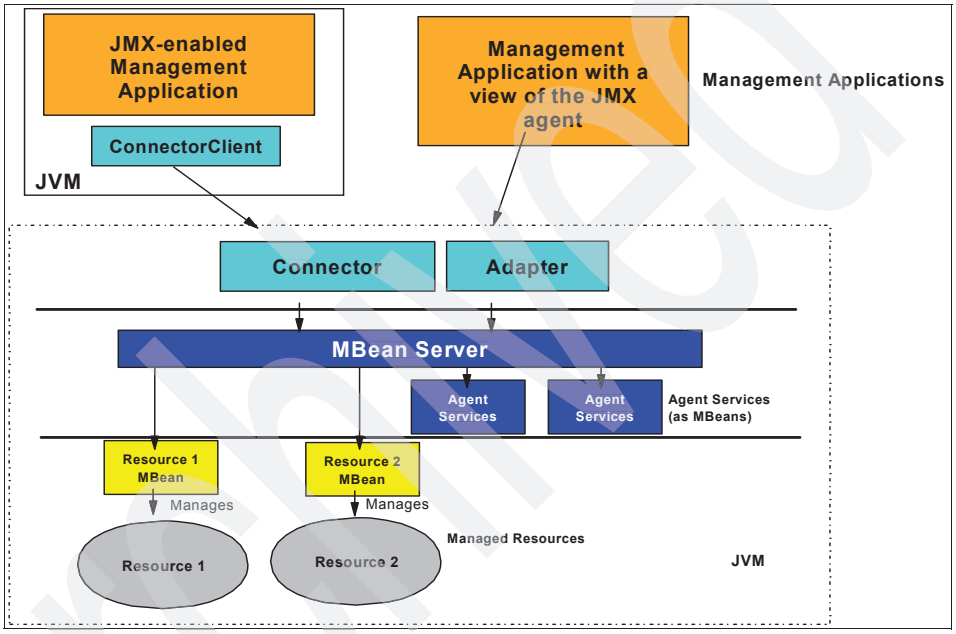


Figure 3-4 JMX connectors and adapters

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.

Abstract

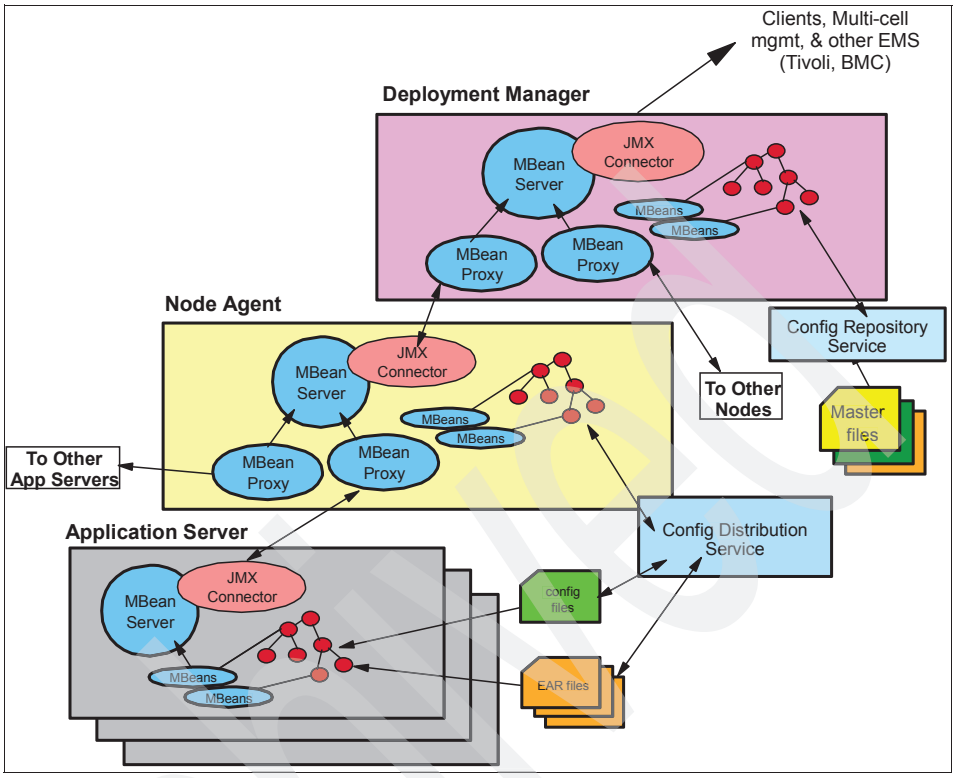


Figure 3-5 JMX distributed administration

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.

Archive

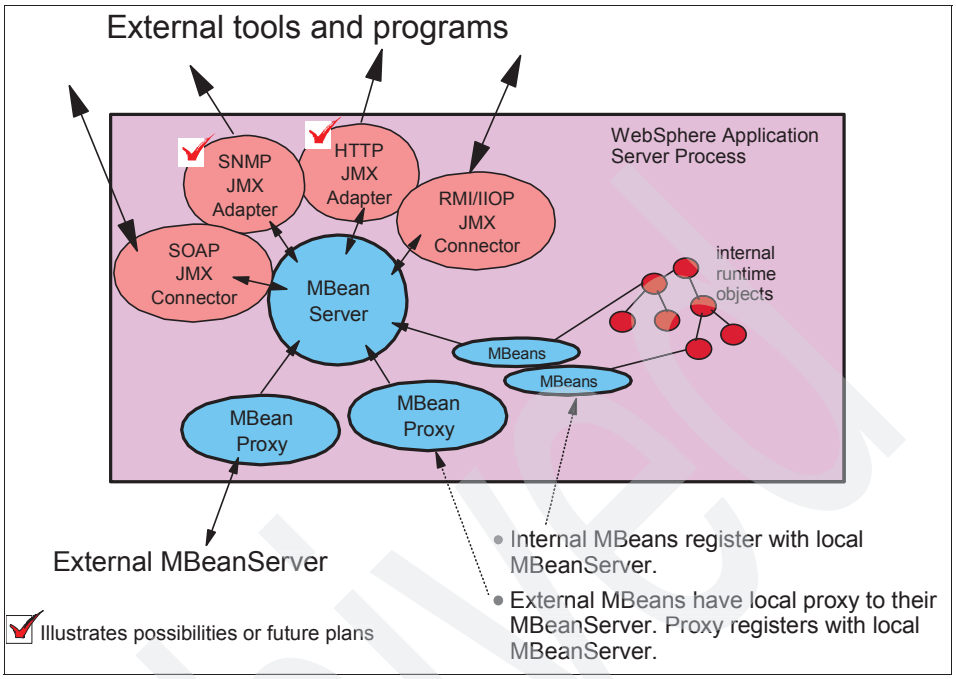


Figure 3-6 JMX architecture

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

Archive

- 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.

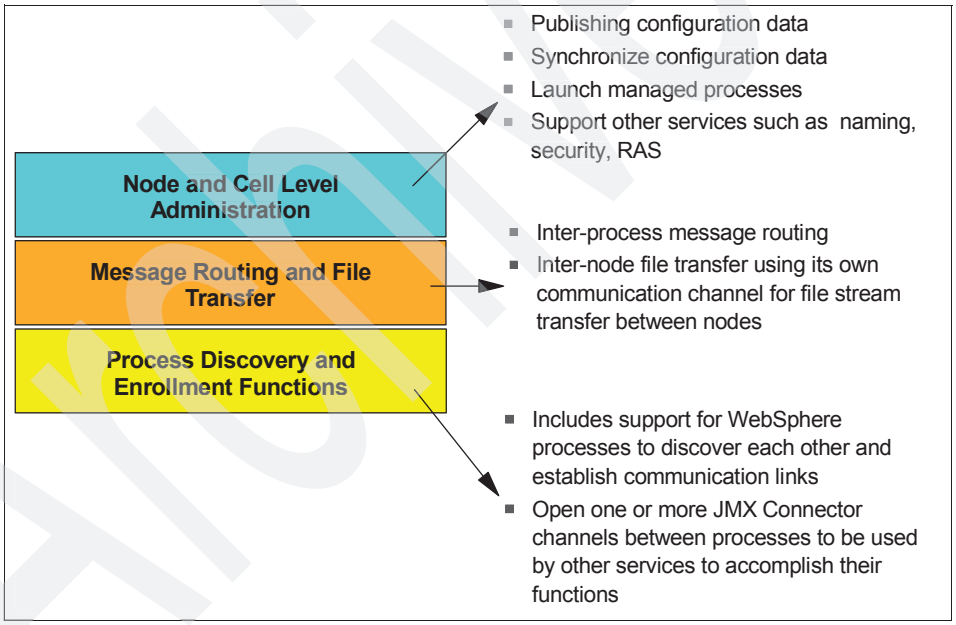


Figure 3-7 Layers of distributed administration services

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.

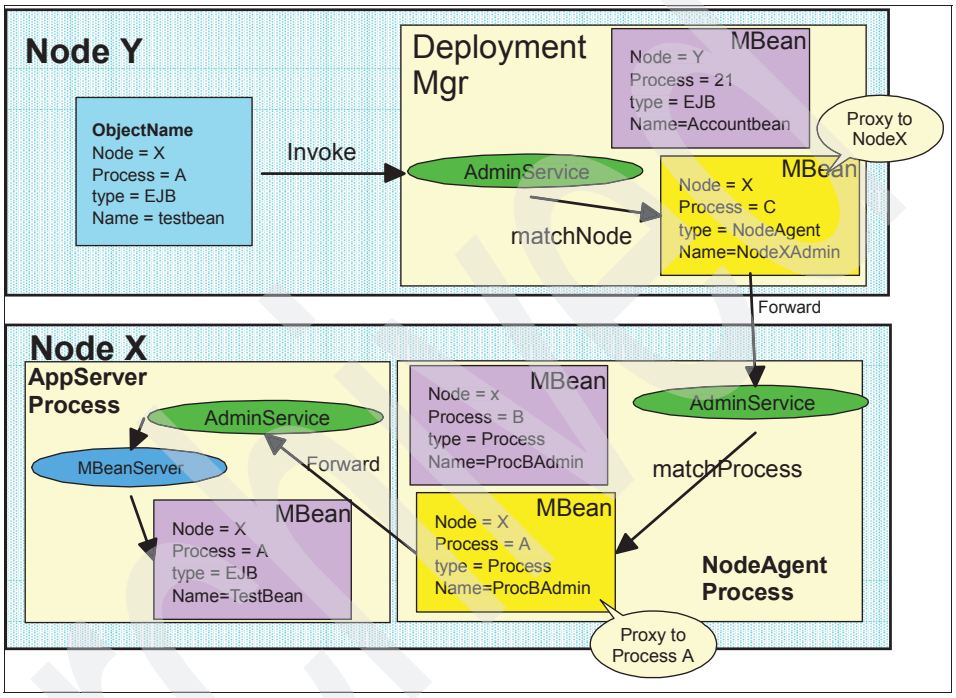


Figure 3-8 Distributed administration message routing

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.

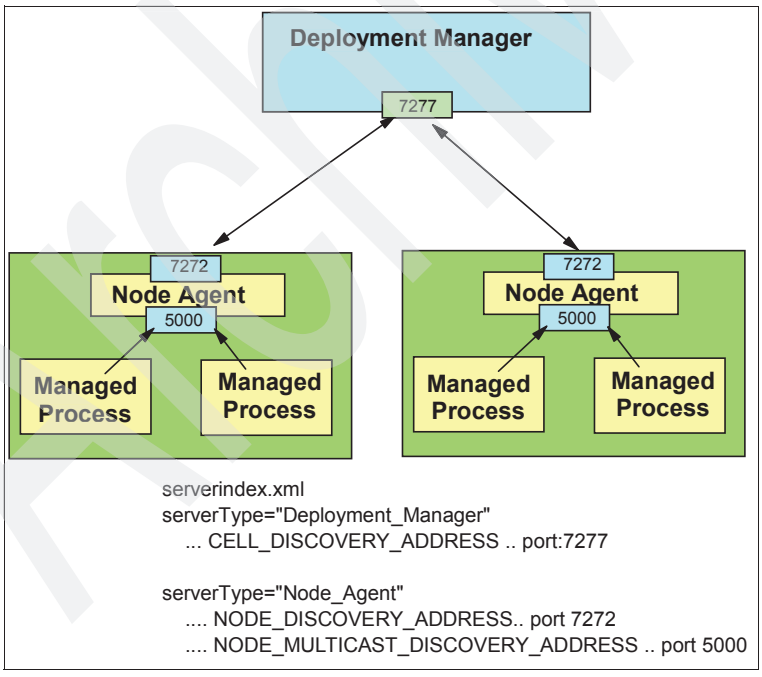


Figure 3-9 Distributed discovery process

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 coNnfiguration 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.

Archived

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.

Archived

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 Reseting the master cell repository

```
C:\WebSphere\AppServer\profiles\DmgrO1\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\AppSrvO1\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]
```

Archived

```
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 carlavm2
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 carlavm2: 8879
ADMU0016I: Synchronizing configuration between node and cell.
ADMU0402I: The configuration for node AppSrvrNode has been synchronized with
           Deployment Manager carlavm2: 8879
```

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

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

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

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.

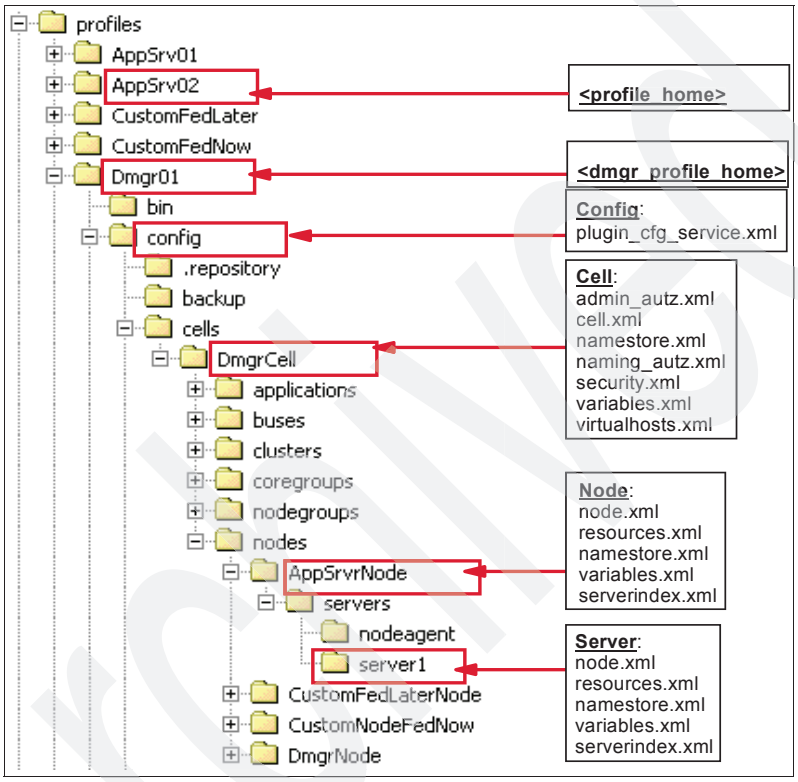


Figure 3-10 Repository directory structure

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:

Archive

- 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

Archive

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.

Archive

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

Archived

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.

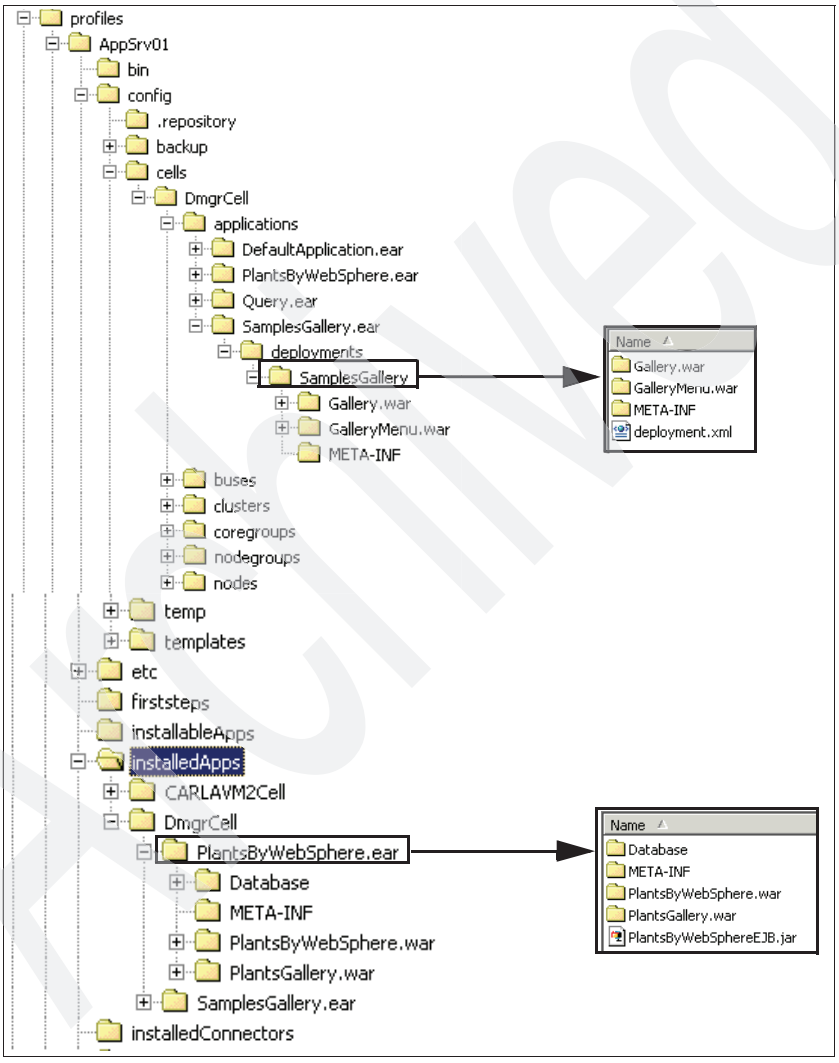


Figure 3-11 Location of application data files

Archived

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.

Archived

Archived

4

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.

© Copyright IBM Corp. 2005. All rights reserved.

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.

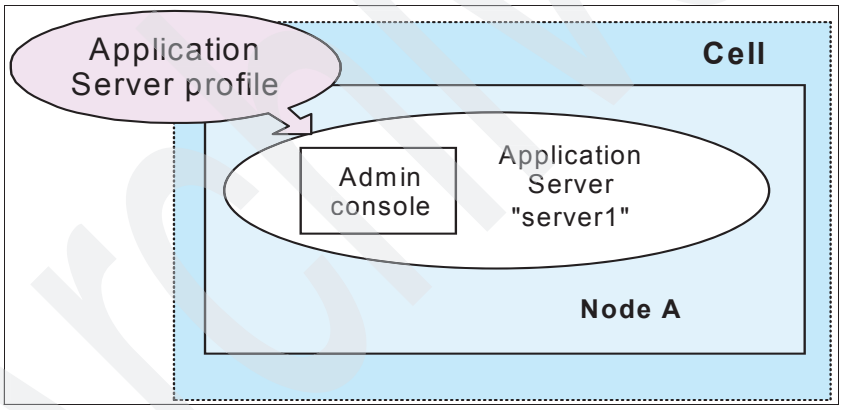


Figure 4-1 System management topology: Standalone server (Base and Express)

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.

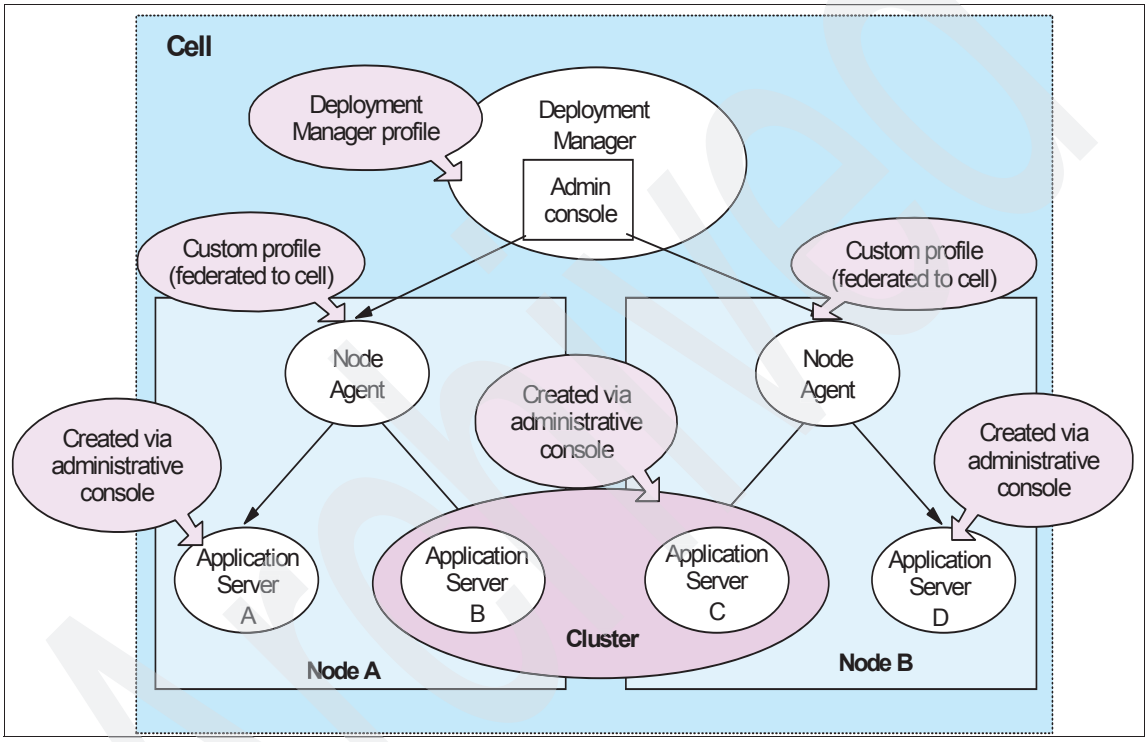


Figure 4-2 System management topology: Network Deployment

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 configurate 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.

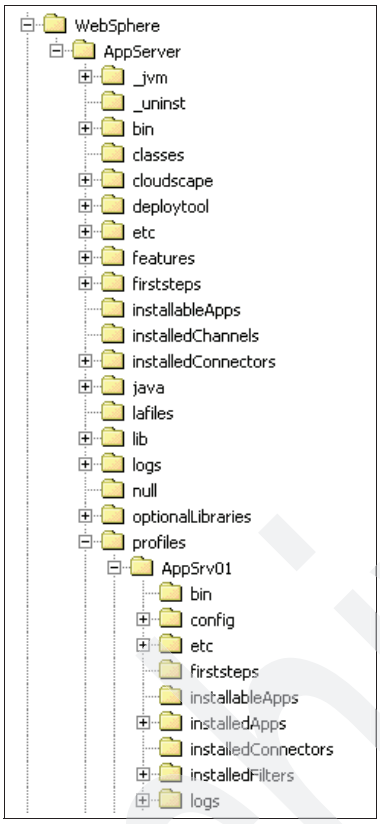


Figure 4-3 Directory structure

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:

Archived

- 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.

Archive

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 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.

Archived

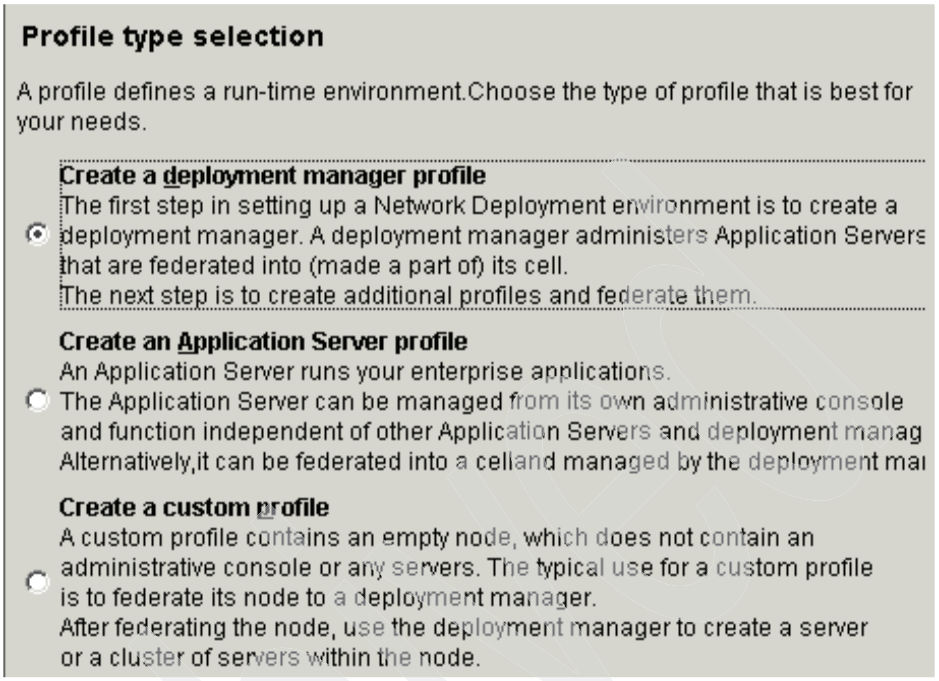


Figure 4-4 Creating a profile: Profile type selection

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**.

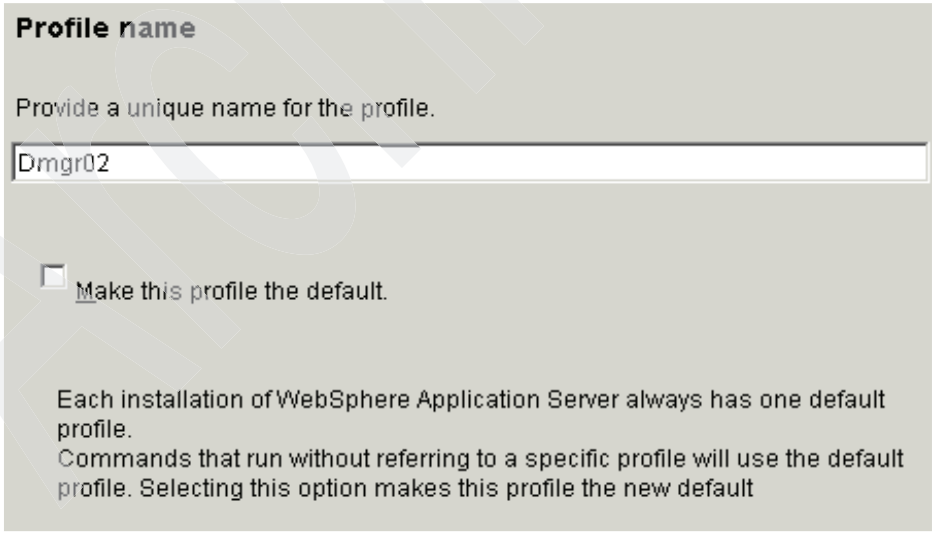


Figure 4-5 Creating a deployment manager profile: Enter a name

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

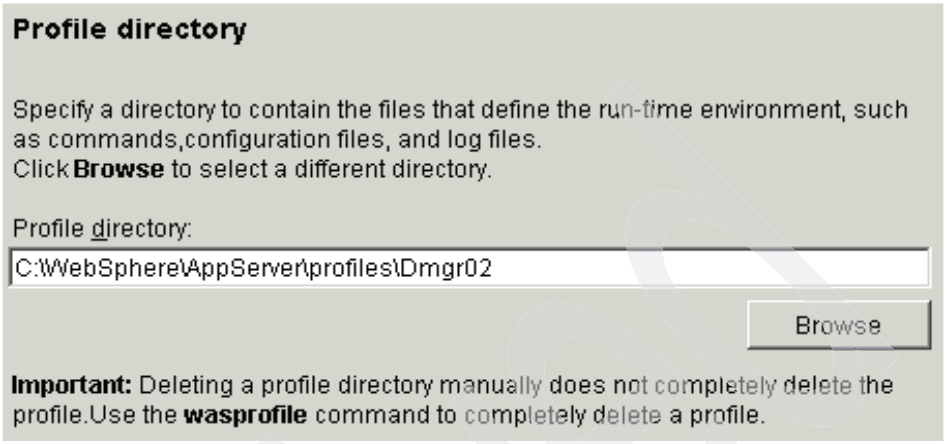


Figure 4-6 Creating a deployment manager profile: Enter a profile directory

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.

Archive

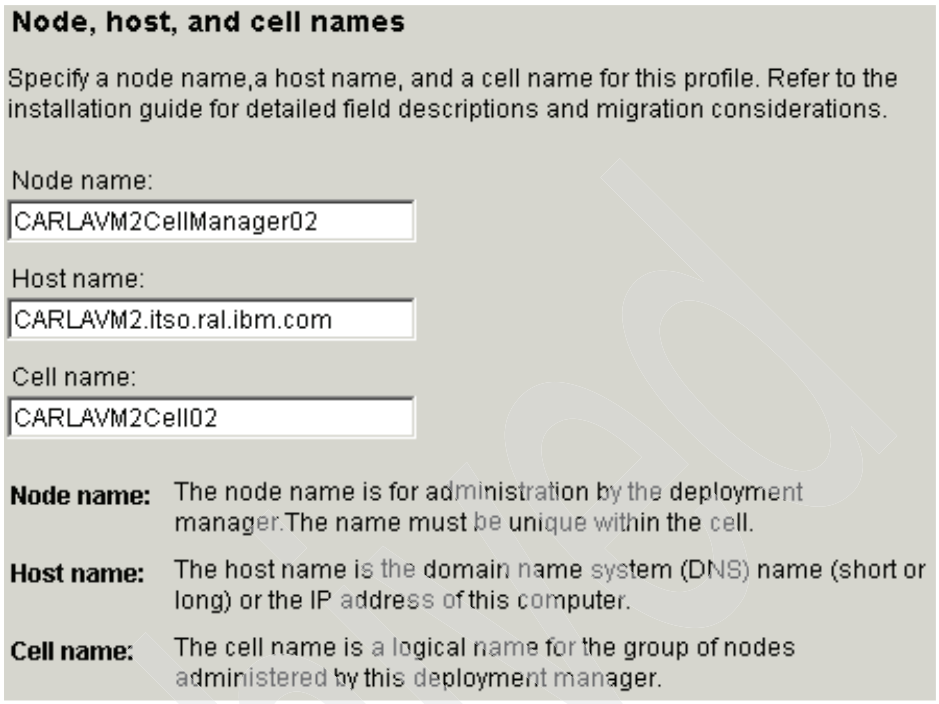


Figure 4-7 Creating a deployment manager profile: Enter cell, host, and node names

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.

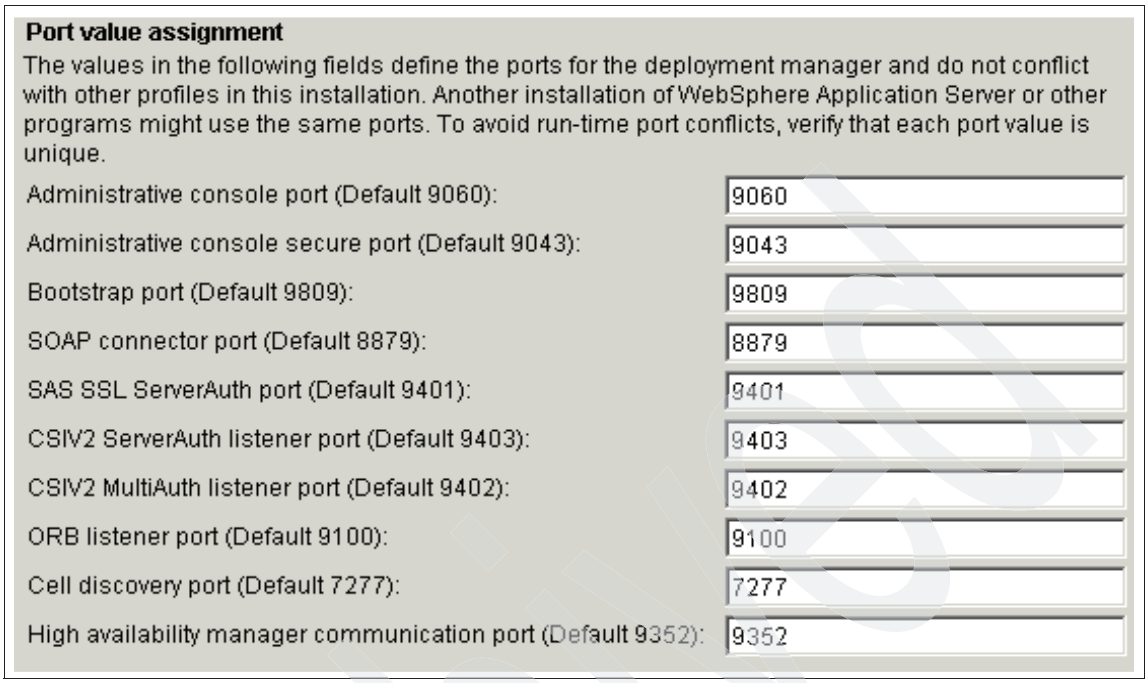


Figure 4-8 Creating a deployment manager profile: Select ports

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,

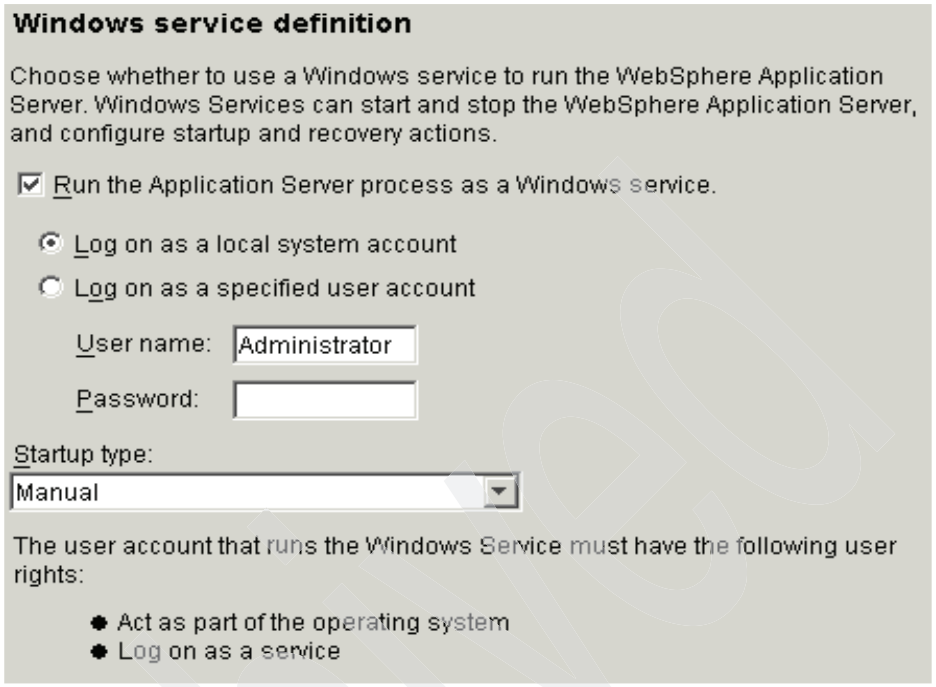


Figure 4-9 Creating a deployment manager profile: Run as a Windows service

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.

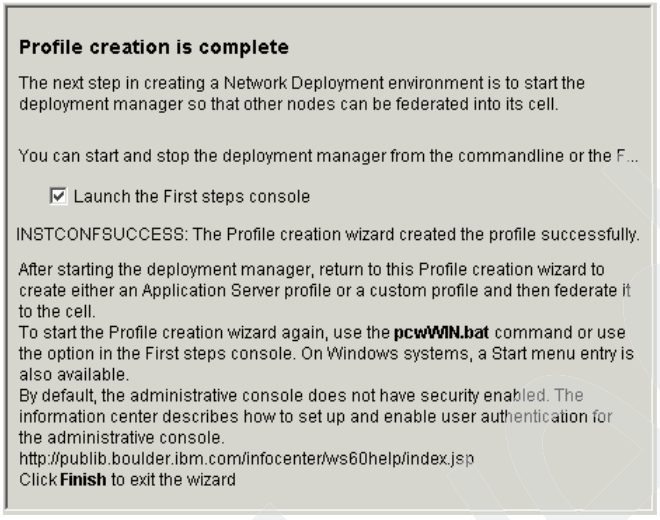


Figure 4-10 Creating a deployment manager profile: Finish

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

Archived

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**.

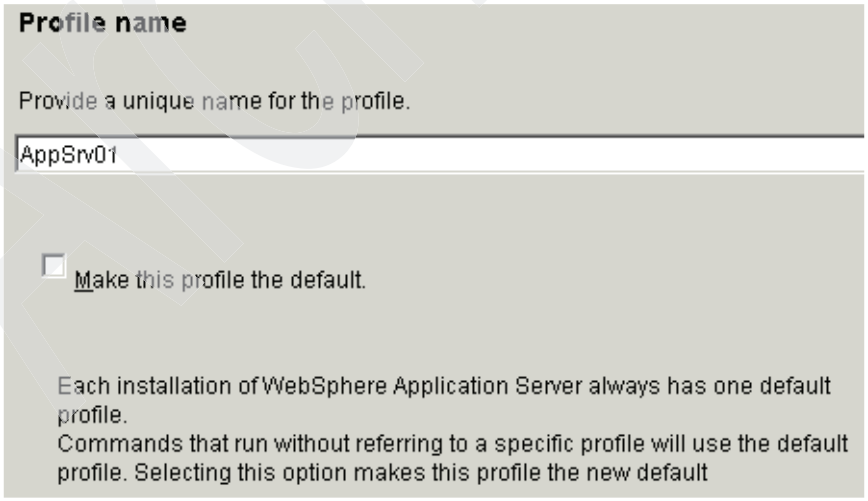


Figure 4-12 Creating an application server profile: Enter a name

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

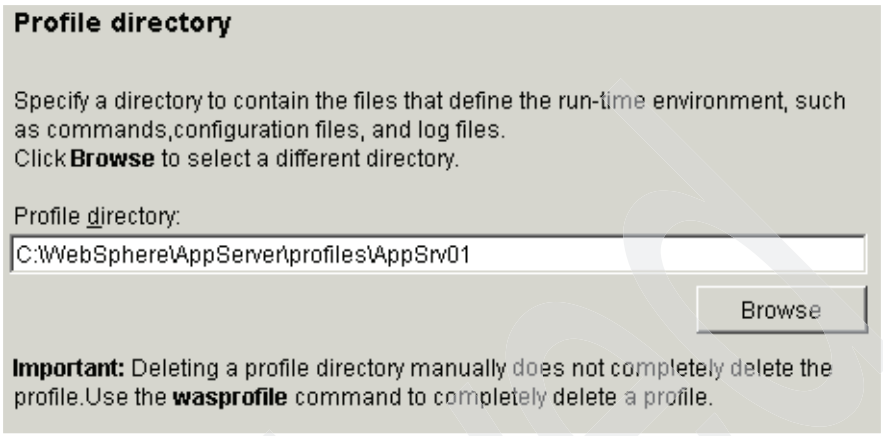


Figure 4-13 Creating an application server profile: Enter a profile directory

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**.

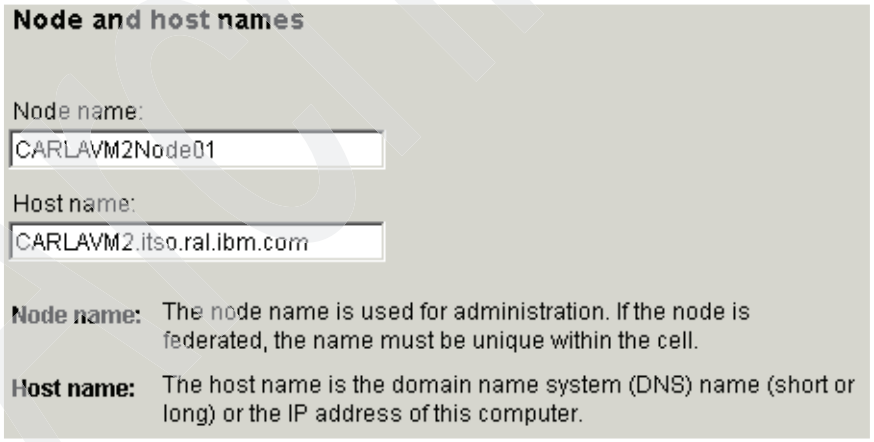


Figure 4-14 Creating an application server profile: Enter host and node names

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.

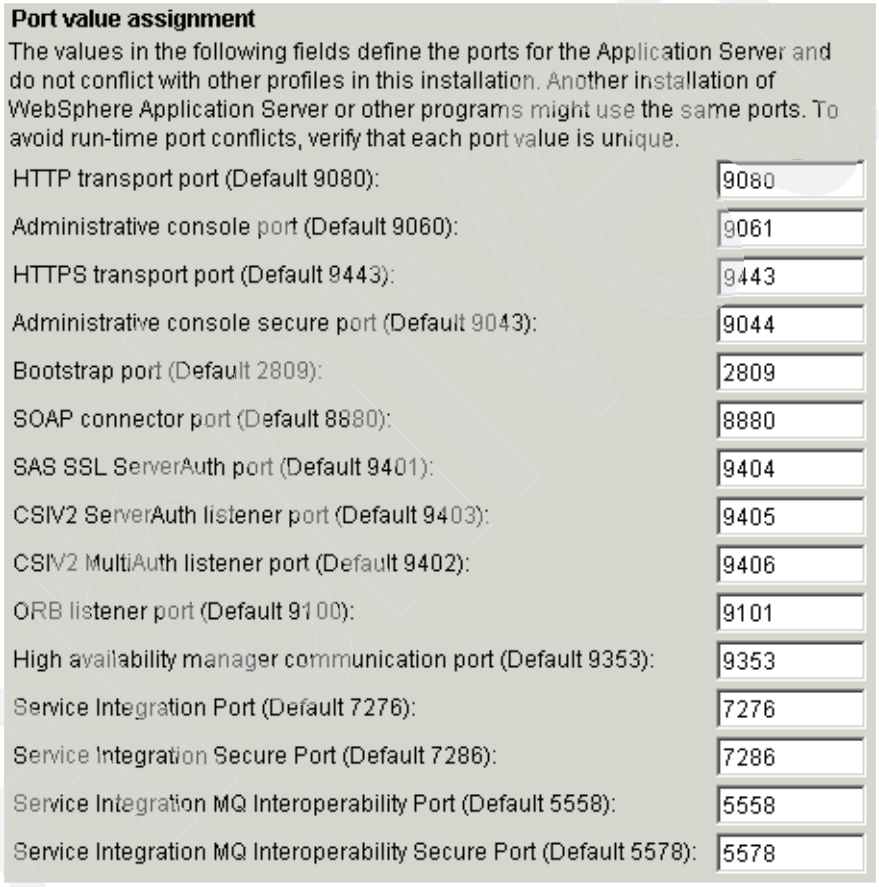


Figure 4-15 Creating an application server profile: Select ports

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.

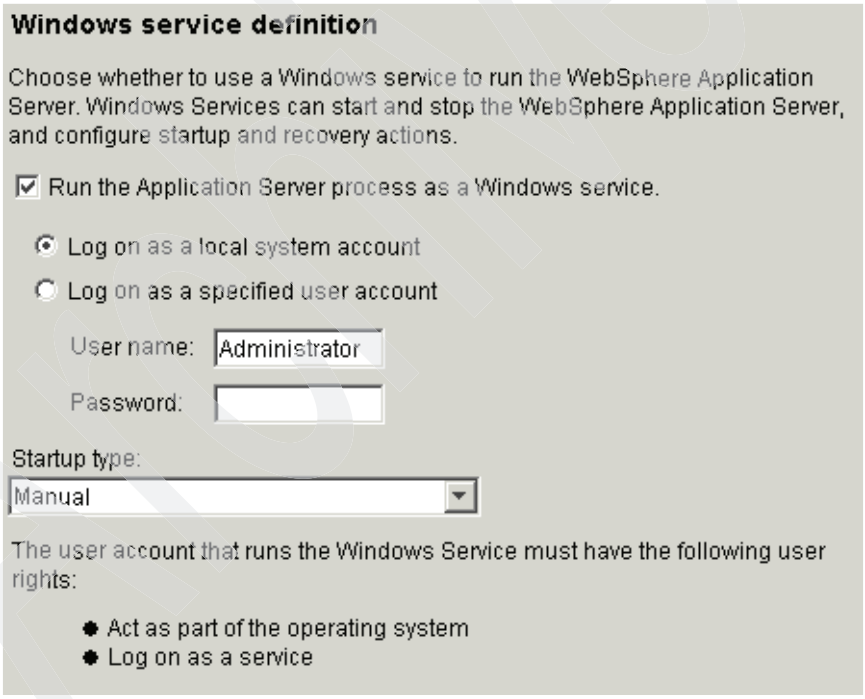


Figure 4-16 Creating an application server profile: Run as a service

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.

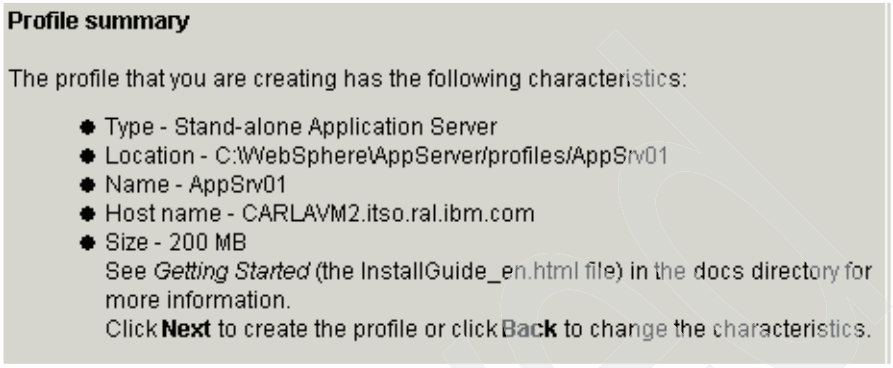


Figure 4-17 Creating a deployment manager profile: Finish

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.

Archive

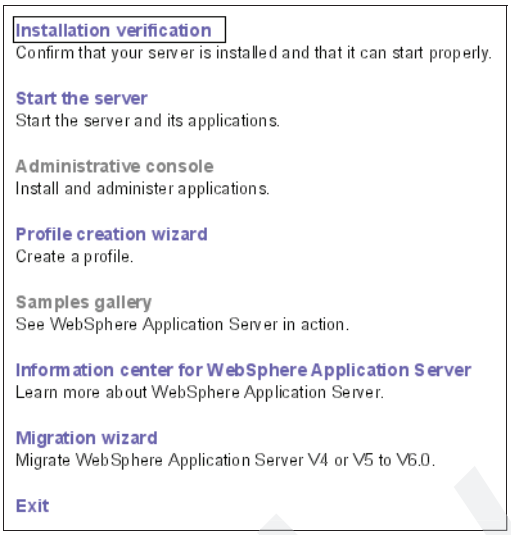


Figure 4-18 Application server First Steps menu

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:

Archived

```
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.

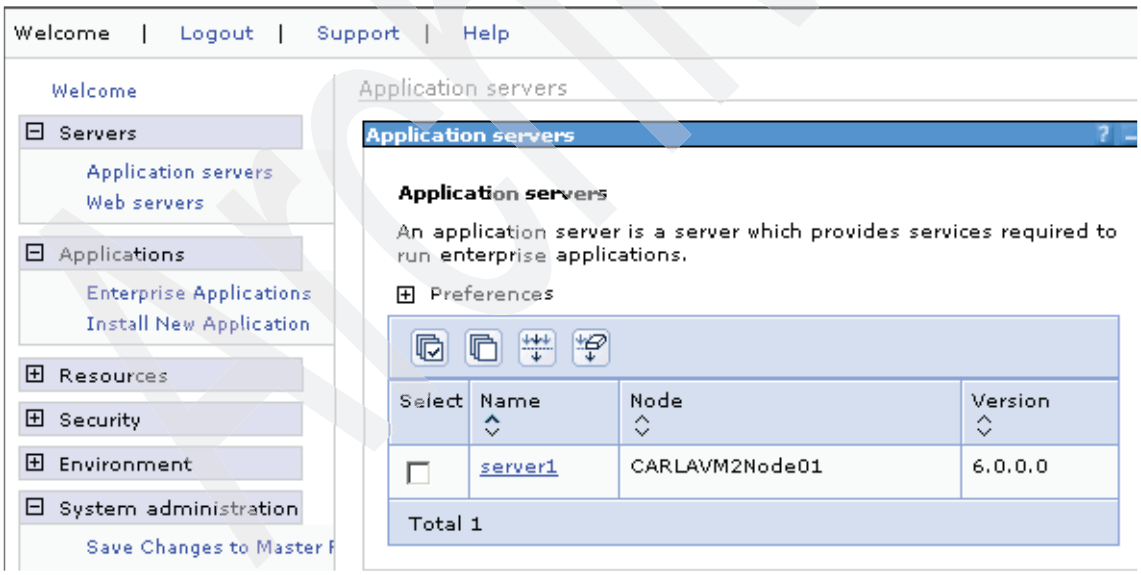


Figure 4-19 Application server defined by the application server profile

   b. Enterprise applications

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

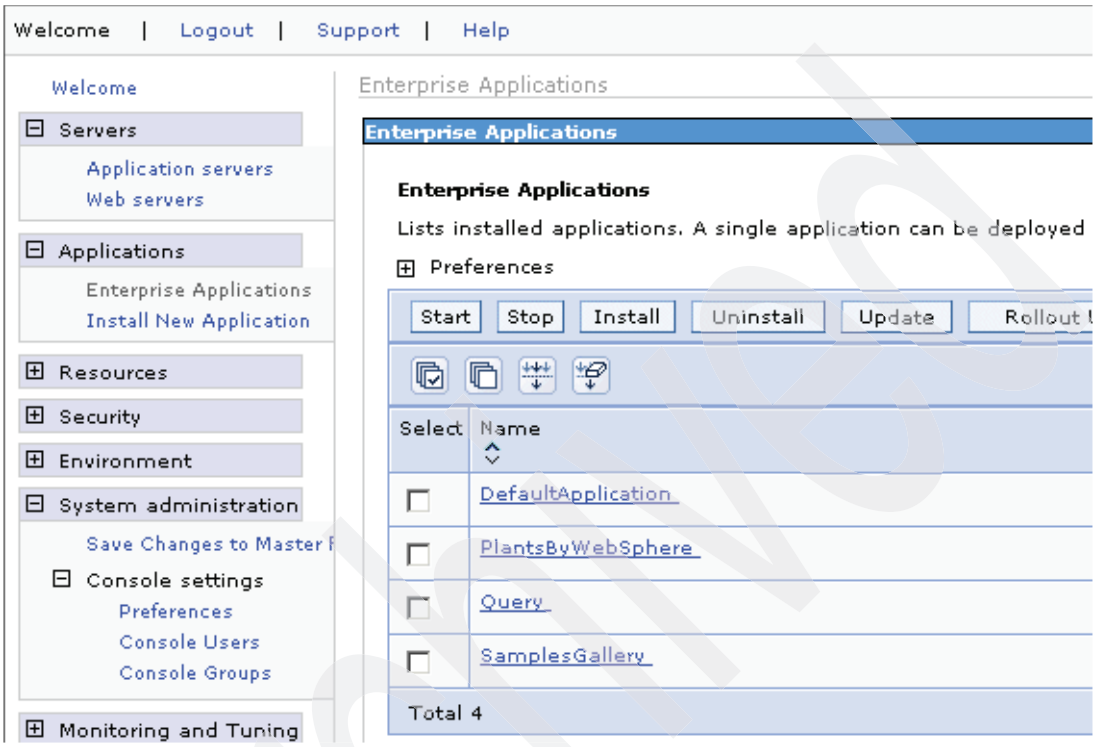


Figure 4-20 Applications installed on server1

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.

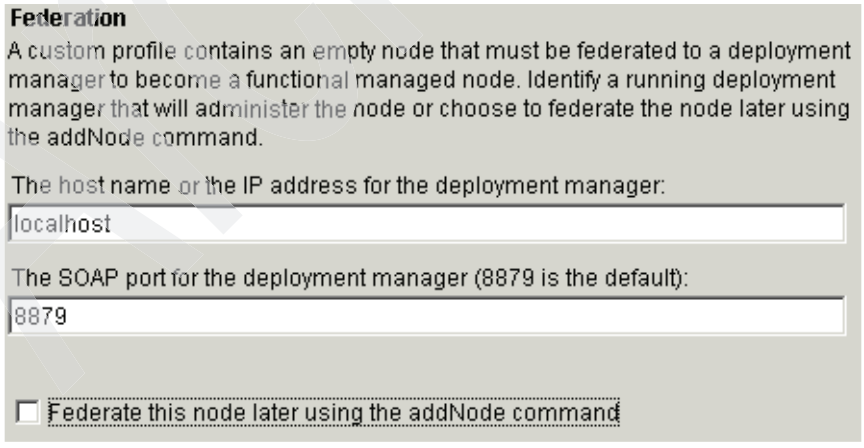


Figure 4-21 Creating a custom profile: Federate now or later

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**.

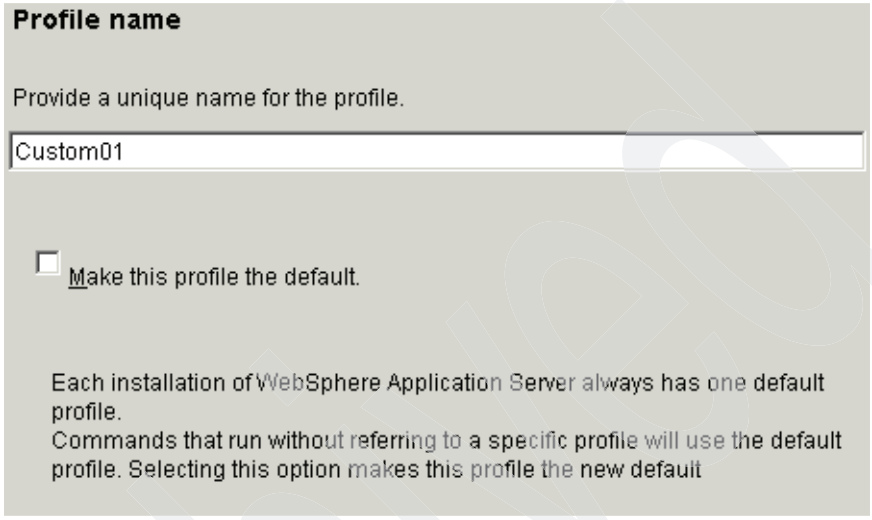


Figure 4-22 Creating a custom profile: Enter a name

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

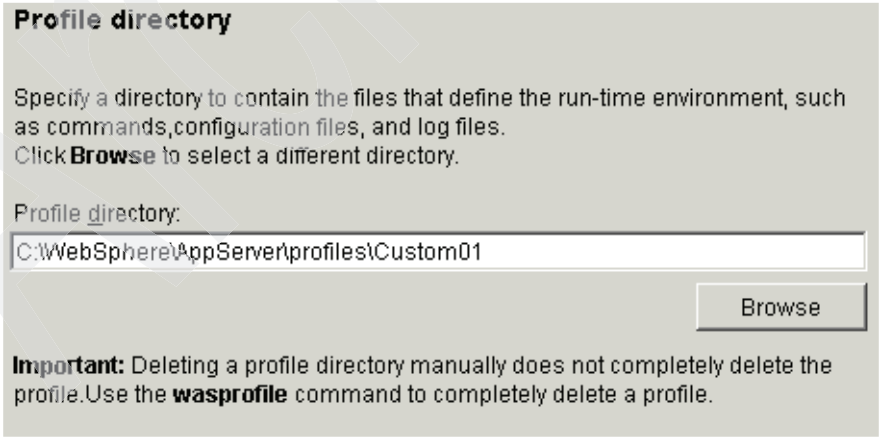


Figure 4-23 Creating a custom profile: Enter a profile directory

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**.

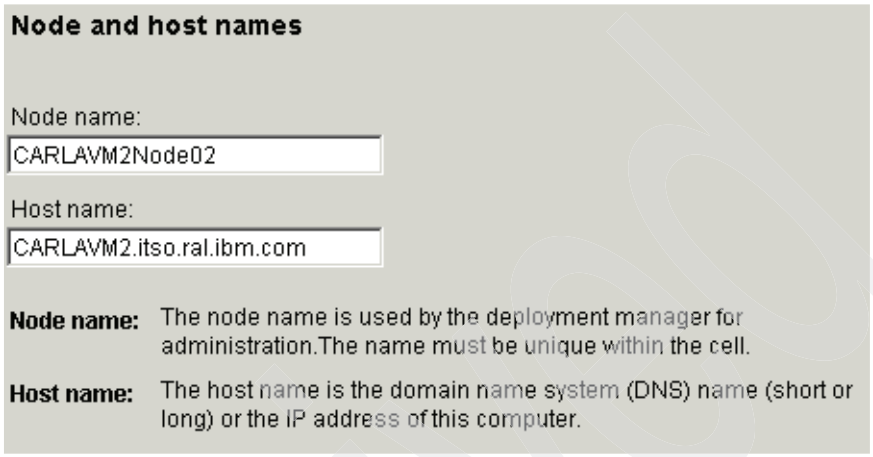


Figure 4-24 Creating a custom profile: Enter host, and node names

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.

Archive

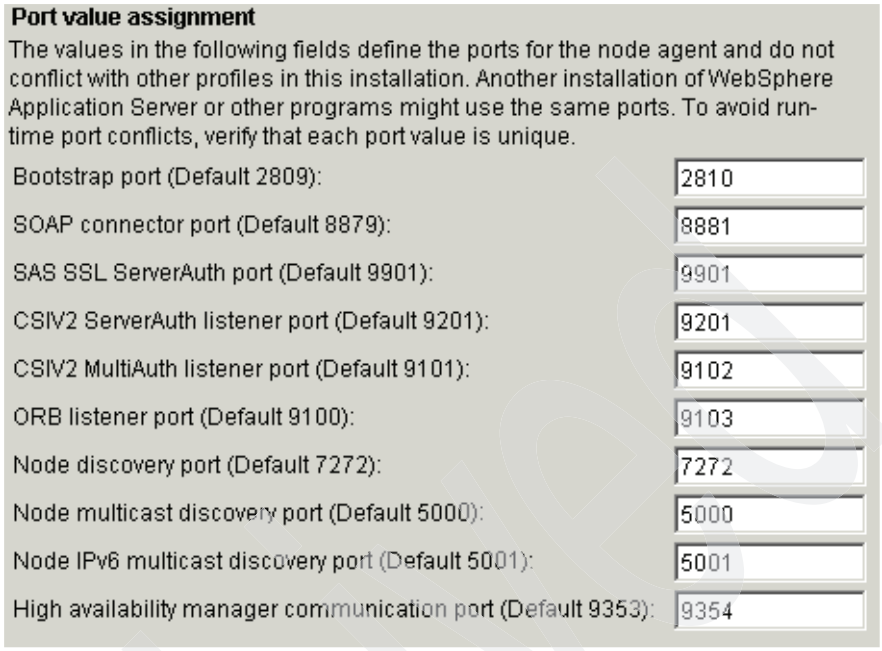


Figure 4-25 Creating a custom profile: Select ports

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.

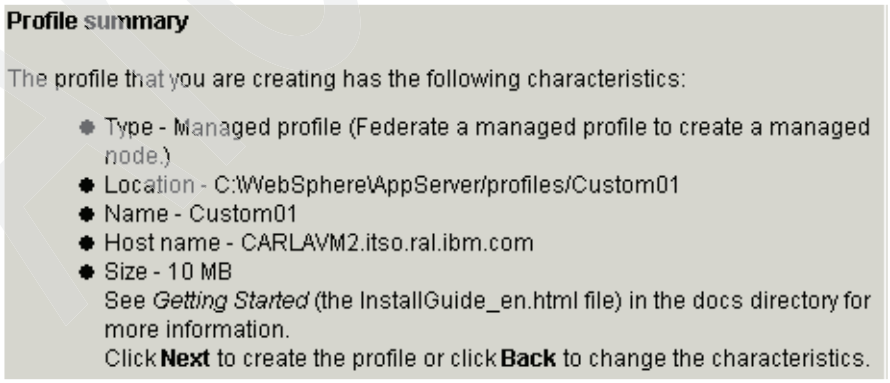


Figure 4-26 Creating a custom profile: Summary

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.

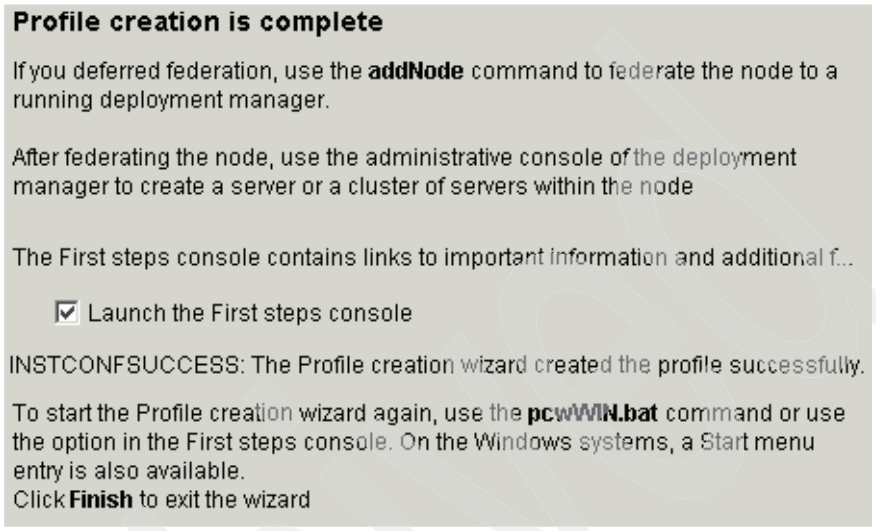


Figure 4-27 Creating a custom profile: Finish and launch First Steps

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.

Archived



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.

Archived

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 carlavm2 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 carlavm2:8879.
ADMU0009I: Successfully connected to Deployment Manager Server: carlavm2:8879
ADMU0507I: No servers found in configuration under:

C:\WebSphere\AppServer/profiles/CustomFedLater\config/cells/CARLAVM2C
ell/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
ell/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
```

Archived

```
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.

Archived

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**.

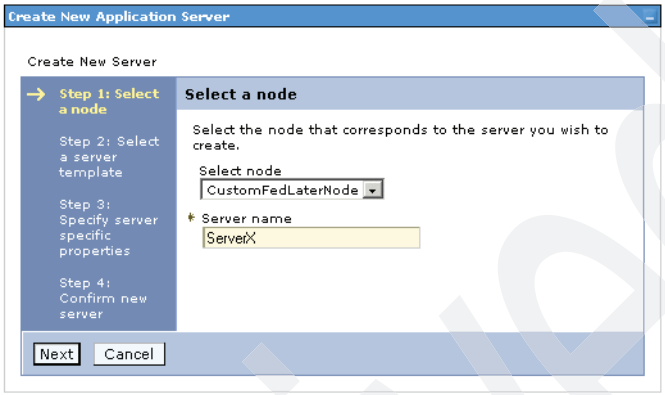


Figure 4-30 Creating a new server: Enter a node and name

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.

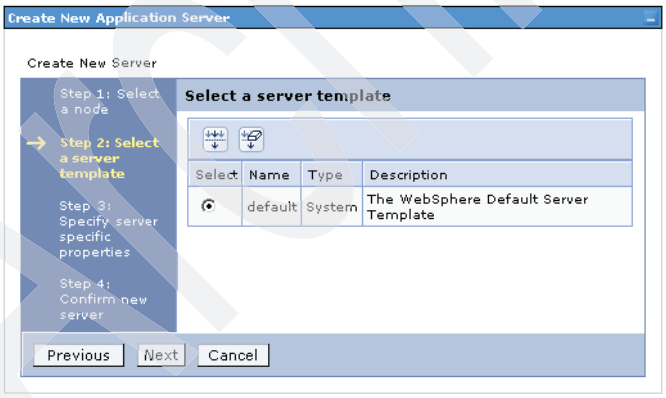


Figure 4-31 Creating a new server: Select a template

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.

Archived

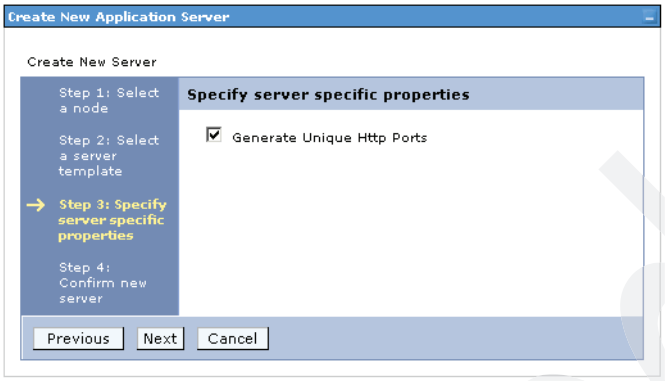


Figure 4-32 Creating a new server: Generate unique ports

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

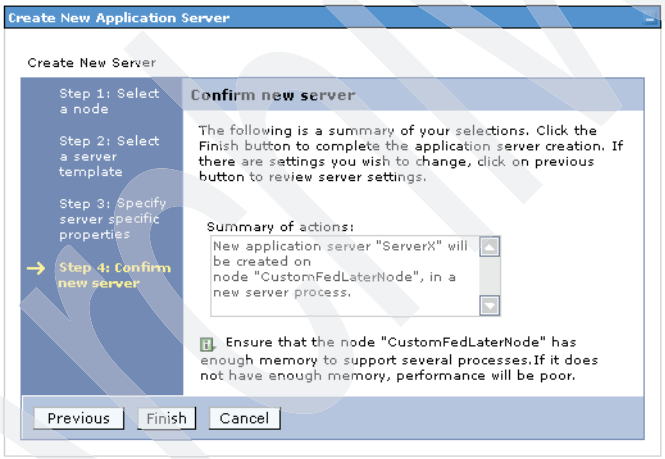


Figure 4-33 Creating a new server: Summary and finish

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**.

Archived

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.

Archived

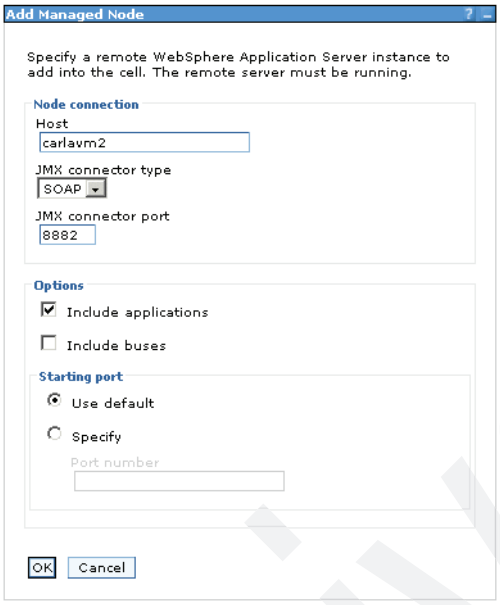


Figure 4-34 Adding a standalone application profile to a cell

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**.

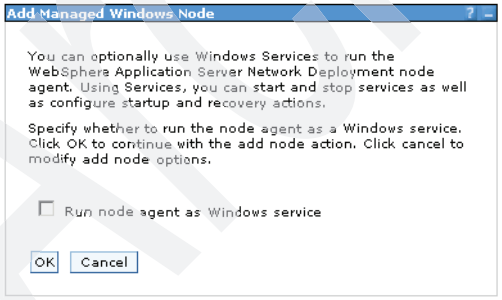


Figure 4-35 Run a node agent as a Windows service

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.

Archived

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 *sanodel* 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\DmgrO1\bin>startmanager
ADMUO116I: Tool information is being logged in file
           C:\WebSphere\AppServer\profiles\DmgrO1\logs\dmgr\startServer.log
ADMUO128I: Starting tool with the DmgrO1 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

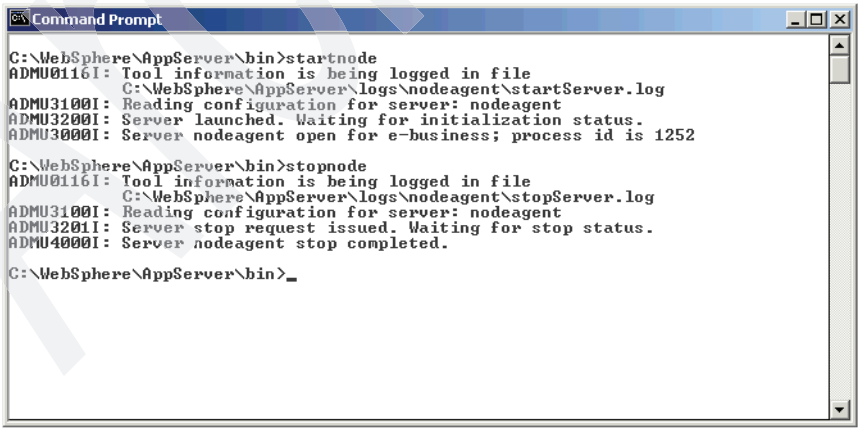


Figure 4-36 Starting and stopping the node agent from the command line

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.

Archived

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>
```

Archived

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.

Archived

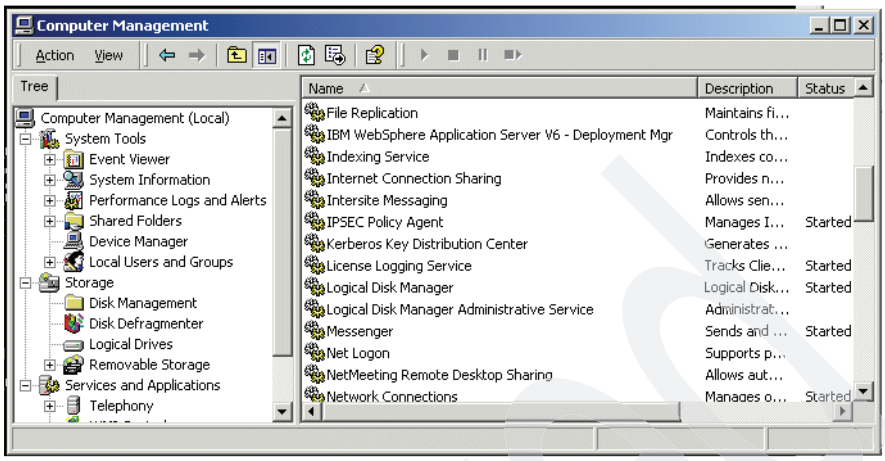


Figure 4-37 New service

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\CUSTOMO1 -restart true

Adding Service: CustomNode
     Config Root: C:\WebSphere\AppServer\profiles\CUSTOMO1 -restart true\config
     Server Name: nodeagent
     Profile Path: C:\WebSphere\AppServer\profiles\CUSTOMO1 -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
-proflePath "C:\WebSphere\AppServer\profiles\CUSTOMO1 -restart true
```

Archived

```
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.

Archived

5

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

© Copyright IBM Corp. 2005. All rights reserved.

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.

Archived

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.

Archived

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.

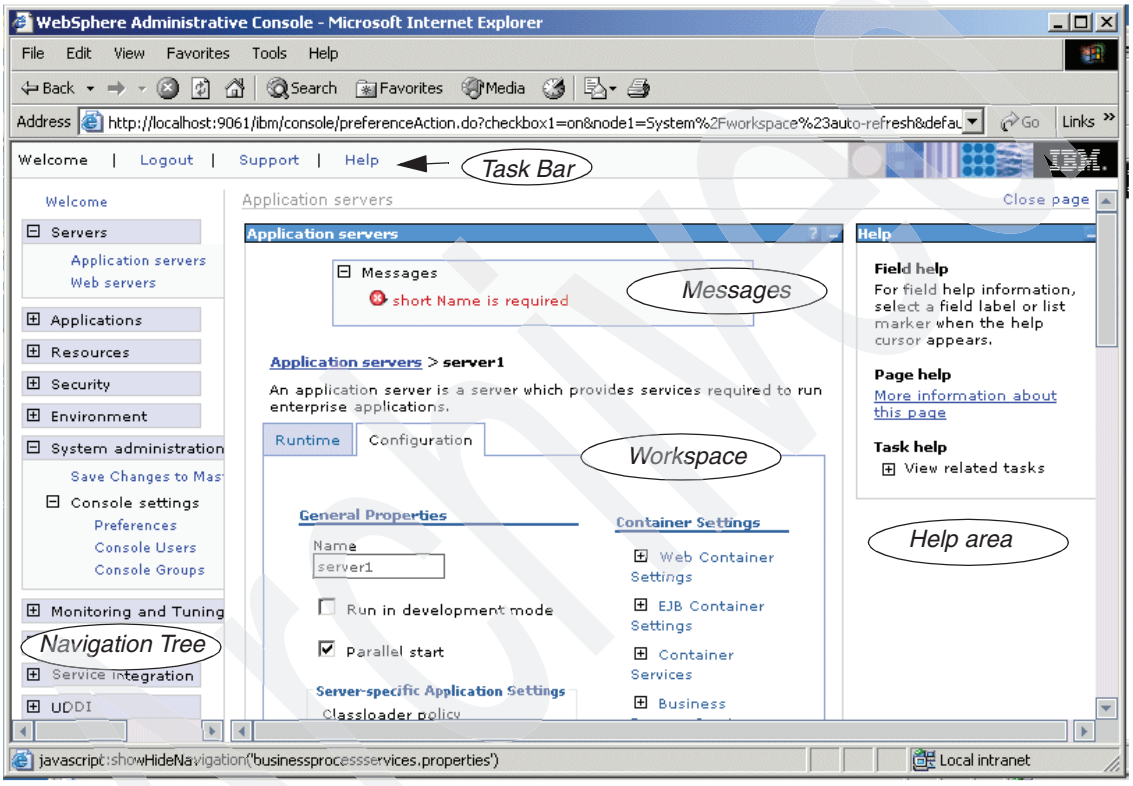


Figure 5-1 The administrative console graphical interface

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

Archive

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.

Archived

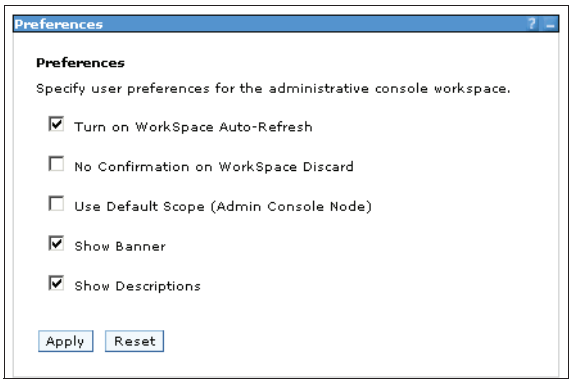


Figure 5-2 Administrative console preferences

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.

Archived

3. Set the preferences to specify how you would like information to be displayed on the page.

Select task

The navigation tree on the left side of the console contains links to console pages that you use to create and manage components in a WebSphere administrative cell. To create a JDBC provider, for example, expand **Resources** and then select the **JDBC Providers** action. See Figure 5-3.

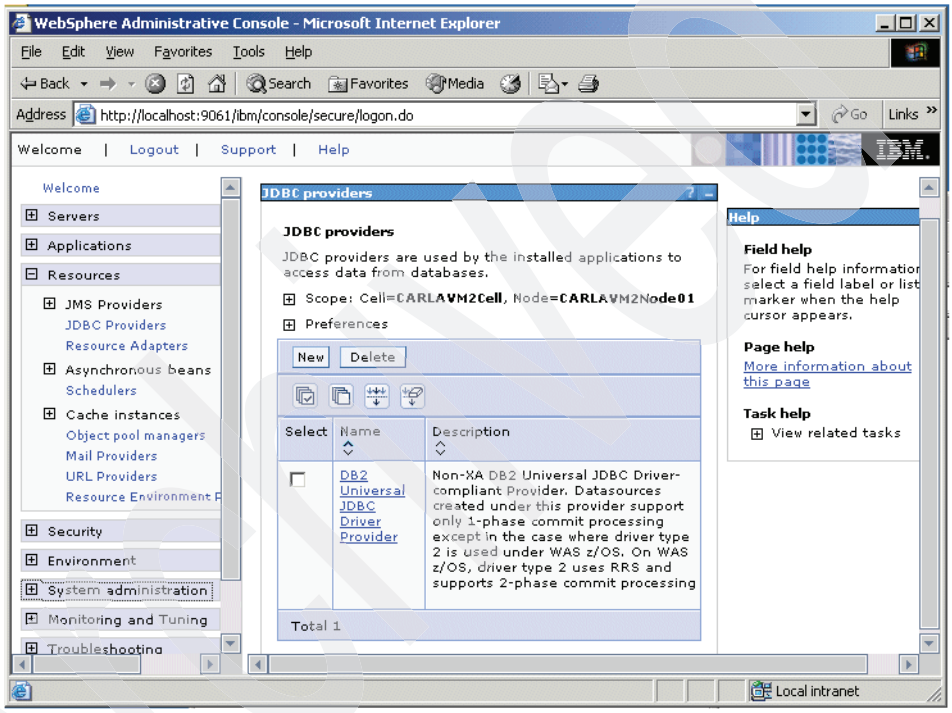


Figure 5-3 Working with the administrative console

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.

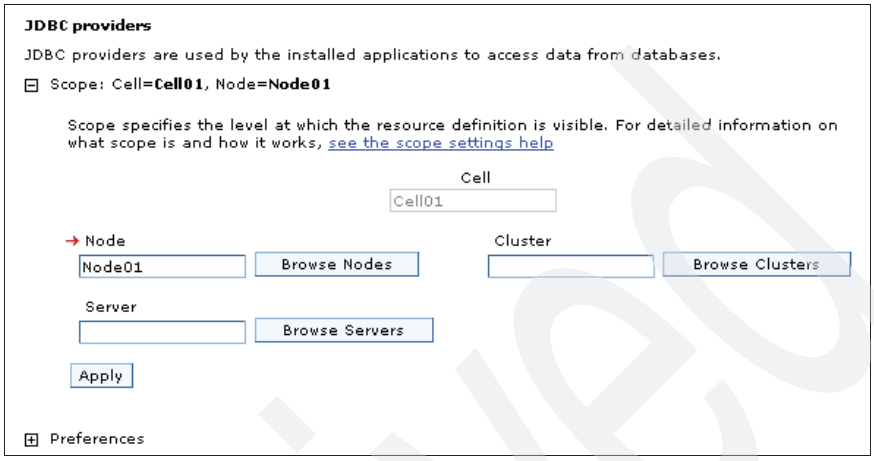


Figure 5-4 Setting scope

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.

Archived

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.

Archived

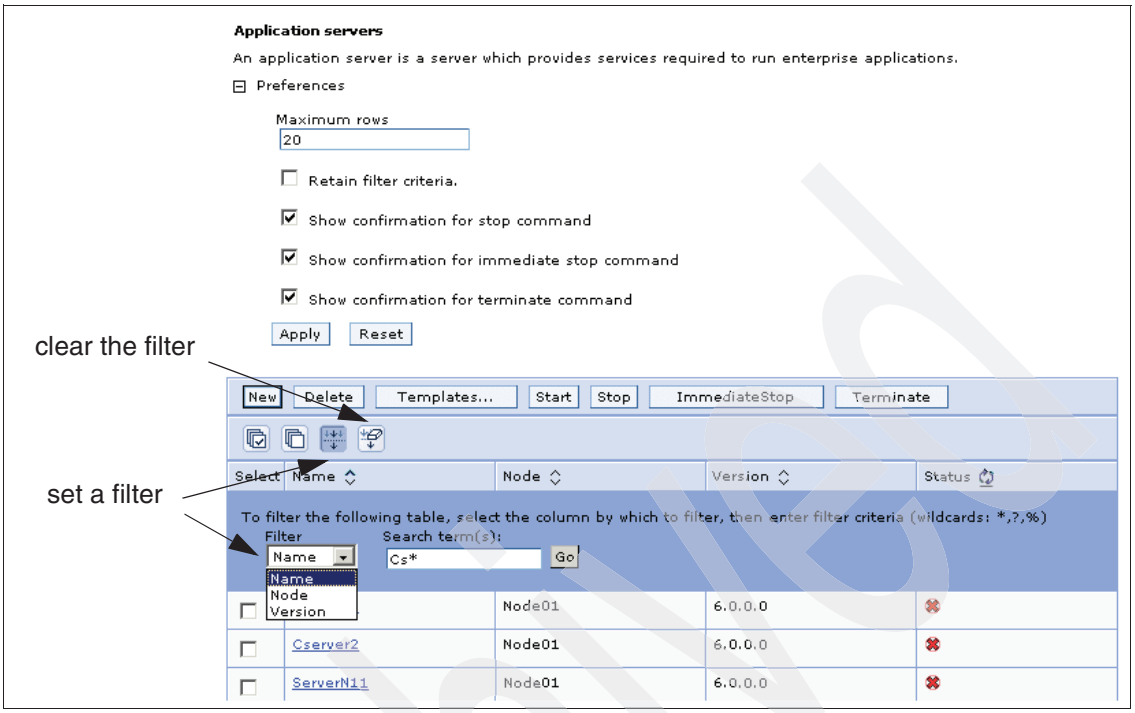


Figure 5-5 Setting filters and preferences

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.

Archived

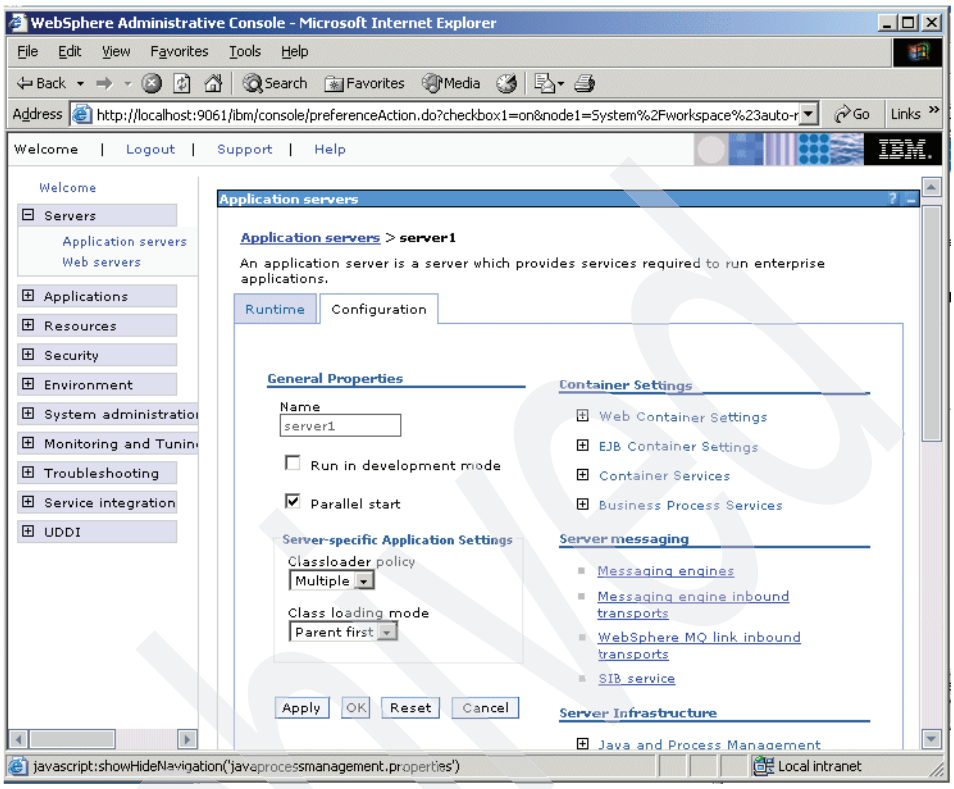


Figure 5-6 Editing application server properties

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.

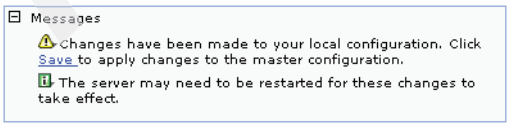


Figure 5-7 Save changes to the master repository

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.

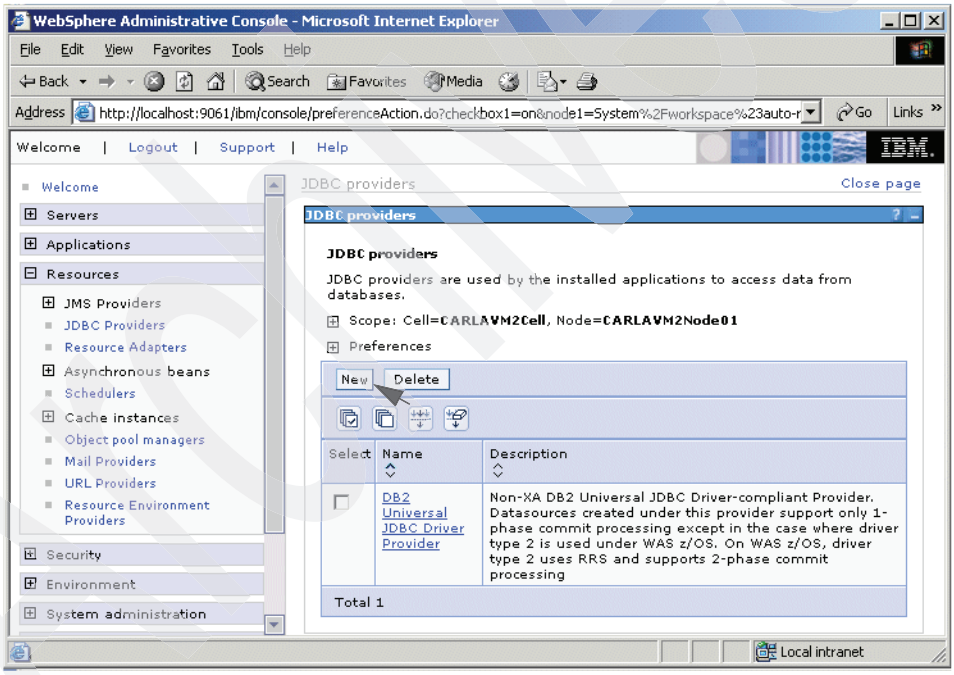


Figure 5-8 Create a new item

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**.

Archived

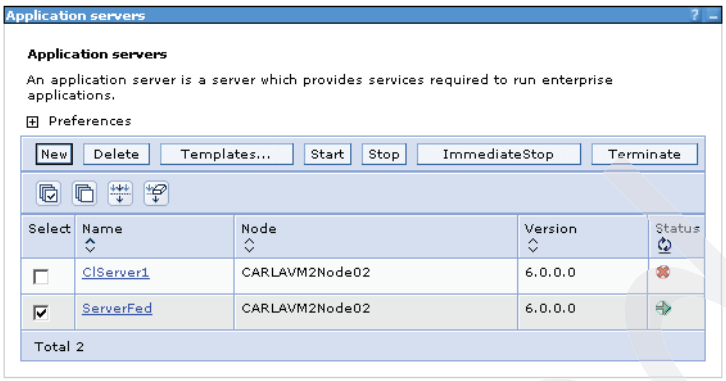


Figure 5-9 Starting and stopping items

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**

Archive

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.

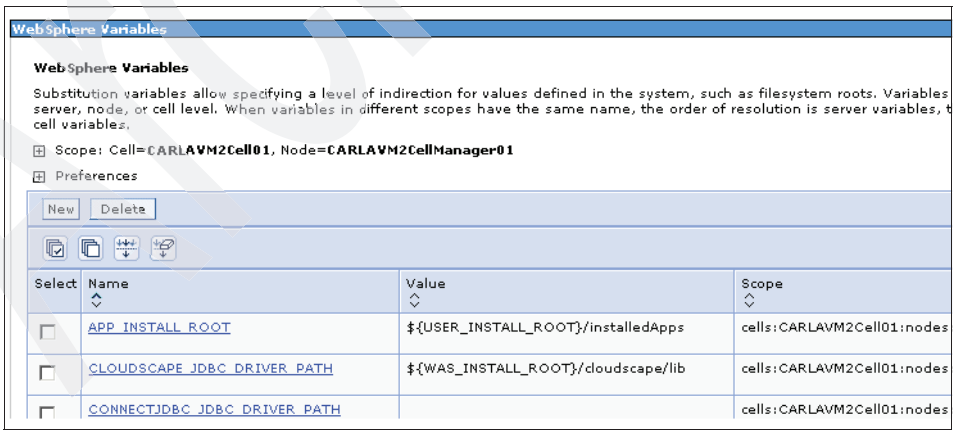


Figure 5-10 WebSphere variables

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.

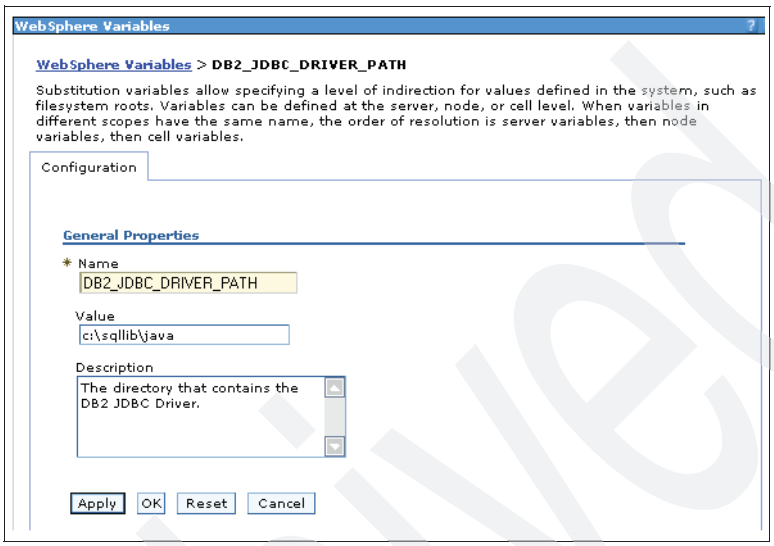


Figure 5-11 New WebSphere variable

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.

Archive

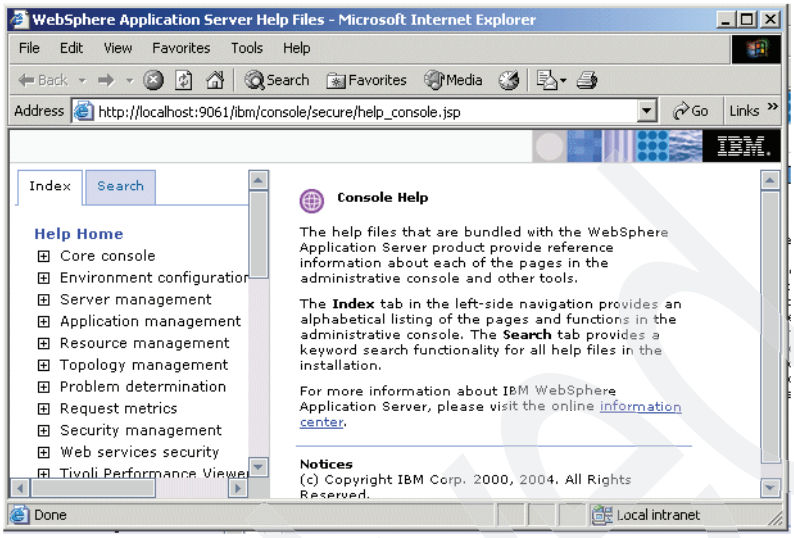


Figure 5-12 Online help

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.

Archived

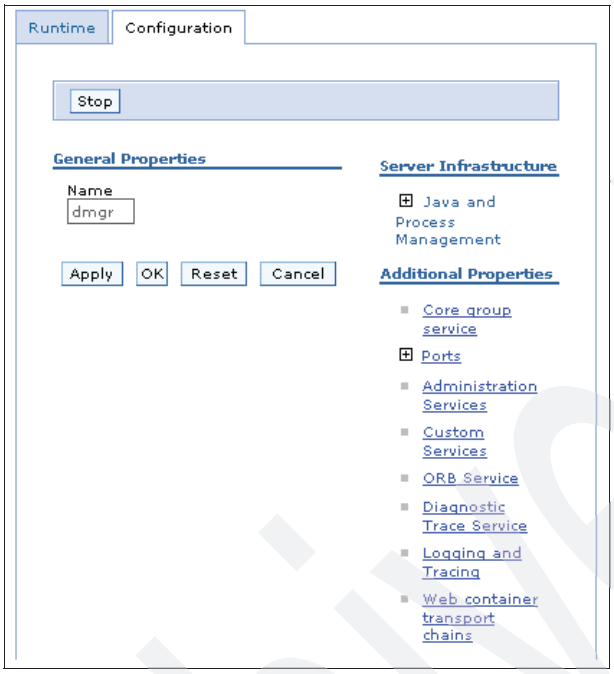


Figure 5-13 Deployment manager configuration

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

Archived

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

Archived

- 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.

Archived

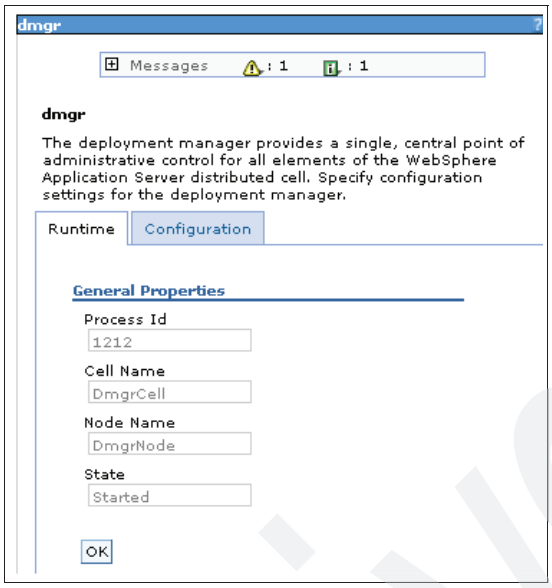


Figure 5-14 Deployment manager runtime page

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 manage**r

Archived

- **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 *<profile_home>*/logs/dmgr/startServer.log

Archived

Option	Description
-profileName <profile>	Specifiy 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>	Specifiy 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.

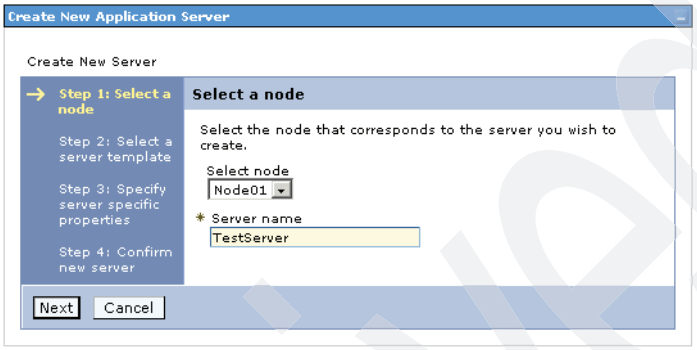


Figure 5-15 Create an application server: Step 1

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).

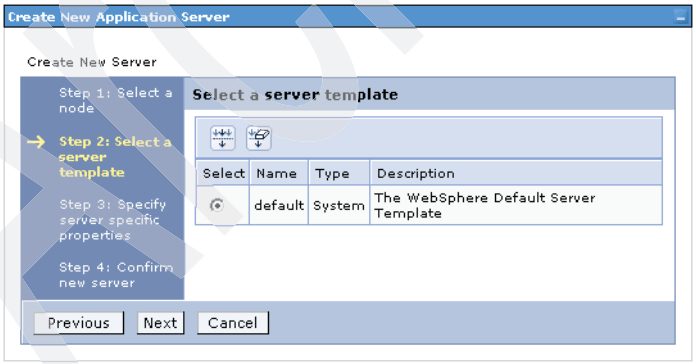


Figure 5-16 Create an application server: Step 2

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.

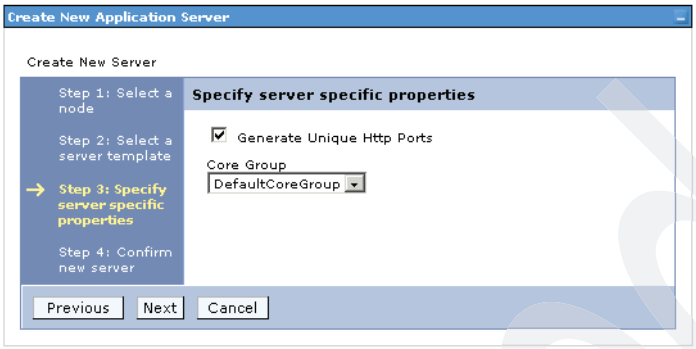


Figure 5-17 Create an application server: Step 3

Click **Next**.

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

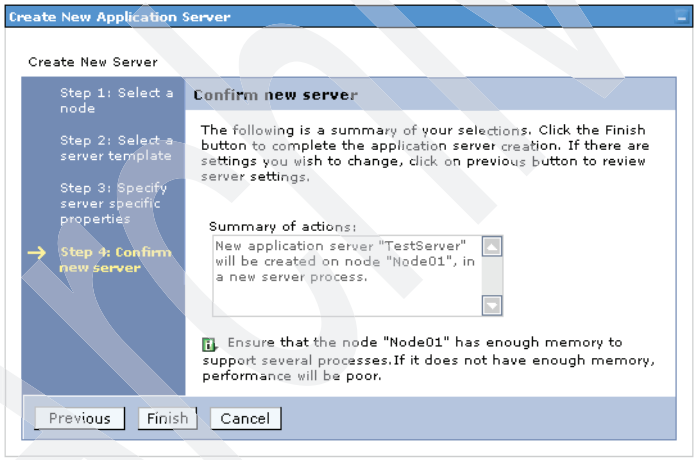


Figure 5-18 Create an application server: Step 4

Click **Finish** to create the server.

8. In the messages box, click **save** to save the changes to the master repository.

Archived

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	To view the status of a standalone application server, type:
		
		`cd <profile_home>/bin`
		`serverStatus(.sh) server1`
		
		To view the status of a managed application server, type:
		
		`cd <profile_home>/bin`
		`serverStatus(.sh) <server_name>`
		
		To check the status of all servers on the node, type:
		
		`cd <profile_home>/bin`
		`serverStatus(.sh) -all`

Administrative console	Managed	Select **Servers** →**Application Servers**

Archived

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.

Archived

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>`	Us 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.

Archive

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>]	Generate a launch script instead of starting the server. The script file name is optional. If the file name is not provided, the default script file name is start_<server>. The script needs to be saved to the bin directory of the node installation.
-username <username>	User name for authentication if WebSphere security is enabled. Ignored if WebSphere security is disabled.
-password <password>	Specify a password for authentication if WebSphere security is enabled. The password is ignored if WebSphere security is disabled.
-timeout <seconds>	Specify the waiting time before server initialization times out and returns an error.
-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.

Archive

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**

Archived

Method	Server types:	Summary

Command line	Managed and standalone	For a standalone application server:
		
		`cd <profile_home>/bin`
		`stopServer(.sh) server1`
		
		For a managed application server :
		
		`cd <profile_home>/bin`
		`stopServer(.sh) <server_name>`

Administrative console	Managed	**Servers →Application Servers**. To stop a managed server from the administrative console, the node agent must be started.
Administrative console	Managed	**System Administration →Node Agents → Restart all Servers on the Node**. This restarts all the servers on the node.

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`

Archived

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.

Archived

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.

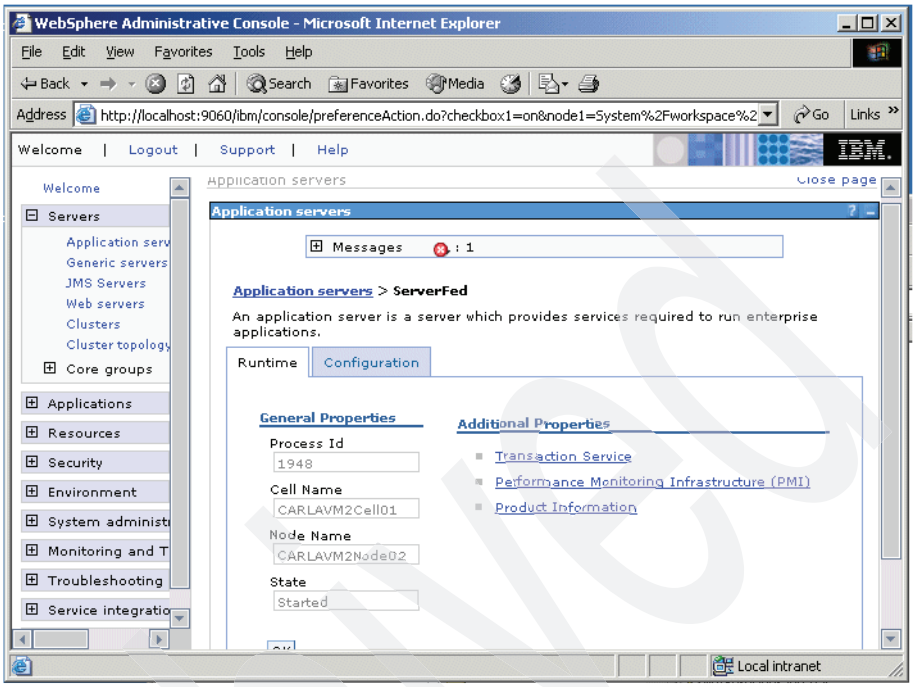


Figure 5-20 Application server Runtime tab

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

Archive

– **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.

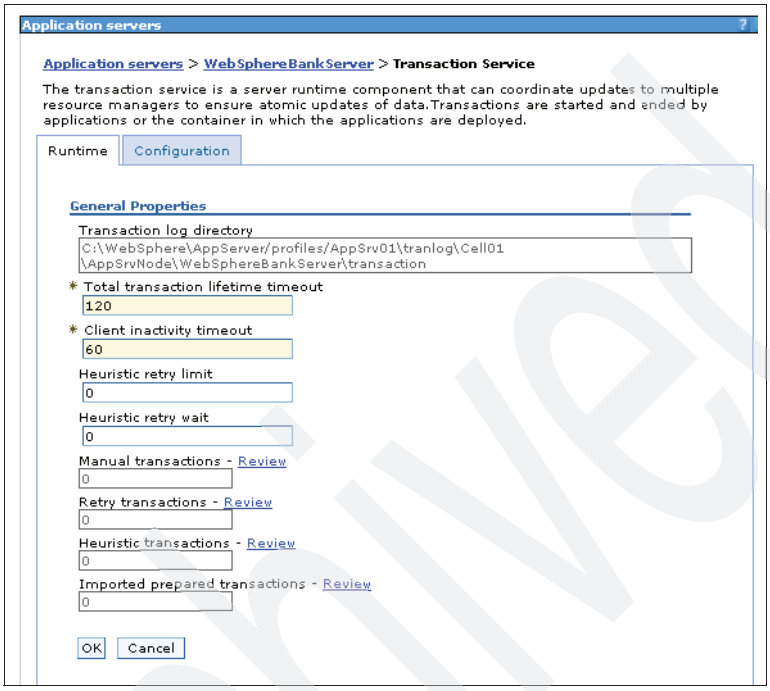


Figure 5-21 Transaction service options and settings

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 retrying, transactions in this state.

Archived

- 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. SeeFigure 5-22 on page 207.

Archived

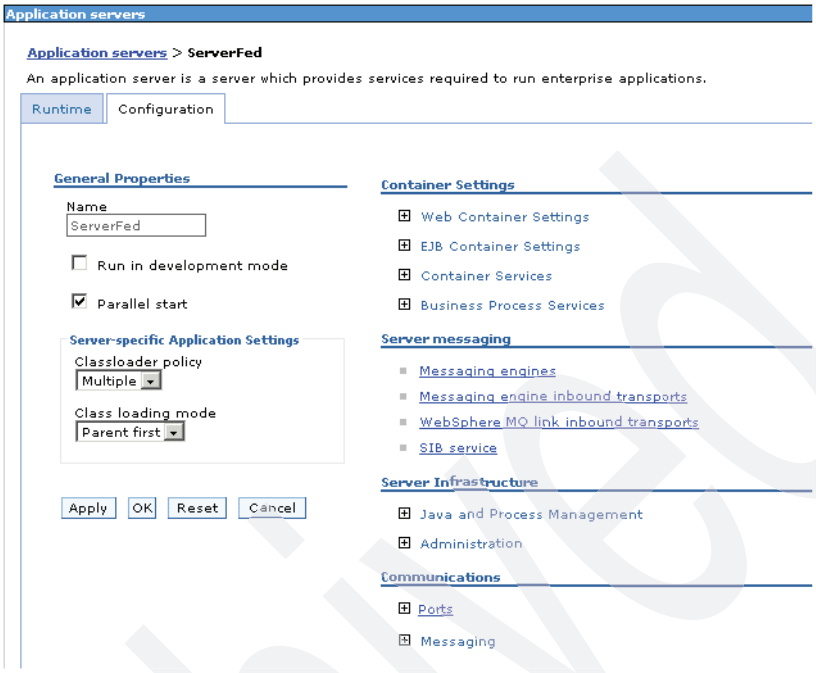


Figure 5-22 Application server configuration

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

Archived

caching, specify session manager settings such as persistence and tuning parameters, and HTTP transport properties. See Figure 5-23.

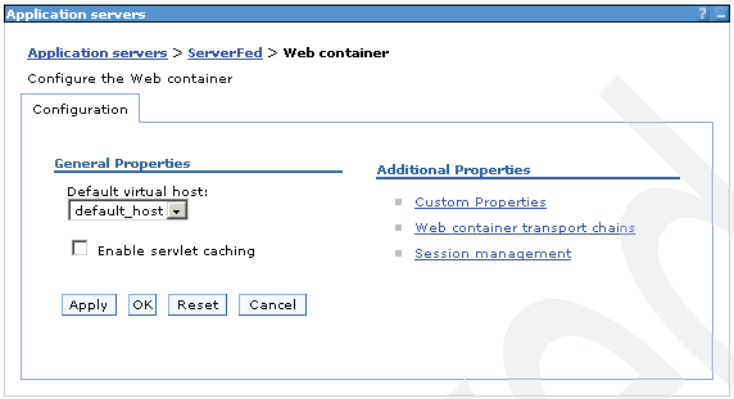


Figure 5-23 Web container settings

- **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.

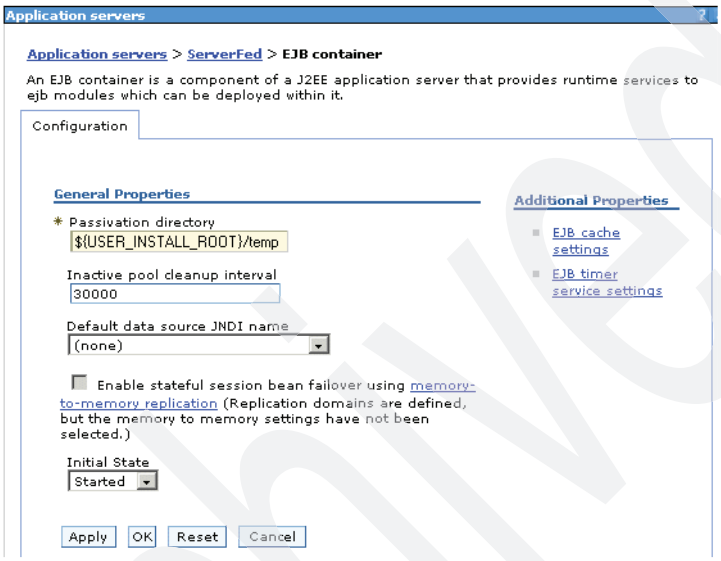


Figure 5-24 EJB container settings

- **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

Archived

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.

Archived

- **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.

Archived

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.

Archived

- 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 ► Enabled ► Host = * ► Port = 9061 ► SSL disabled	TCP Inbound Channel (TCP 1) ► Host = * ► Port = 9061 ► Thread pool=Web container ► Max open connections = 100 ► Inactivity timeout = 60 sec
		HTTP Inbound Channel (HTTP 1) ► Keepalive enabled ► Max persistent requests = 100 ► Read timeout = 60 sec ► Write timeout = 60 sec ► Persistent timeout = 30 sec
		Web Container Inbound Channel (WCC 1) ► Discrimination weight = 1 ► Write buffer size = 32768

Archived

Port	Transport chain	Transport channels
WC_adminhost_secure	WCInboundAdminSecure ► Enabled ► Host = * ► Port = 9044 ► SSL enabled	TCP Inbound Channel (TCP 1) ► Host = * ► Port = 9044 ► Thread pool=Web container ► Max open connections = 100 ► Inactivity timeout = 60 sec
		SSL Inbound Channel (SSL 1) ► SSL repertoire DMGRNode/ DefaultSSLSettings
		HTTP Inbound Channel (HTTP 3) ► Keepalive enabled ► Max persistent requests = 100 ► Read timeout = 60 sec ► Write timeout = 60 sec ► Persistent timeout = 30 sec
		Web Container Inbound Channel (WCC 1) ► Discrimination weight = 1 ► Write buffer size = 32768
WC_defaulthost	WCInboundAdminSecure ► Enabled ► Host = * ► Port = 9080 ► SSL disabled	TCP Inbound Channel (TCP 2) ► Host = * ► Port = 9080 ► Thread pool=Web container ► Max open connections = 20000 ► Inactivity timeout = 60 sec
		HTTP Inbound Channel (HTTP 2) ► Keepalive enabled ► Max persistent requests = 100 ► Read timeout = 60 sec ► Write timeout = 60 sec ► Persistent timeout = 30 sec
		Web Container Inbound Channel (WCC 2) ► Discrimination weight = 1 ► Write buffer size = 32768

Archive

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:

Archived

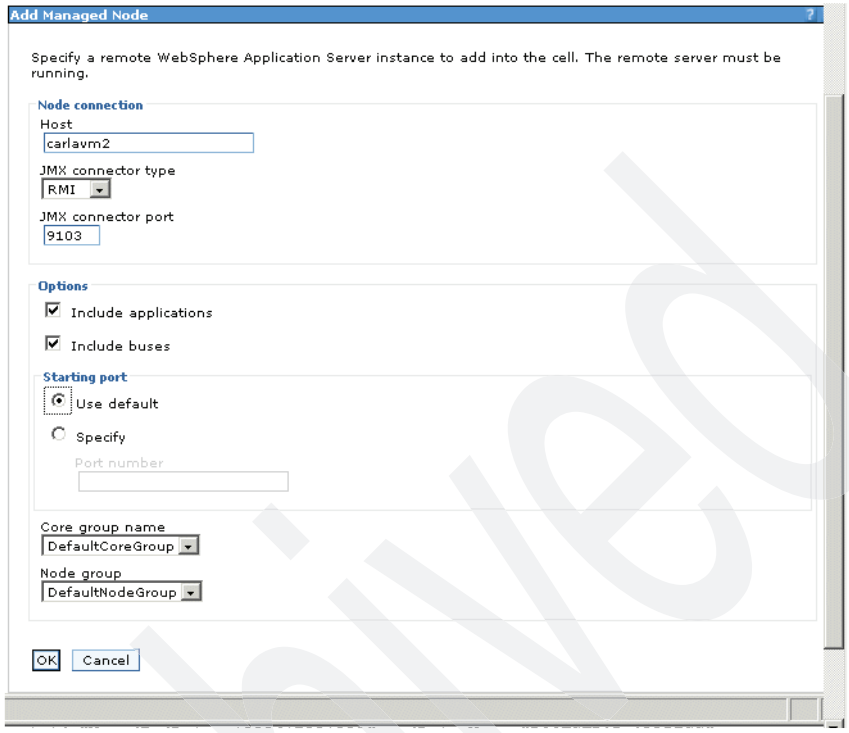


Figure 5-25 Working with nodes

- 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.

Archived

– 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:
```

Archived

```
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
`-nowait`	Tell the command not to wait for successful completion of the node addition.
`-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 addNode.log.
`-logfile <log file path>`	Specify an alternative location for command's log output, instead of addNode.log. The path can be specified in the following forms: absolute, relative, or file name. The default is *<profile_home>*/logs/addNode.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.
`-conntype <type>`	Specify the JMX connector to use for connection. Valid values are SOAP or RMI. If not specified, SOAP is assumed. If RMI is specified, then the deployment manager's correct RMI/IIOP JMX connector port must be specified by the <dmgr_port> argument.
`-profileName <profile>`	Specify the profile to run the command against. If the command is run from *<was_home>*/bin and -profileName is not specified, the default profile is used. If it is run from *<profile_home>*/bin, that profile is used.
`-username <username>`	Specify a user name for authentication if WebSphere security is enabled. The user name is ignored if WebSphere security is disabled.
`-password <password>`	Specify a password for authentication if WebSphere security is enabled. The password is ignored if WebSphere security is disabled.
`-includeapps`	Attempt to include the applications in the incorporation of the base node into a cell. Default is not to include the applications.

Archived

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/CARLAVM2Node02Cel
l/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.

Archived

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.

Archived

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.

Archived

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

Archived

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.

Archived

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.

Archived

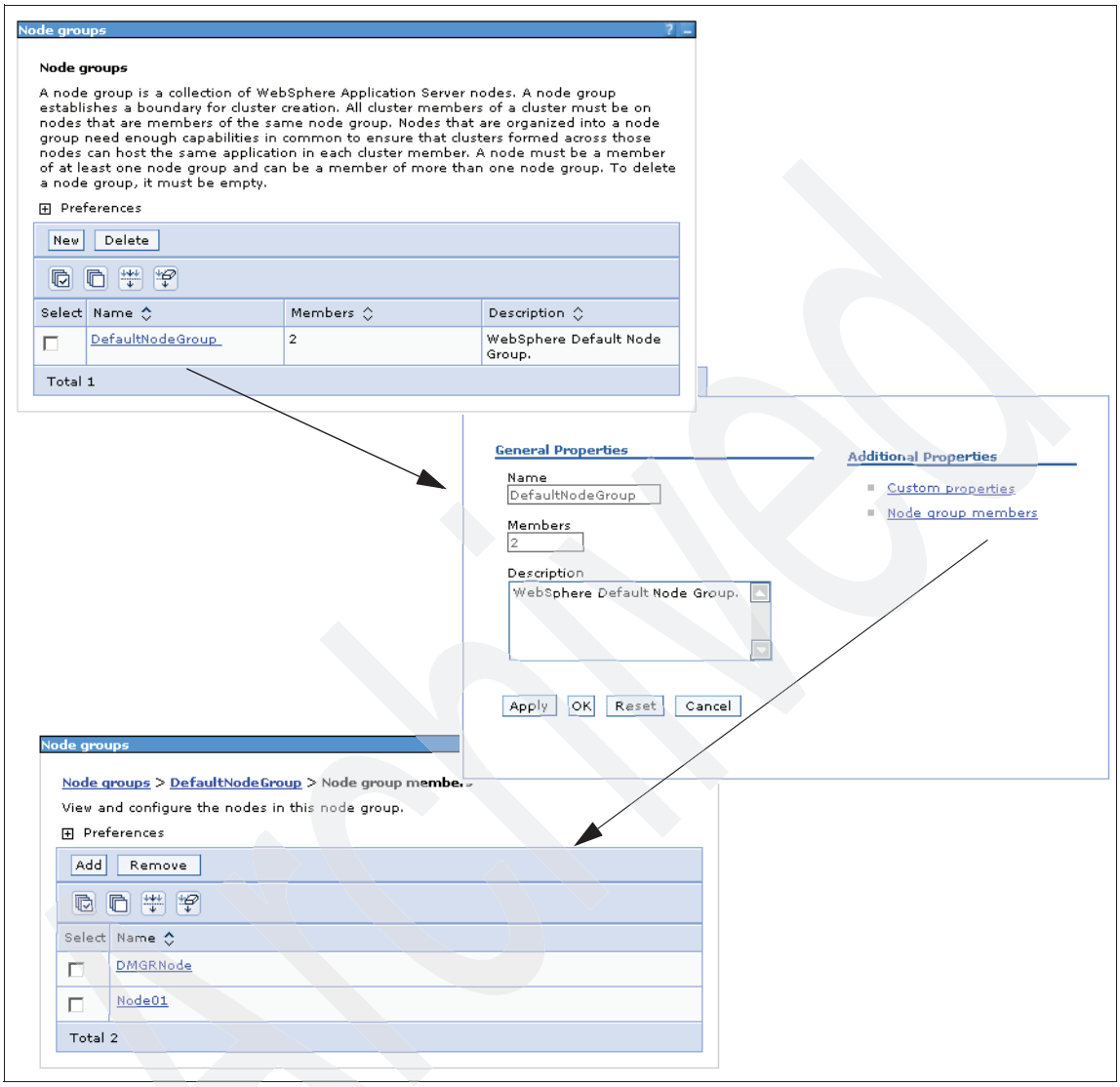


Figure 5-26 Displaying node groups

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.

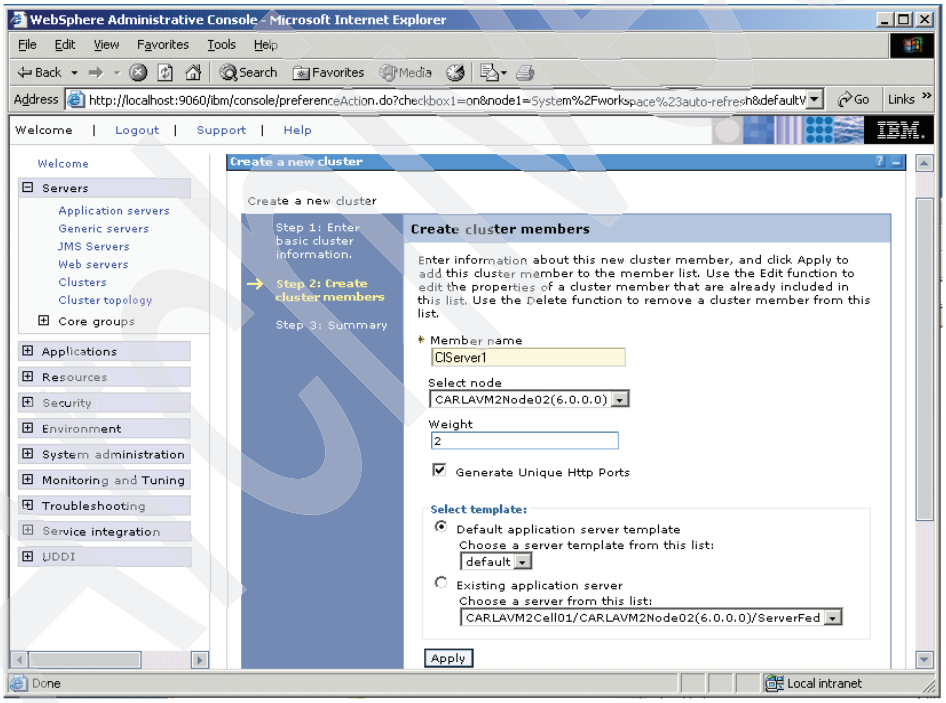


Figure 5-27 Creating a new cluster

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.

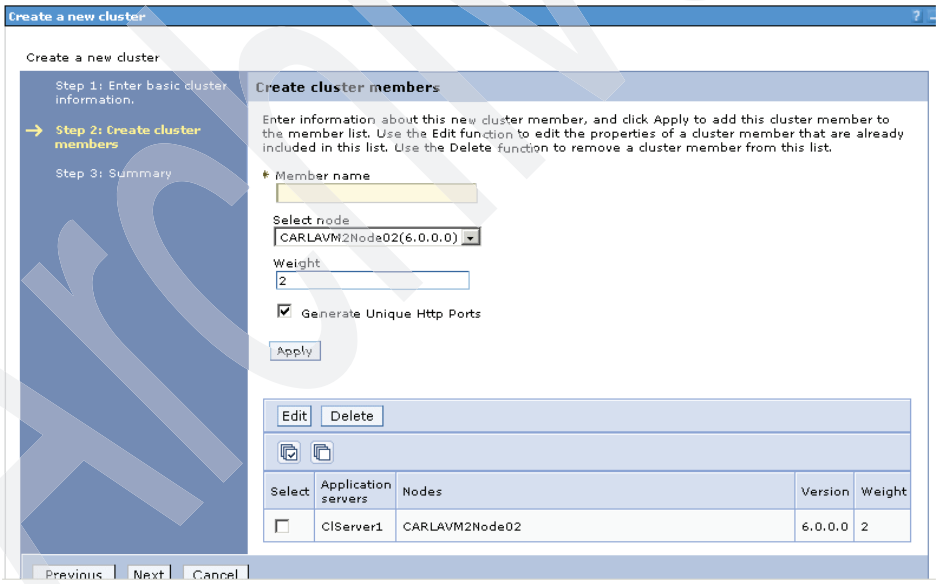


Figure 5-28 Add servers to the new cluster

– **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.

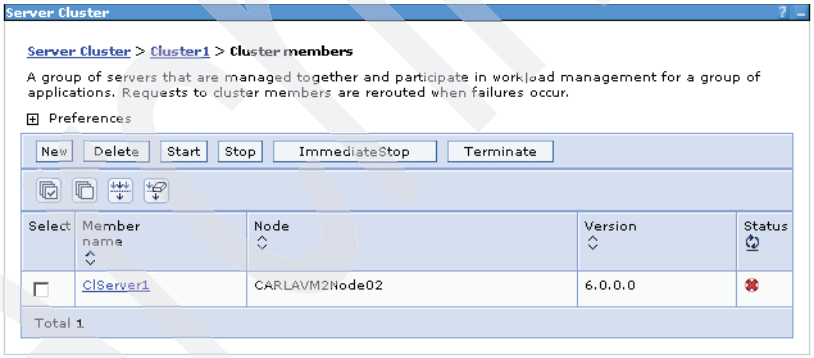


Figure 5-29 Cluster topology view

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

Archived

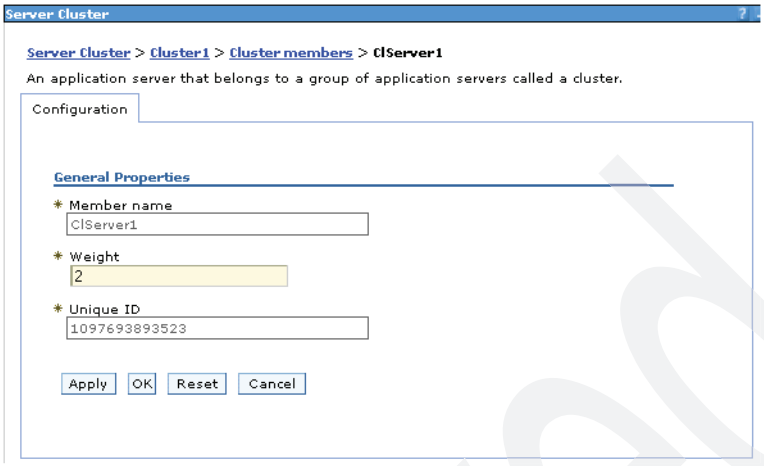


Figure 5-30 Viewing the cluster characteristics of a server

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

Archived

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.

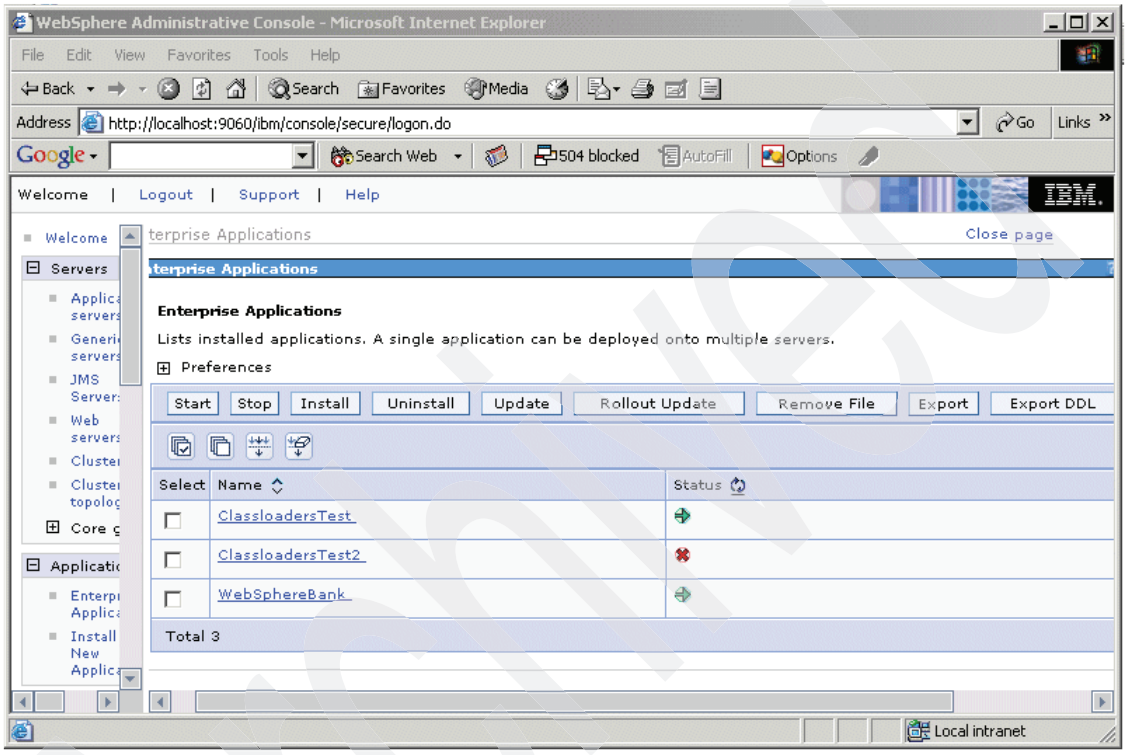


Figure 5-31 Working with enterprise applications

- 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**.

Archived

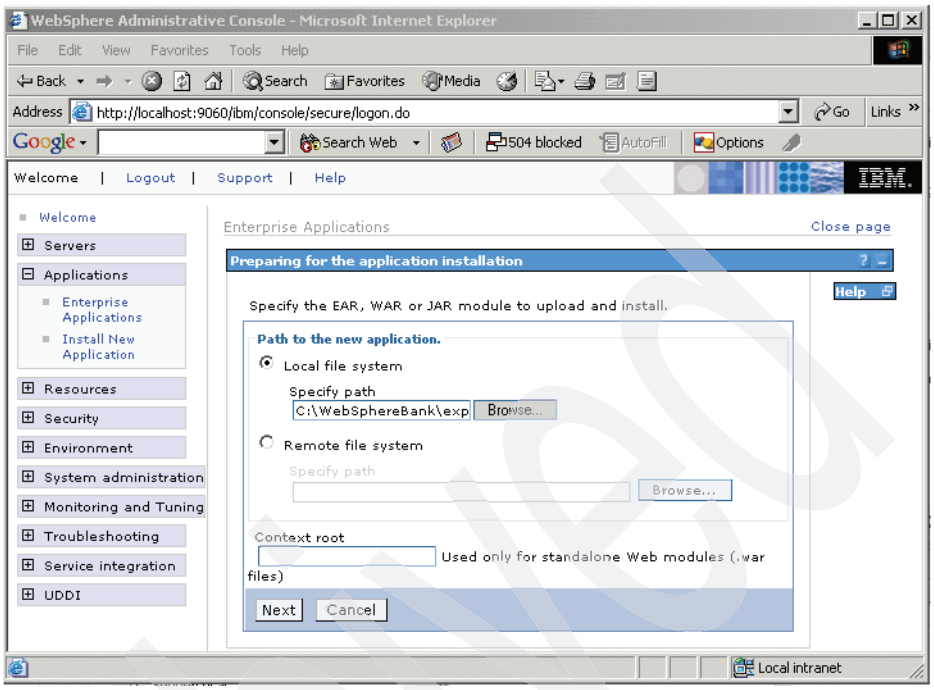


Figure 5-32 Installing an enterprise application

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 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**.

Archived

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

Archived

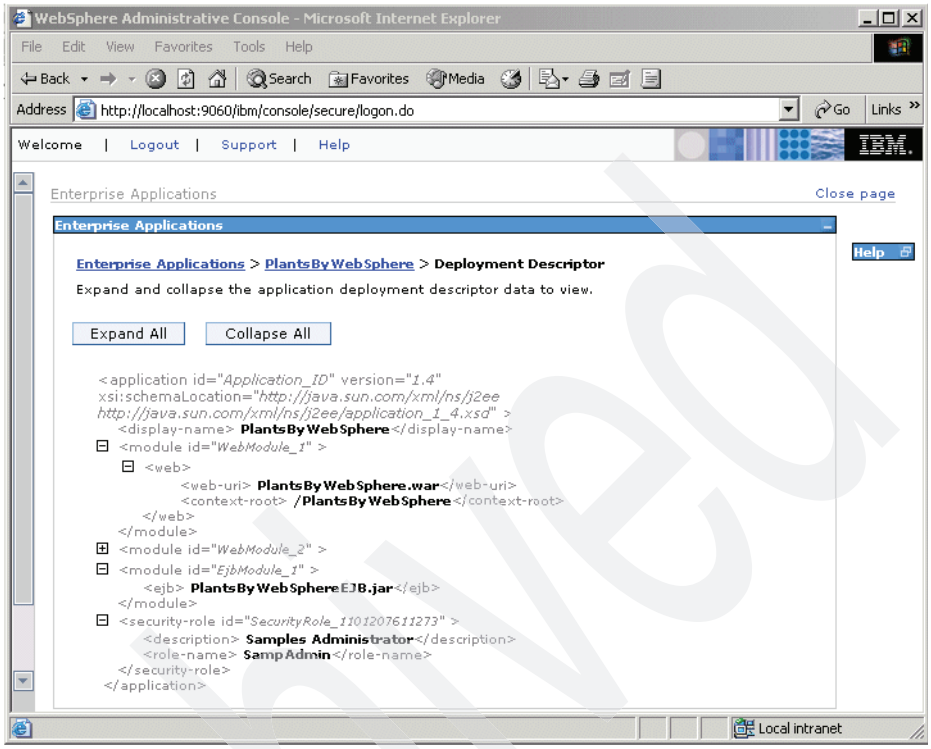


Figure 5-33 Enterprise application deployment descriptor

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.

Archived

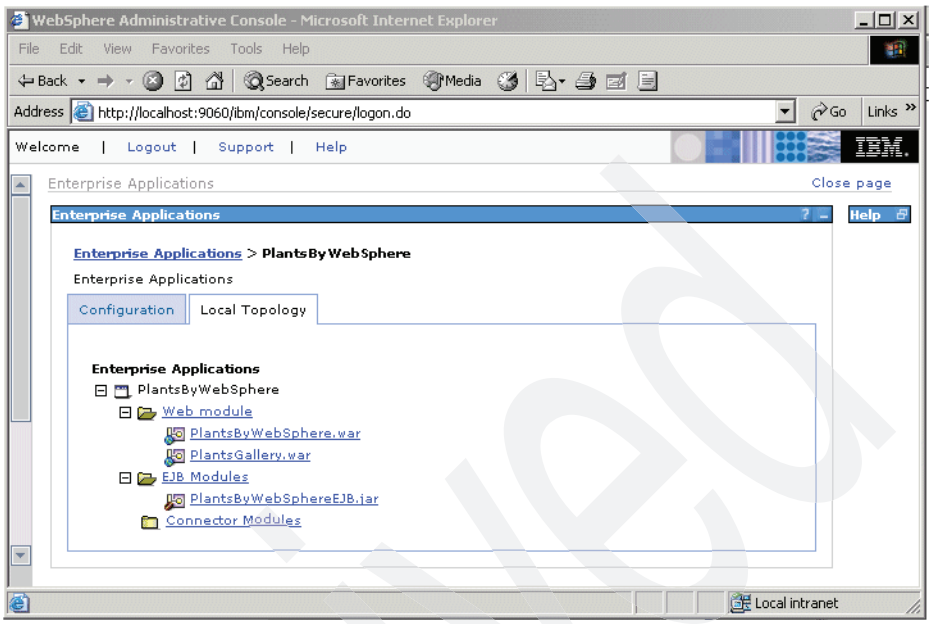


Figure 5-34 Topology view of the application

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.

Archived

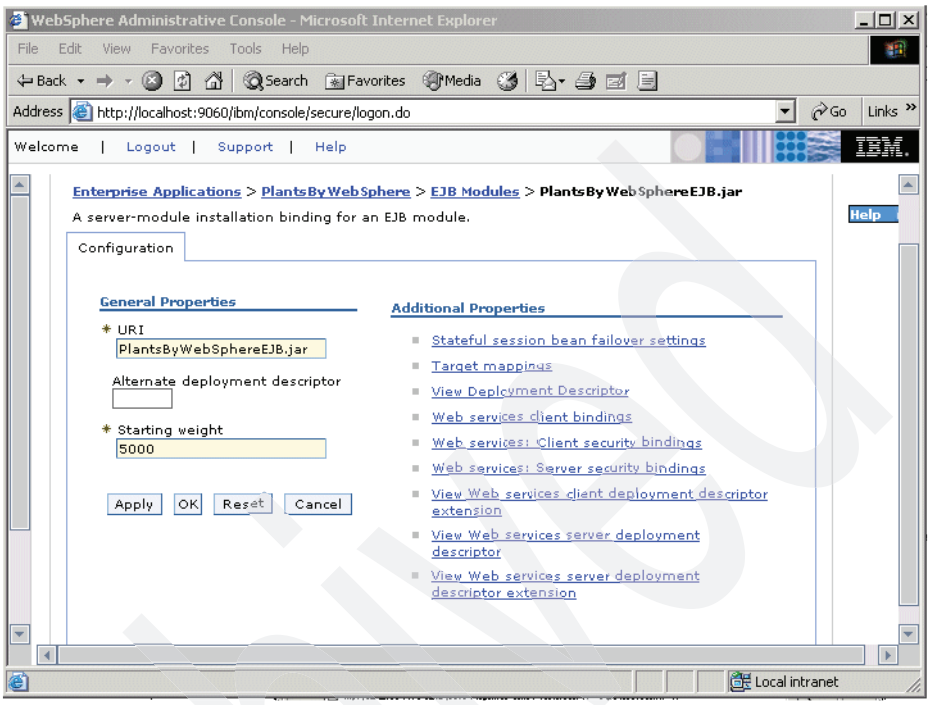


Figure 5-35 Viewing an EJB module configuration

5. Click **View Deployment Descriptor** under Additional Properties to see the EJB deployment descriptor.

Archived

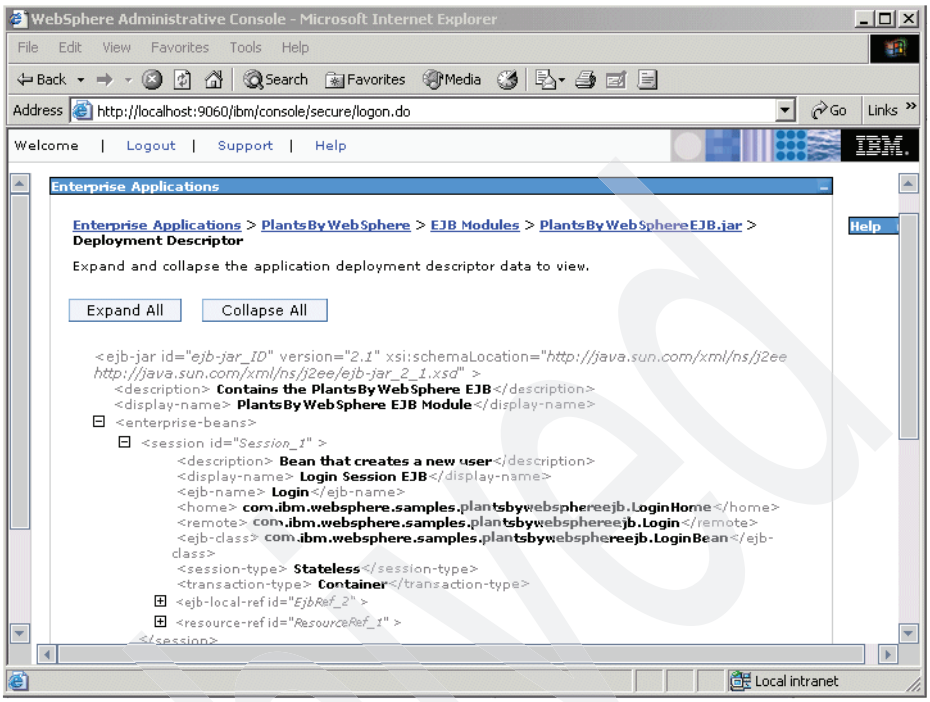


Figure 5-36 EJB module deployment descriptor

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**.

Archived

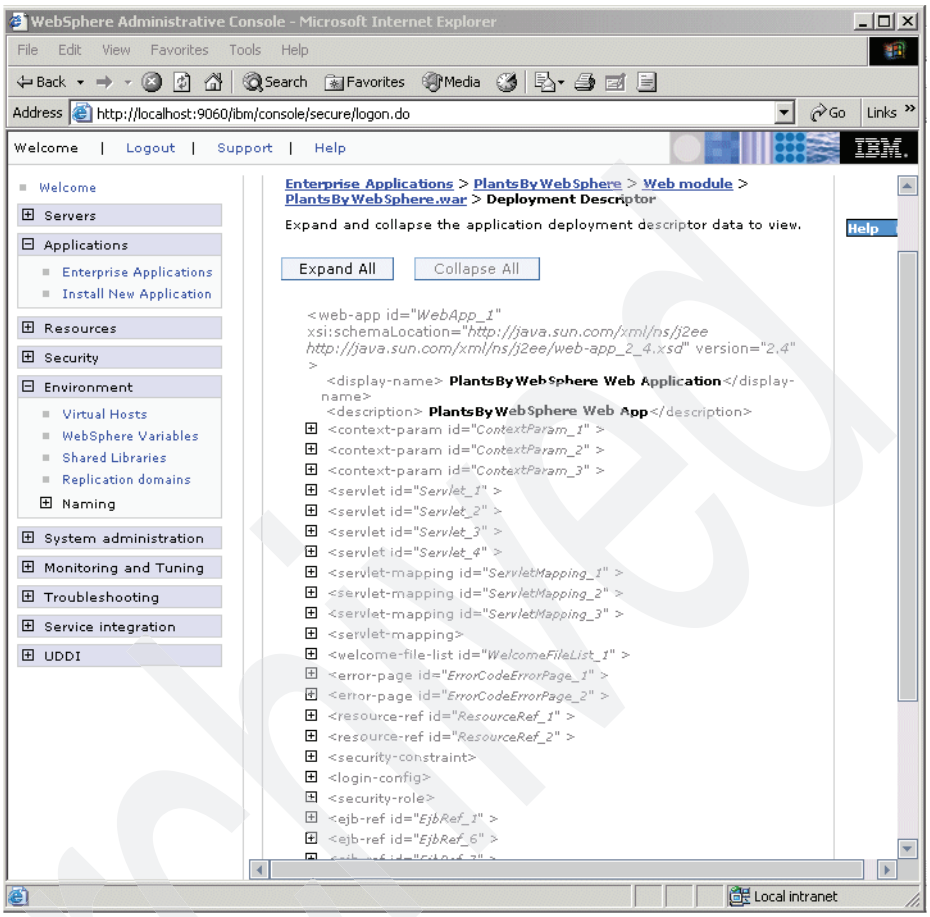


Figure 5-37 Web module deployment descriptor

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.

Anonymous

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.

Archived

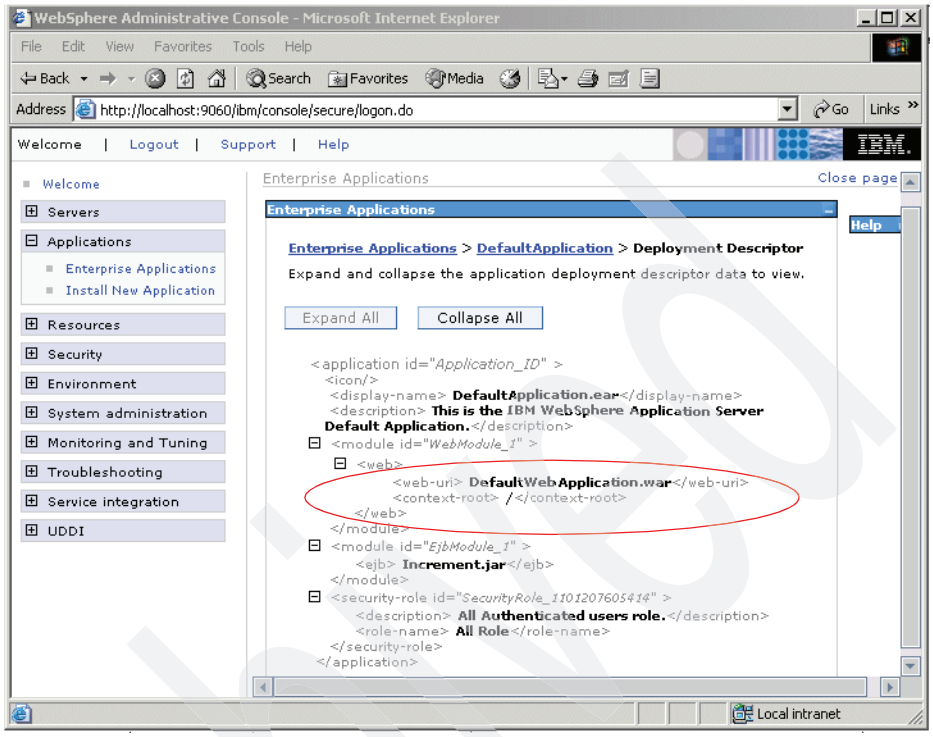


Figure 5-38 Deployment descriptor for DefaultApplication

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 “/”.

Arihwo

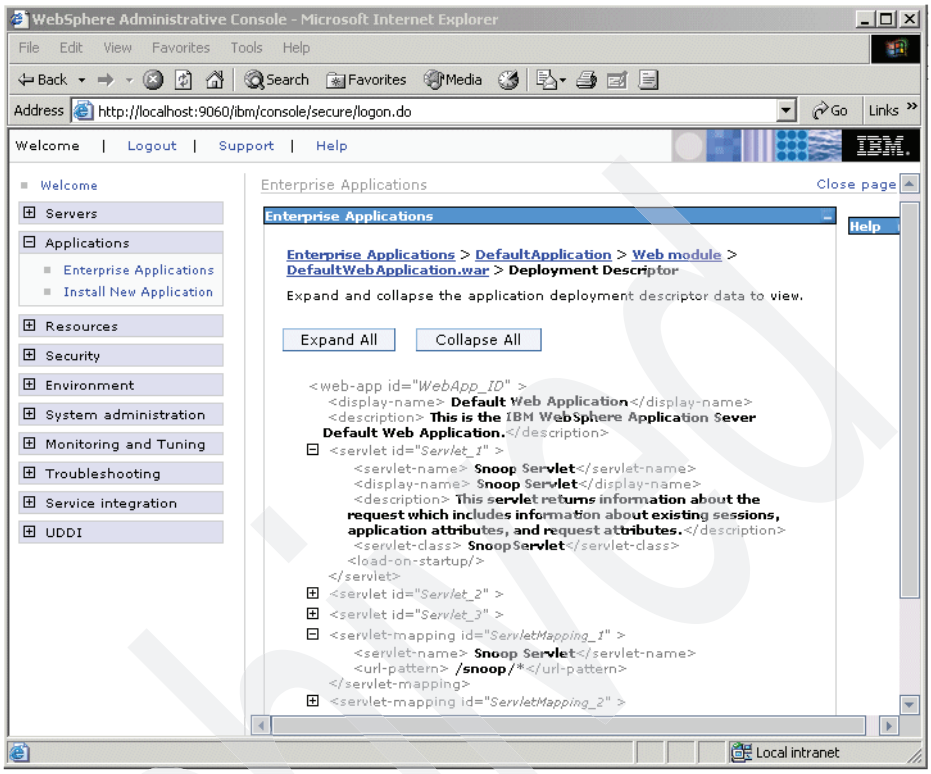


Figure 5-39 DefaultWebApplication Web module deployment descriptor

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.

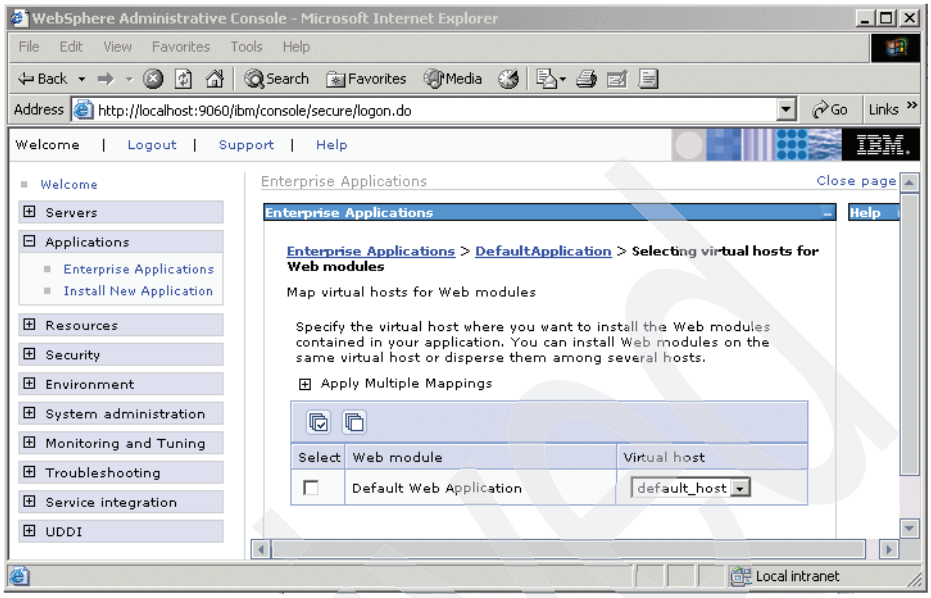


Figure 5-40 List of virtual hosts

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.

Archive

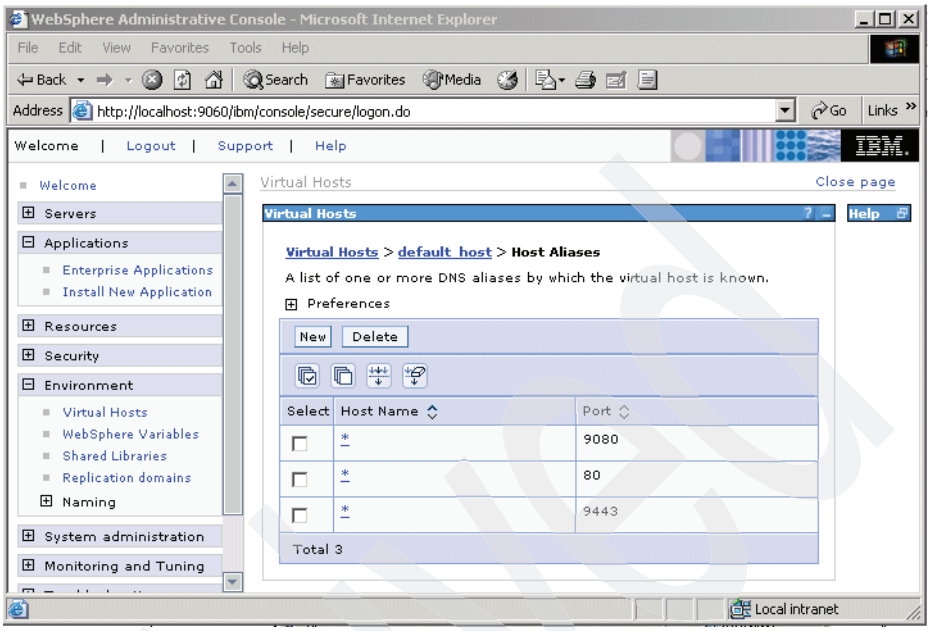


Figure 5-41 Default_host virtual host aliases

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
-nostop	Servers are not to be stopped before backing up the configuration.
-quiet	Suppresses the printing of progress information.
-logfile <fileName>	Name of the log file to which information gets written. 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.

Archived

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
```

```
           already exists; renaming to
           C:\WebSphere\AppServer\profiles\AppSrv01\config.old
ADMU5504I: Restore location successfully renamed
ADMU5505I: Restoring file d:\wasbackups\appsrv01Nov022004 to location
           C:\WebSphere\AppServer\profiles\AppSrv01\config
.....................................................................
ADMU5506I: 187 files successfully restored
ADMU6001I: Begin App Preparation -
ADMU6009I: Processing complete.
```

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

Archived

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 it's pre-federated configuration that was saved as part of the federation process.
  - Delete the profile using **wasprofile -delete**.
  - Delete the *<profile_home>* directory.

Archive

  - 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 it's 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:
```

Archived

```
nodeUninstConfig:
BUILD SUCCESSFUL
Total time: 10 seconds

detectCurrentOSFamily:
setOSFileSeparator:
defineOSSpecificConfigFlag:
processJScriptForDeletion:
     [copy] Copying 1 file to C:\WebSphere\AppServer\temp
replaceAllInstancesOfGivenTokenWithGivenValueForTheGivenFile:
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.

Archived

Archived

6

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.

© Copyright IBM Corp. 2005. All rights reserved.

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.

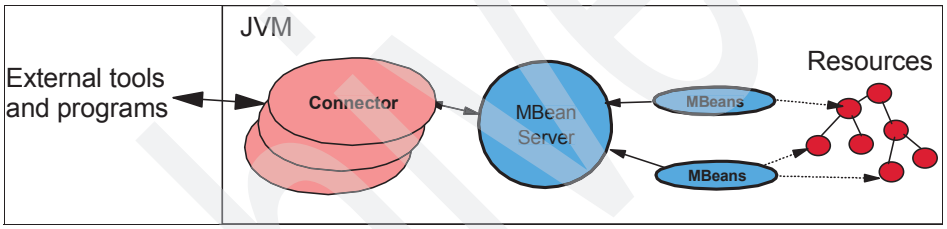


Figure 6-1 wsadmin scripting

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 ]
```

Archived

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 take 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
```

Archived

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 he 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

Archived

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
```

Archived

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.

Us 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 is 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/P
latoCellManager/servers/dmgr/server.xml#Server_1,cell=PlatoCell,processType=Dep
loymentManager
```

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                RO
shortName                       java.lang.String                RO
threadMonitorInterval           int                             RW
threadMonitorThreshold          int                             RW
threadMonitorAdjustmentThreshold int                            RW
pid                             java.lang.String                RO
cellName                        java.lang.String                RO
cellShortName                   java.lang.String                RO
```

```
deployedObjects                 [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 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.

Archive

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
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 Architec ture (J2C)-related resources.
LocalModeGroup - No description available
ManagedObjectMetadata - Managed Object Metetadata 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
```

Archived

```
    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

Us 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.

Archived

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 [$AdminControl queryNames
type=ApplicationManager,node=PericlesNode,process=server1,*]
WebSphere:platform=dynamicproxy,cell=PericlesCell,version=6.0.0.0,name=Applicat
ionManager,mbeanIdentifier=ApplicationManager,type=ApplicationManager,node=Peri
clesNode,process=server1
wsadmin>$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.

Archived

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 `tstClst` 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 tstClst [$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 $tstClst 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 `tstClst` 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 tstClst [$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 $tstClst 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,mbea
nIdentifier=cells/PlatoCell/nodes/PlatoCellManager/servers/dmgr/server.xml#Trac
eService_1,type=TraceService,node=PlatoCellManager,process=dmgr

WebSphere:platform=common,cell=PlatoCell,version=6.0.0.0,name=TraceService,mbea
nIdentifier=cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1/server.x
ml#TraceService_1097069277091,type=TraceService,node=SocratesNode,process=Socra
tesServer1

WebSphere:platform=common,cell=PlatoCell,version=6.0.0.0,name=TraceService,mbea
nIdentifier=cells/PlatoCell/nodes/SocratesNode/servers/nodeagent/server.xml#Tra
ceService_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,mbea
nIdentifier=cells/PlatoCell/nodes/SocratesNode/servers/SocratesServer1/server.x
ml#TraceService_1097069277091,type=TraceService,node=SocratesNode,process=Socra
tesServer1

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

The 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

Archived

- 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 }
```

Archive

```
socServer3(cells/PlatoCell/nodes/SocratesNode/servers/socServer3|server.xml#Ser
ver_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#Ser
ver_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)
```

Archived

```
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>
  ```

Archived

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|
```

Archived

```
   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
```

Archived

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)
```

Archived

```
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)
```

Archived

```
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.

Archived

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.

Archived

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#bui
ltin_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/

Archived

Archived

7

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

© Copyright IBM Corp. 2005. All rights reserved.

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.

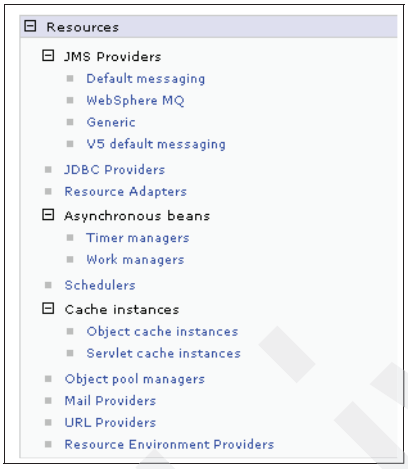


Figure 7-1 WebSphere Application Server resource types

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:

Archived

- *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

Archived

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.

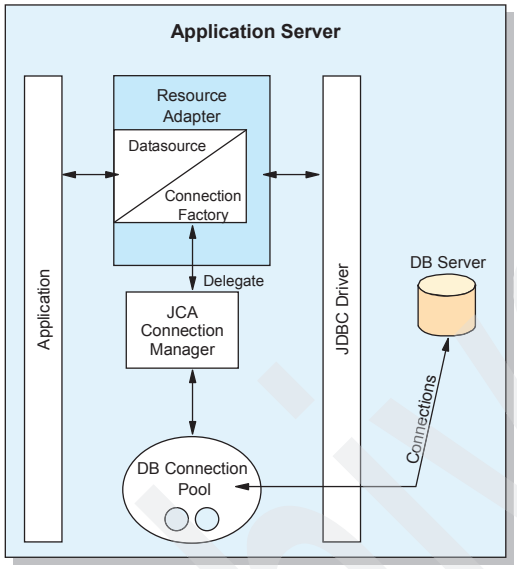


Figure 7-2 Resource adapter in J2EE connector architecture

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.

Archived

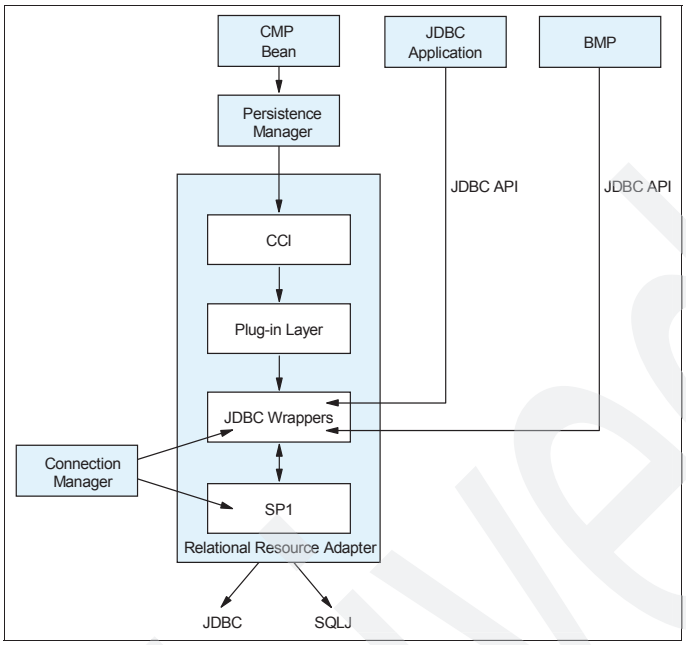


Figure 7-3 Persistence resource adapter model

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.

Archived

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.

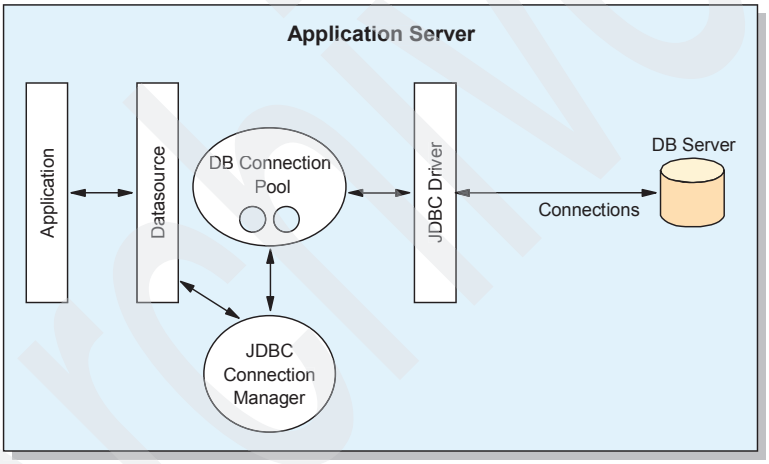


Figure 7-4 Connection pooling in WebSphere Application Server Version 4

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.

Archived

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.

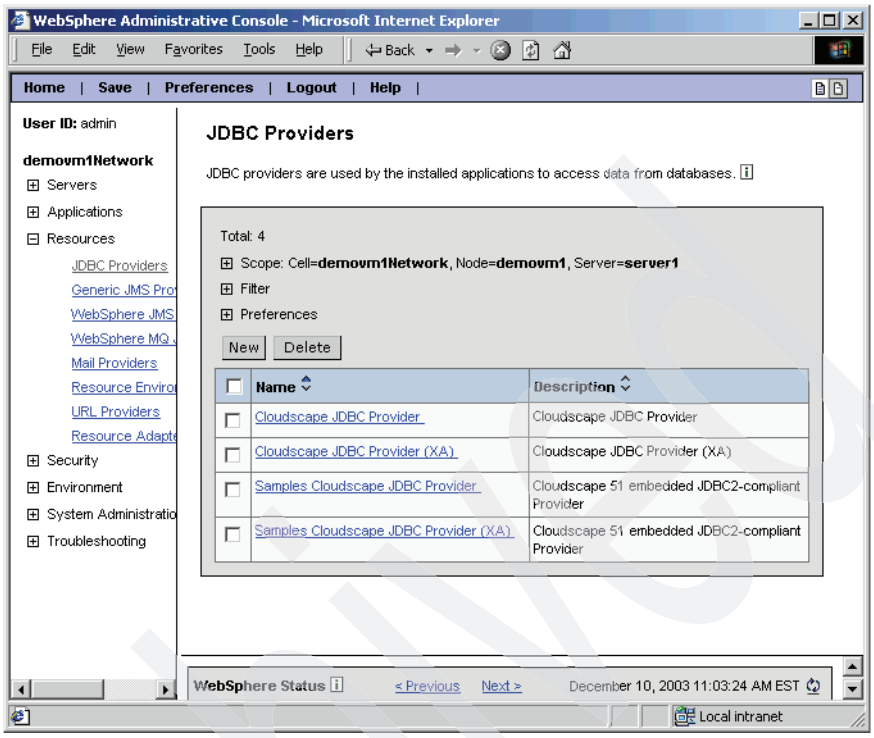


Figure 7-5 JDBC providers

4. Select **New** to create a new JDBC provider.

Archive

5. Use the list boxes to select the type of provider you want to create. See Figure 7-6.

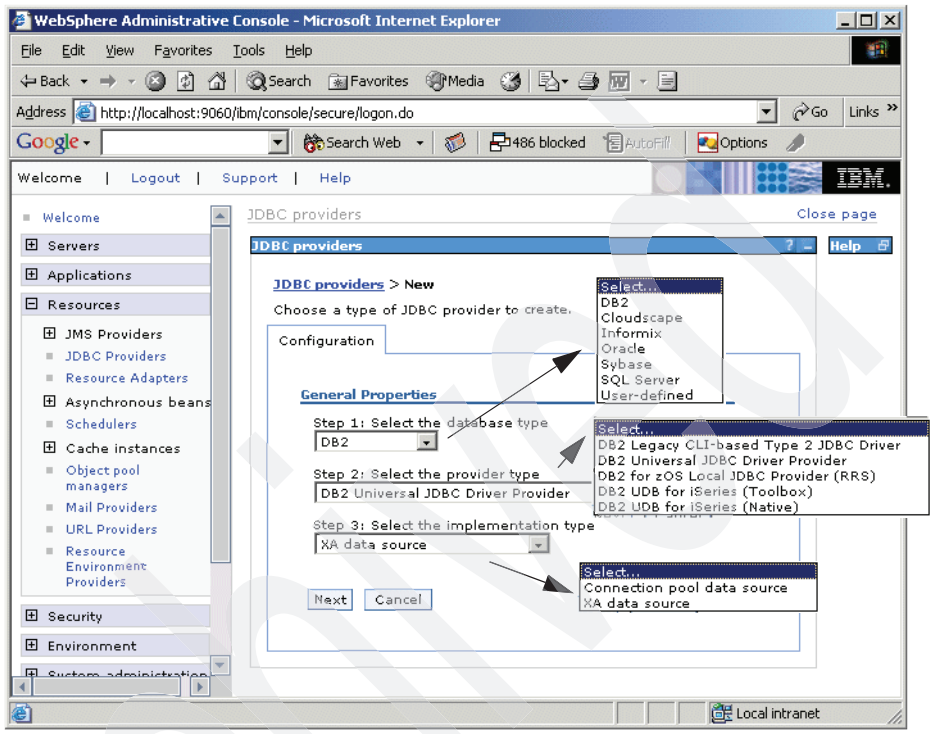


Figure 7-6 Define a new JDBC provider: Panel 1

- 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.

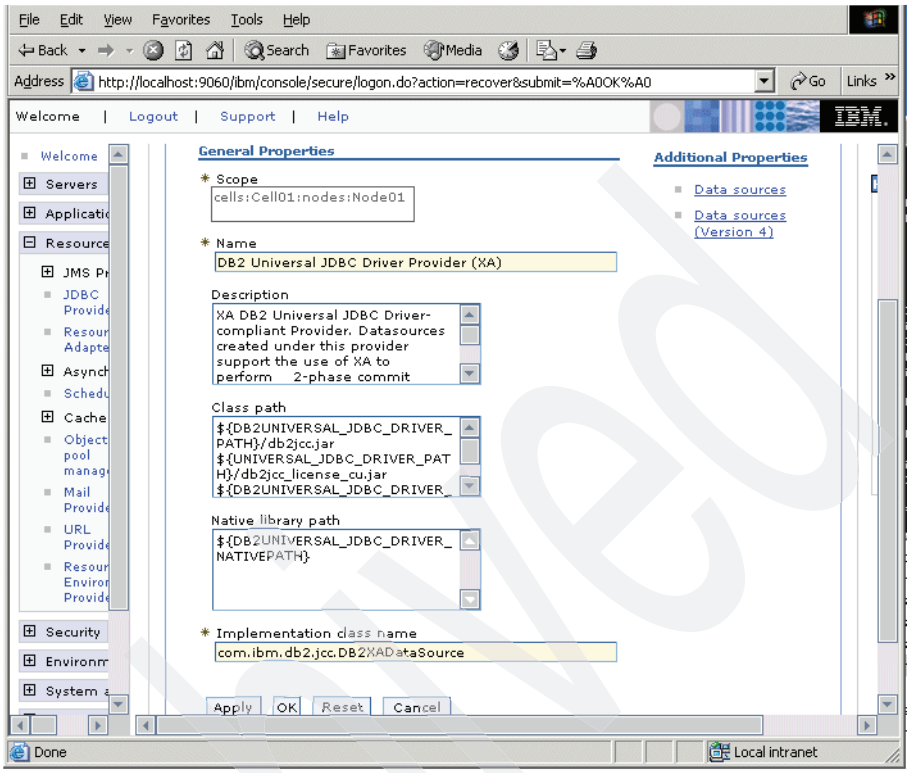


Figure 7-7 Define a new JDBC provider: Panel 2

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 JBDC provider chosen uses non-Java, or native, libraries.

Archived

– 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**.

Archived

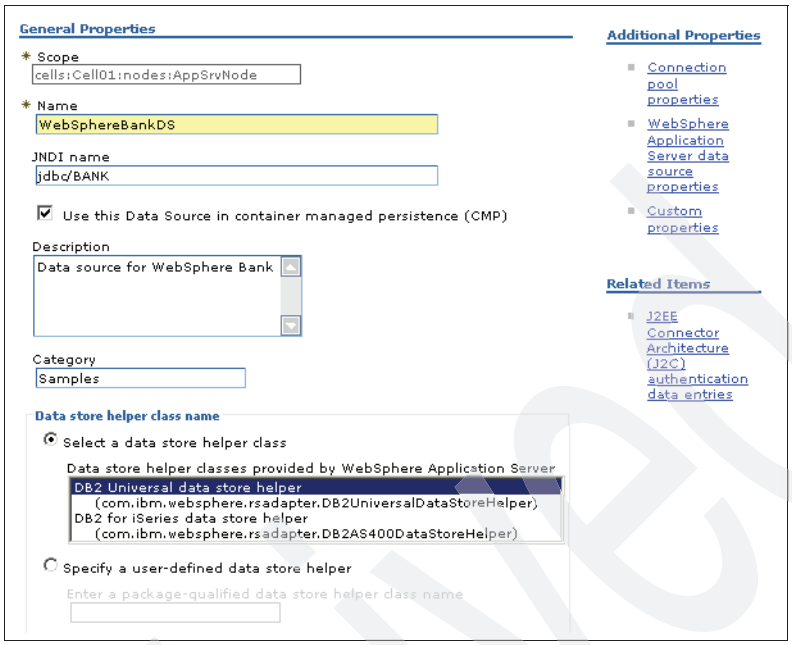


Figure 7-8 Data source general properties

- 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.

Archived

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.rsadapter.<database>DataStoreHelper
  ```

  For example, if the JDBC provider is DB2, then the default DataStoreHelper class is:

  ```
  com.ibm.websphere.rsadapter.DB2DataStoreHelper
  ```

  You can change the value to refer to a specific subclass of this DataStoreHelper if necessary.

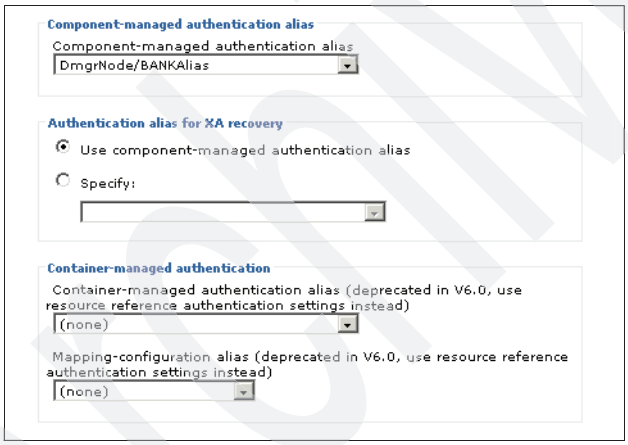


Figure 7-9 Data source security settings

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.

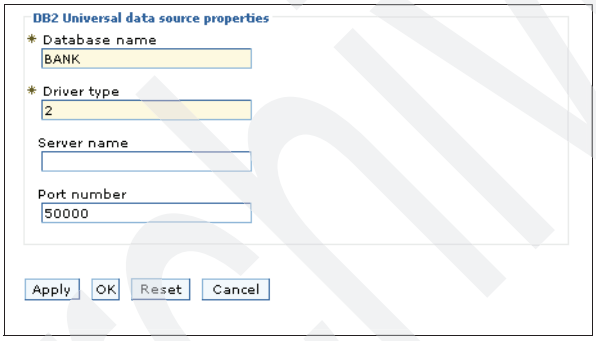


Figure 7-10 Data source-specific properties

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.

Archived

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.

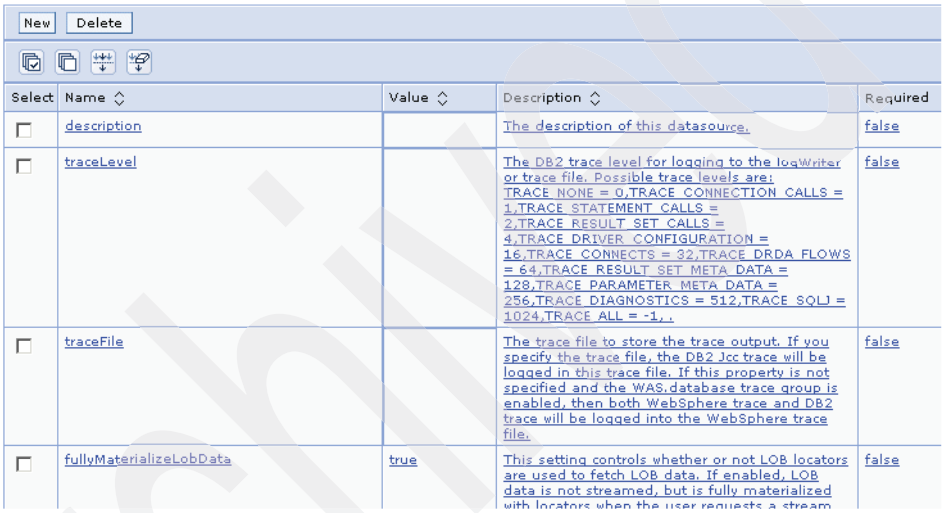


Figure 7-11 Data Source custom properties

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.

Archived

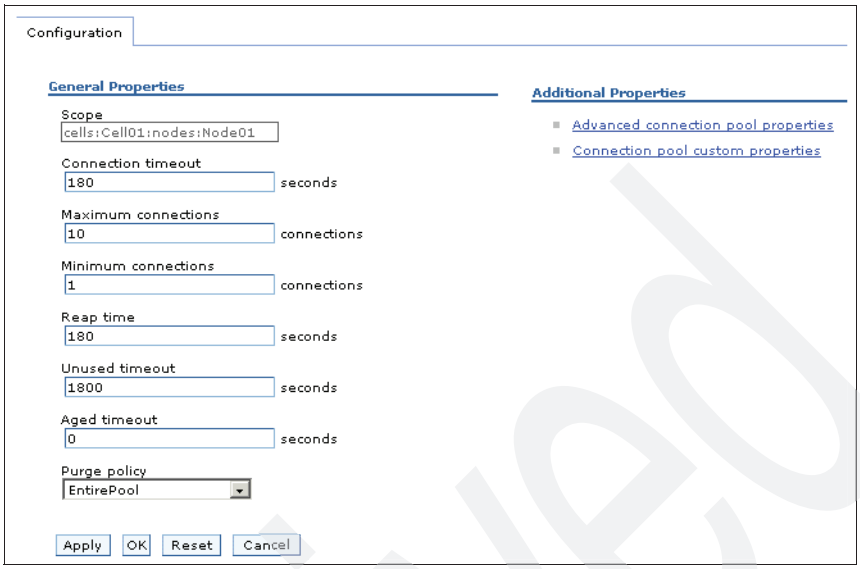


Figure 7-12 Data source connection pool properties

- 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.

Archived

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.

Archive

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.

Archived

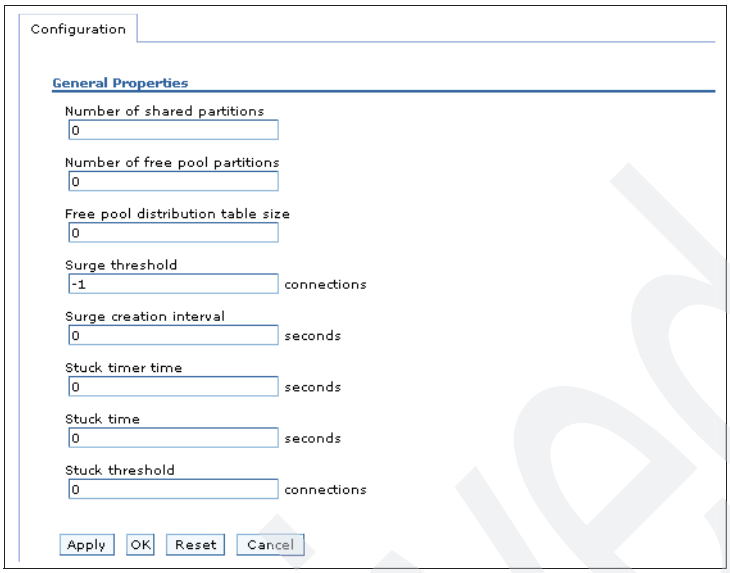


Figure 7-13 Advanced connection pool properties

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.

Archived

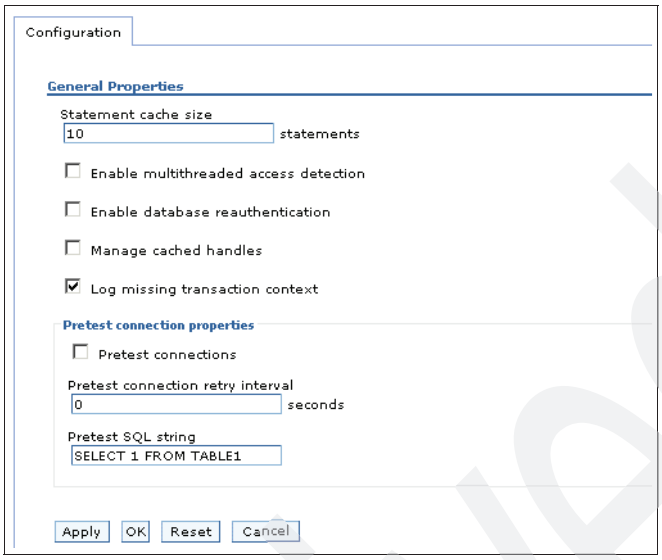


Figure 7-14 WebSphere data source custom properties

- 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.

Archived

- 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

Archived

- 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.

Archived

– 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.

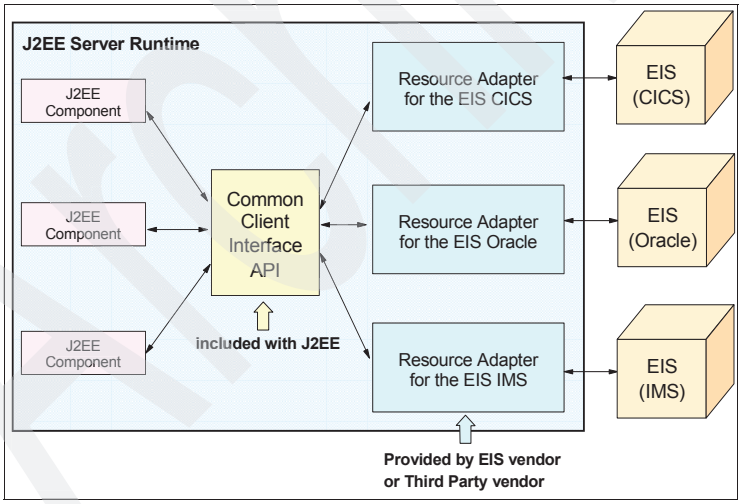


Figure 7-15 Common Client Interface API

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

Archived

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.

Archived

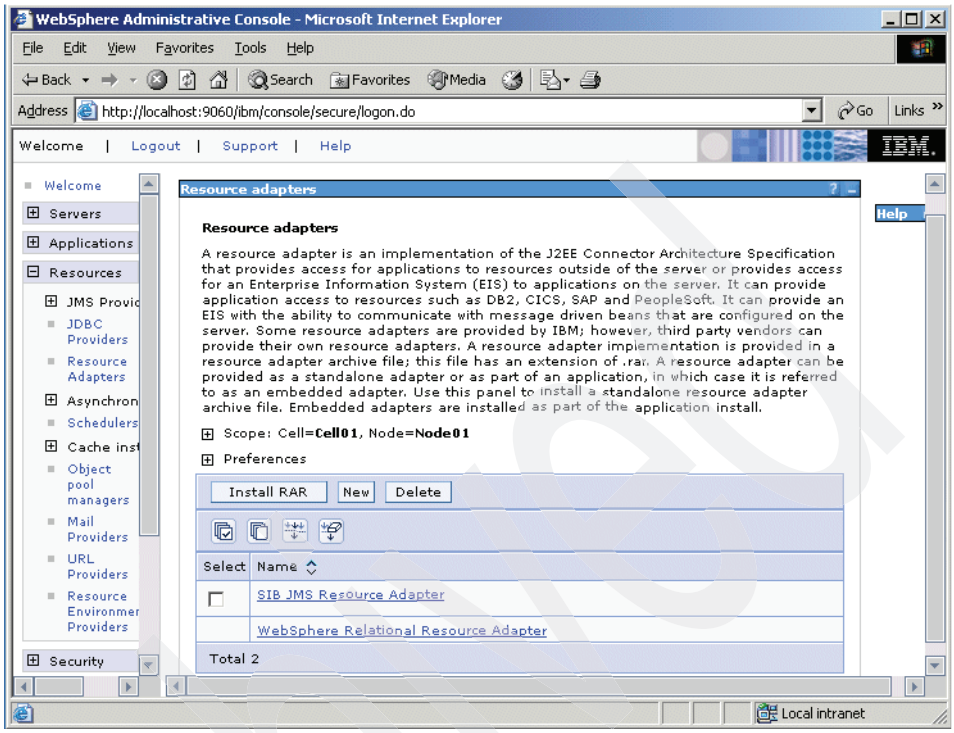


Figure 7-16 JCA resource adapters

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.

Archive

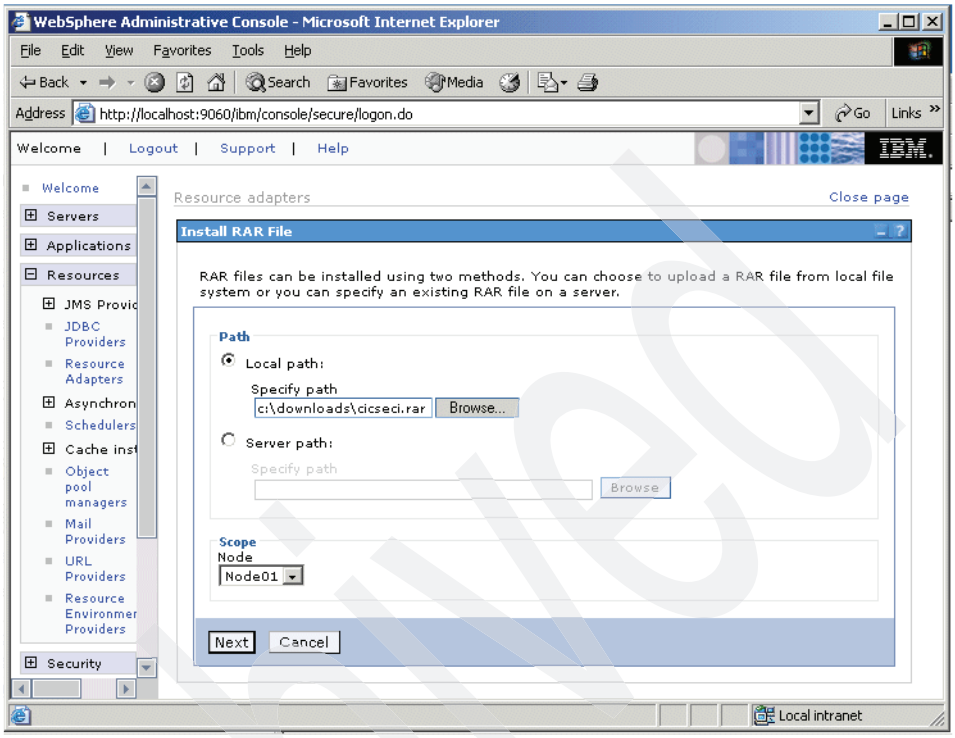


Figure 7-17 RAR file location

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.

Archive

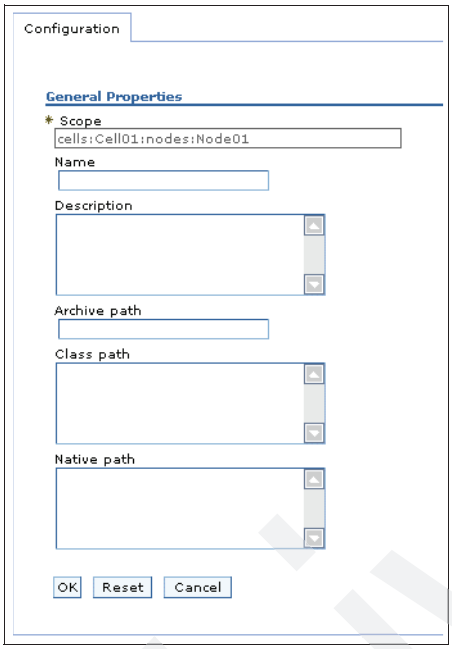


Figure 7-18 JCA resource adapter properties

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.

Archived

– 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

Archive

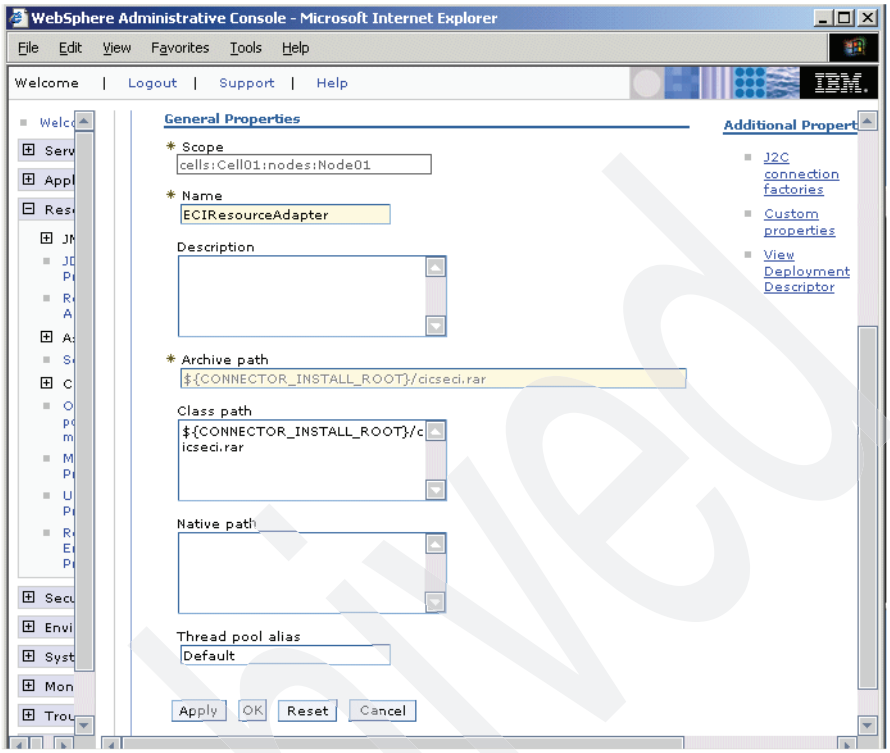


Figure 7-19 Configurable resources for the JCA connector

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.

Archived

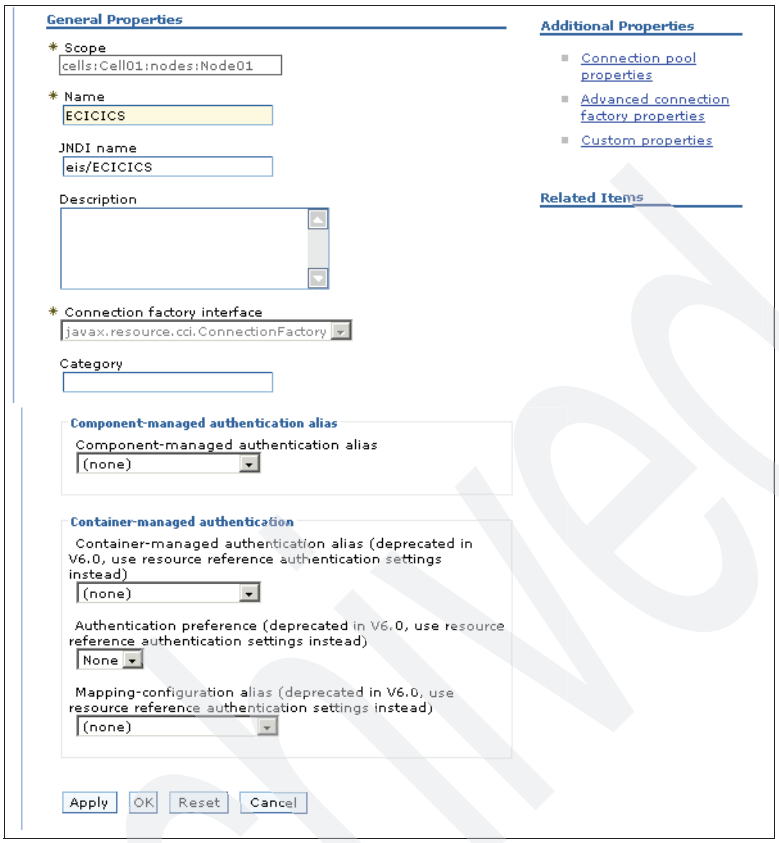


Figure 7-20 J2C connection factory properties

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>`

Archived

– 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.

Aricred

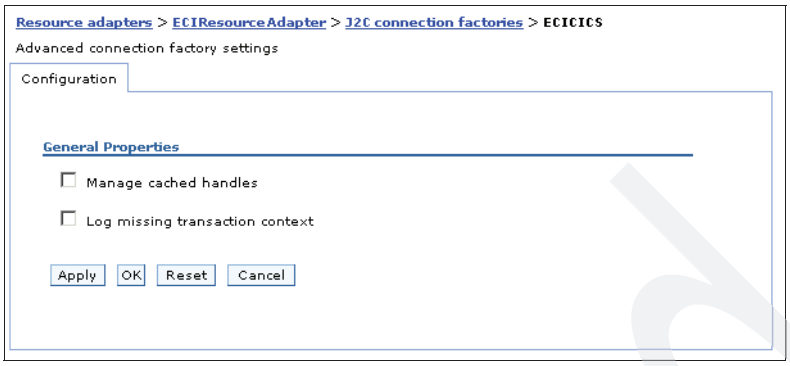


Figure 7-21 Advanced connection factory properties

- 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 connectionFactory =
      (javax.resource.cci.ConnectionFactory) env.lookup("eis/ref/ECICICS");

   // get a connection to the EIS
   javax.resource.cci.Connection connection =
connectionFactory.getConnection();

   // create an interaction and a CICS ECI specific interaction spec
   javax.resource.cci.Interaction interaction =
connection.createInteraction();
   com.ibm.connector2.cics.ECIInteractionSpec interactionSpec = new
com.ibm.connector2.cics.ECIInteractionSpec();

   // create the comm area record
   source = (source.trim().toUpperCase()+"          ").substring(0, 12);
   GenericRecord record = new GenericRecord((source).getBytes("IBM037"));

   // set the CICS program name we want to call
   interactionSpec.setFunctionName("CALCRATE");

   // invoke the CICS program
   interaction.execute(interactionSpec, record, record);

   // close the interation 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.

Archived

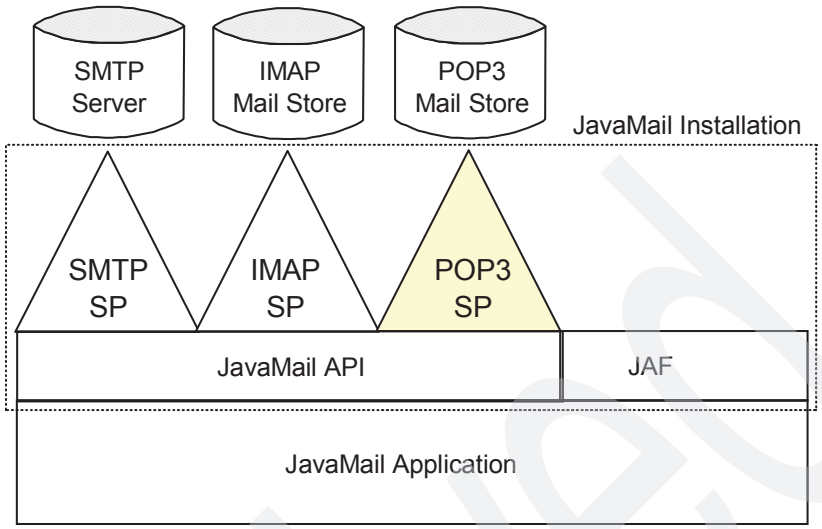


Figure 7-22 JavaMail components

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.

Archived

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.

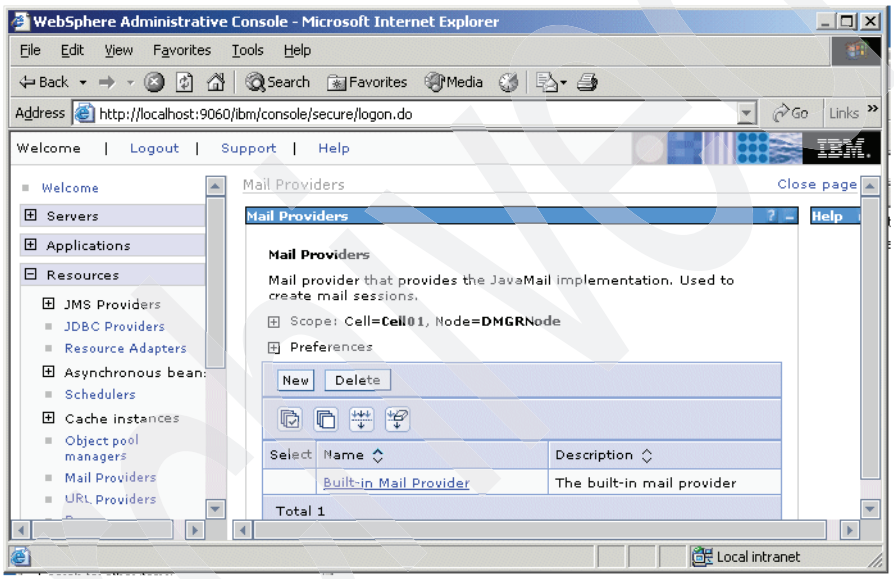


Figure 7-23 Mail provider page

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.

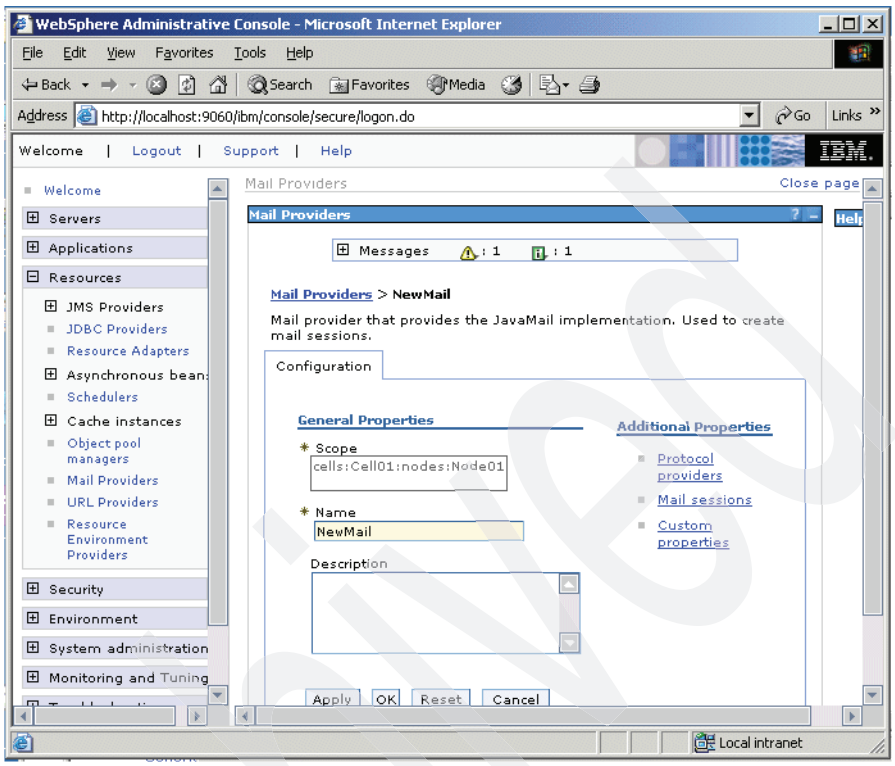


Figure 7-24 Mail Provider general properties

6. Click **Protocol Providers** under the Additional Properties section.
7. Click **New** to add a protocol provider.

Archive

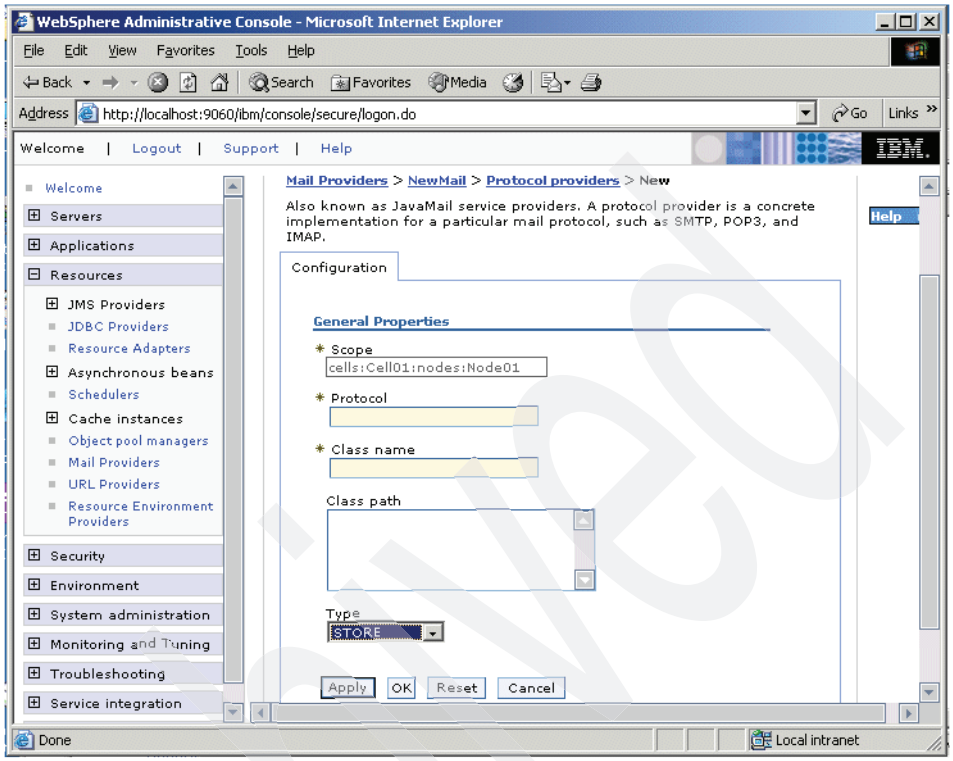


Figure 7-25 Protocol provider configuration page

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.

Archived

For guidance, you can look at the protocol providers provided with the built-in mail provider, shown in Figure 7-26.

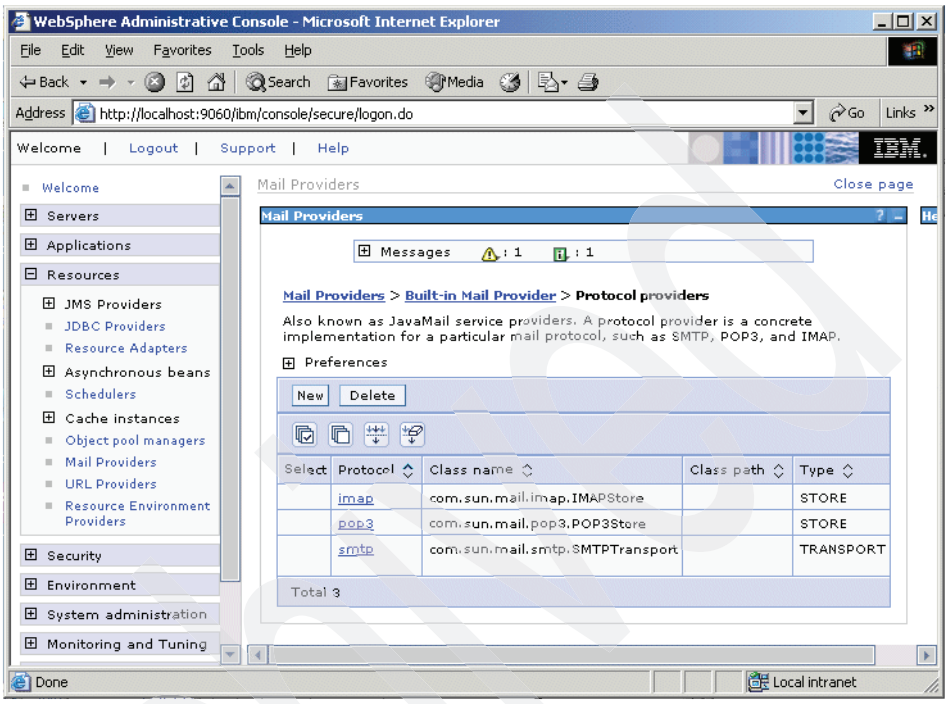


Figure 7-26 Protocol providers

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.

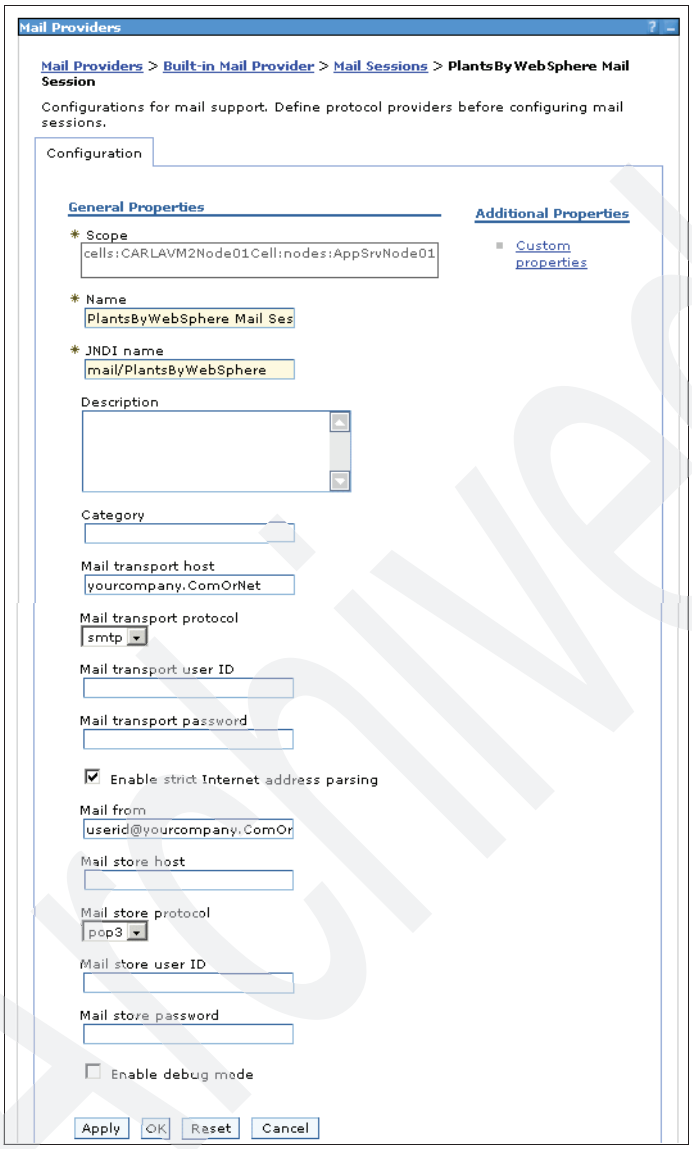


Figure 7-27 Configuration page for the mail session

Define the following properties, according to your situation:

– Name

Type an administrative name for the JavaMail session object.

Archived

– 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`.

Archived

– 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
```


Archived

```
 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.net.URLConnection classes. It enables communication between the application and a URL resource that is served by that particular protocol.

A URL provider named Default URL Provider is included in the initial WebSphere configuration. This provider utilizes the URL support provided by the IBM JDK. Any URL resource with protocols based on the Java 2 Standard Edition 1.3.1, such as HTTP, FTP or File, can use the default URL provider.

You can also plug in your own URL provider for another protocol not supported by the JDK.

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.

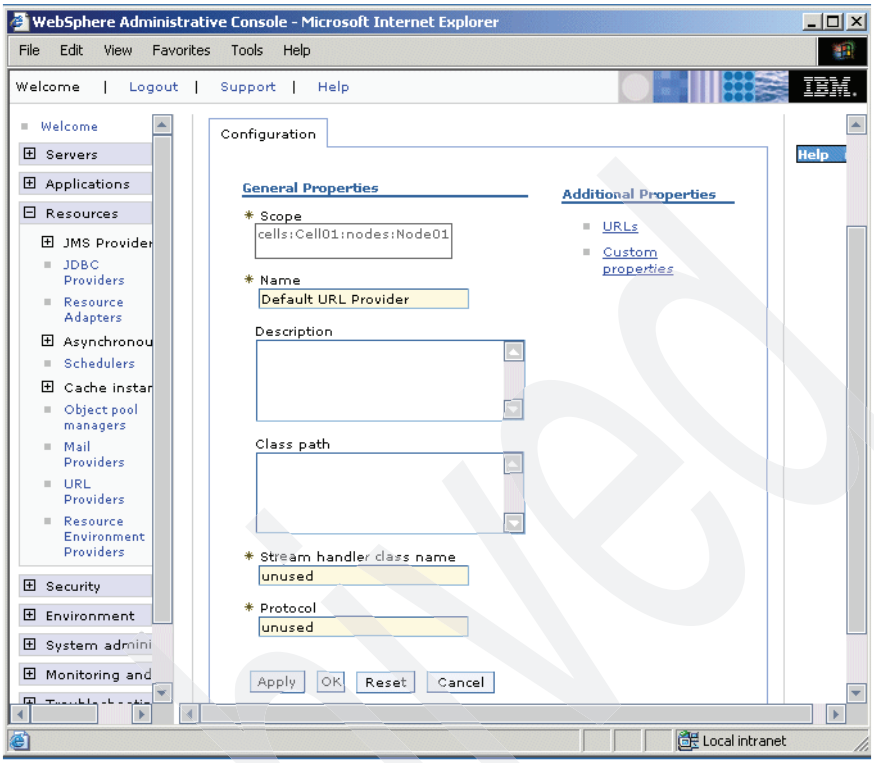


Figure 7-28 URL provider configuration page

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.

Archived

– 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.

Archived

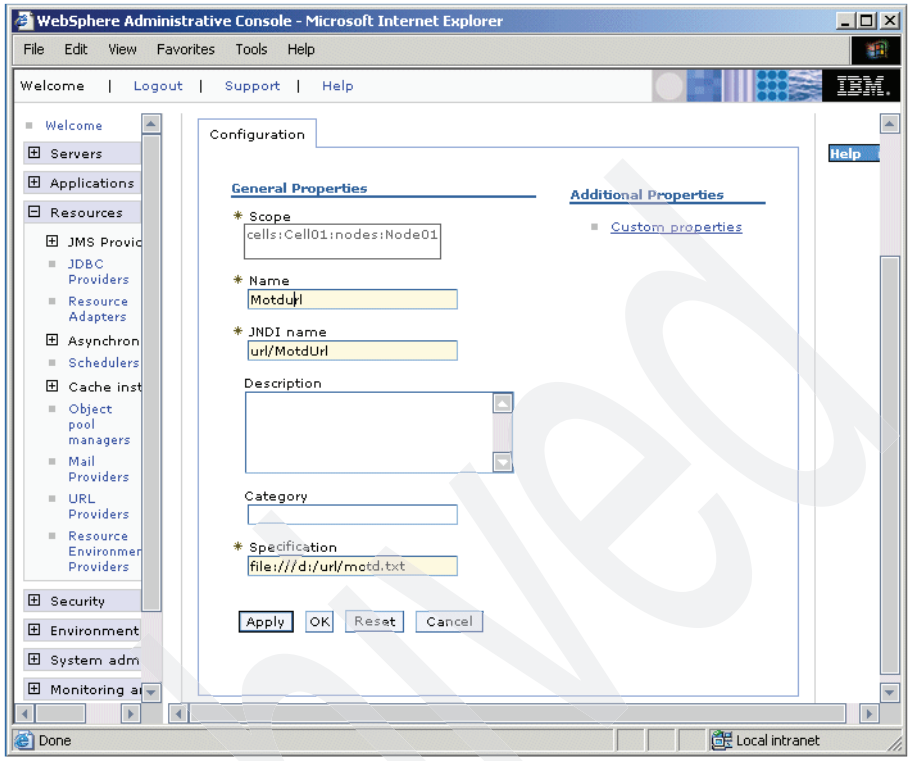


Figure 7-29 Defining URLs

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.

Archived

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.

Archived

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);
.....
```

Archive

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"/>
```

Archived

```
   <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.

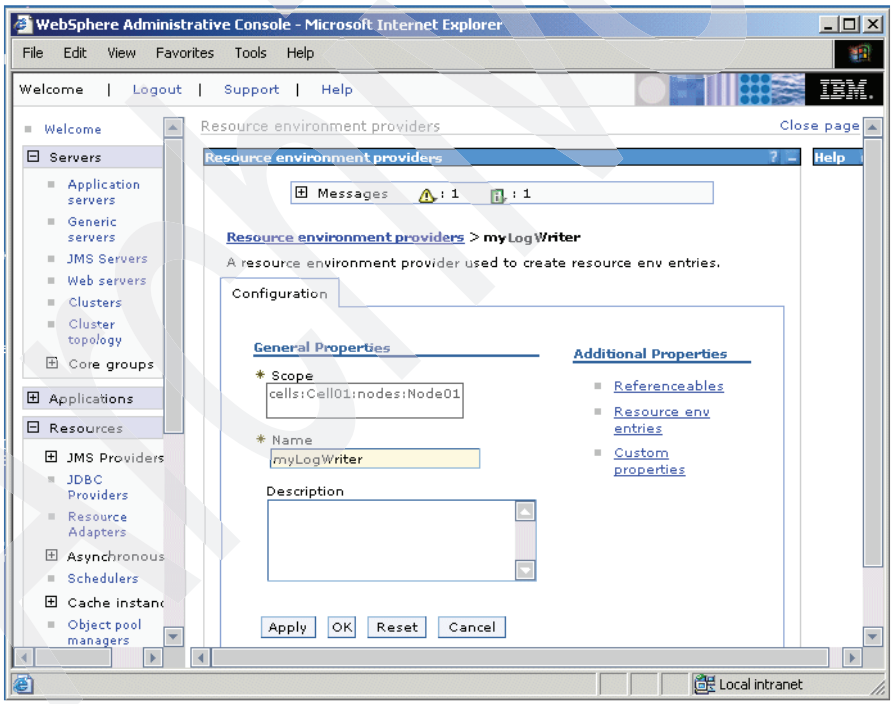


Figure 7-30 Creating a resource environment provider

6. Click **Referenceables** in the Additional Properties section.

Archived

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.

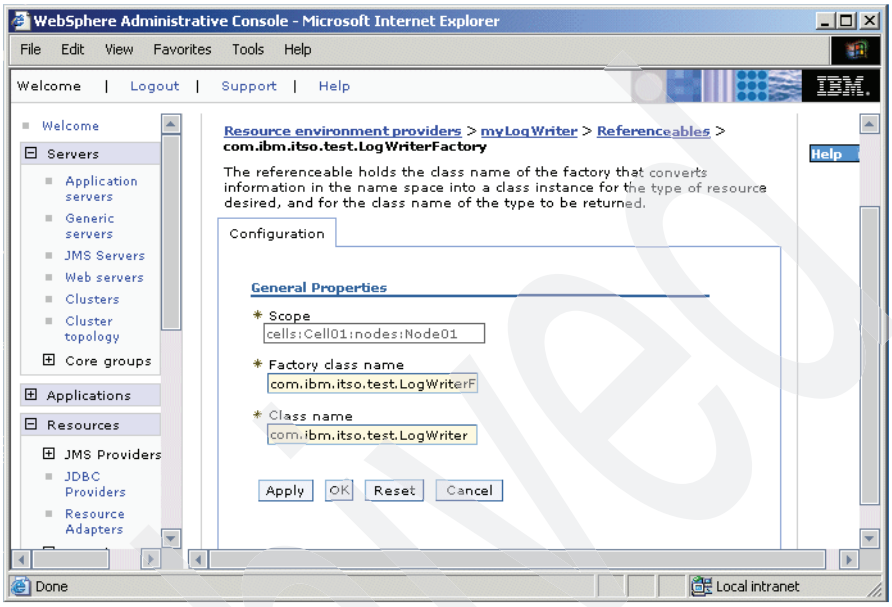


Figure 7-31 Create a reference

   – 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.

Archived

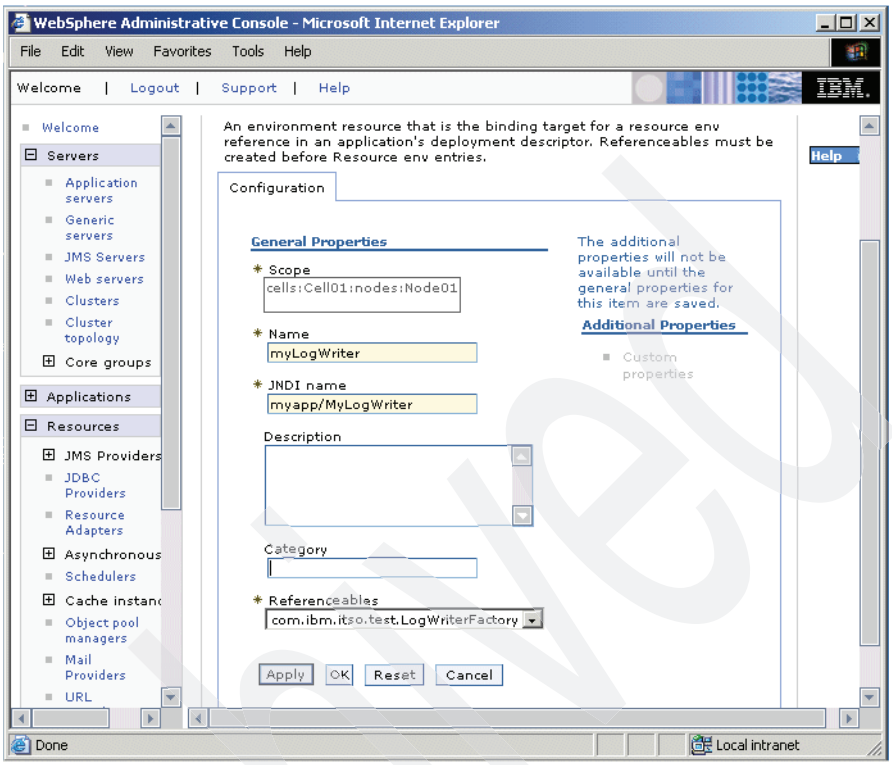


Figure 7-32 Creating a resource environment entry

- 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.

Archive

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 authetication 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

Archive

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

Archived

8

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.

© Copyright IBM Corp. 2005. All rights reserved.

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.

Archived

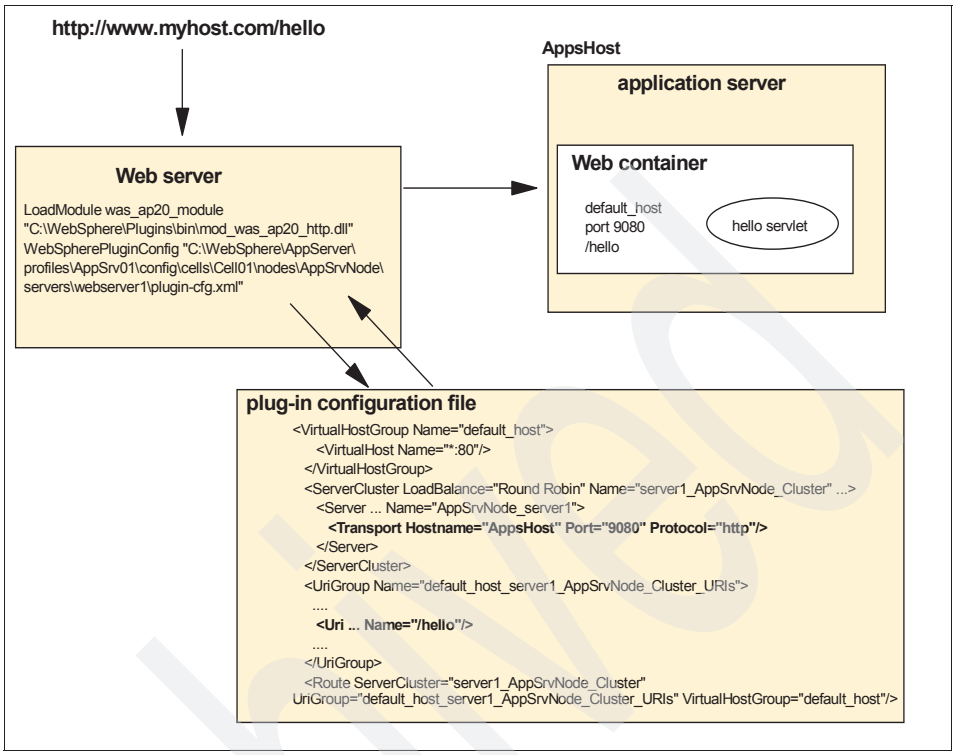


Figure 8-1 Web server plug-in routing

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 association 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.

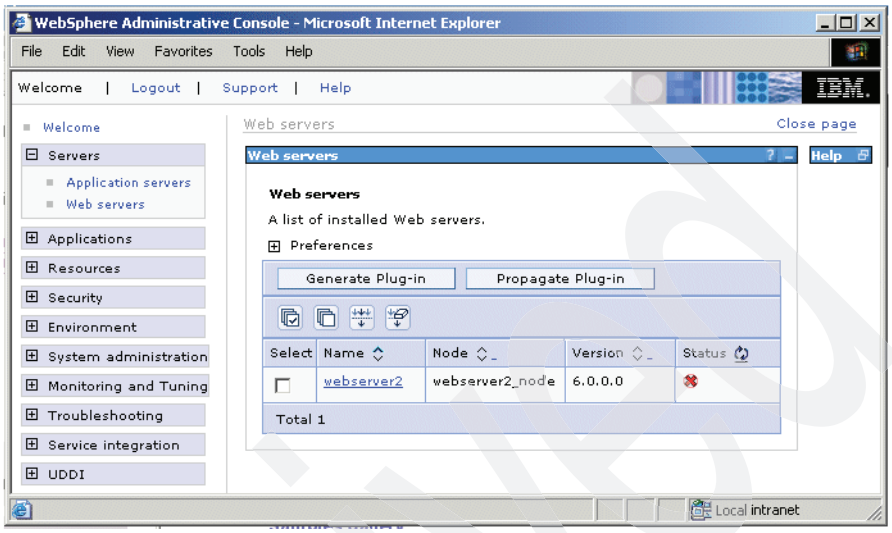


Figure 8-2 Web server definition

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 `configure`*`Web_server_name`* script. There is a potential conflict between a shell environment variable, the WAS_USER_SCRIPT variable, and the real default profile. The script always works against the default profile. However, if the WAS_USER_SCRIPT environment variable is set, a conflict arises as the script attempts to work on the profile identified by the variable.

If you 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.

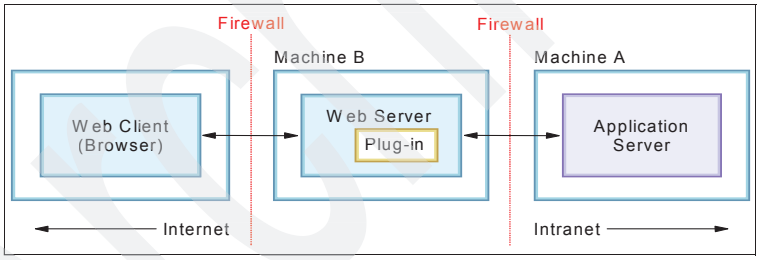


Figure 8-3 Remote Web server in a standalone server environment

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.

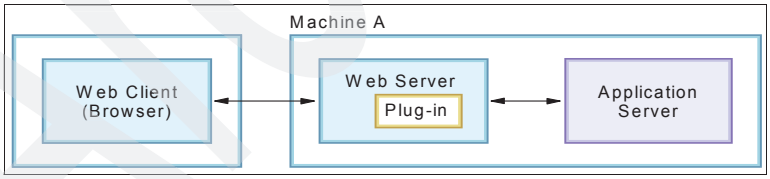


Figure 8-4 Local Web server in a standalone server environment

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/webserver1/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.

Archive

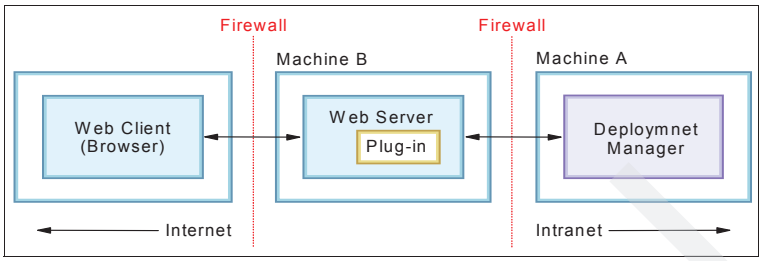


Figure 8-5 Remote Web server in a standalone server environment

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.

Archived

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.

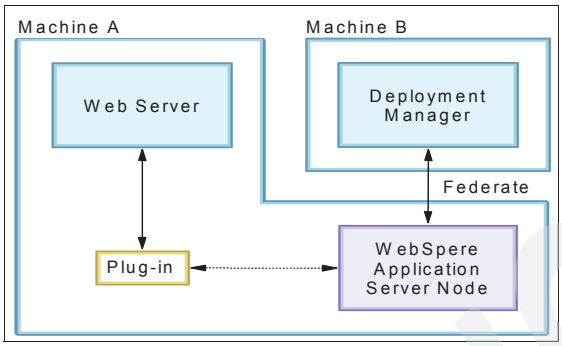


Figure 8-6 Web server installed locally on an application server system

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/webserver1/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>.`

Archived

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.

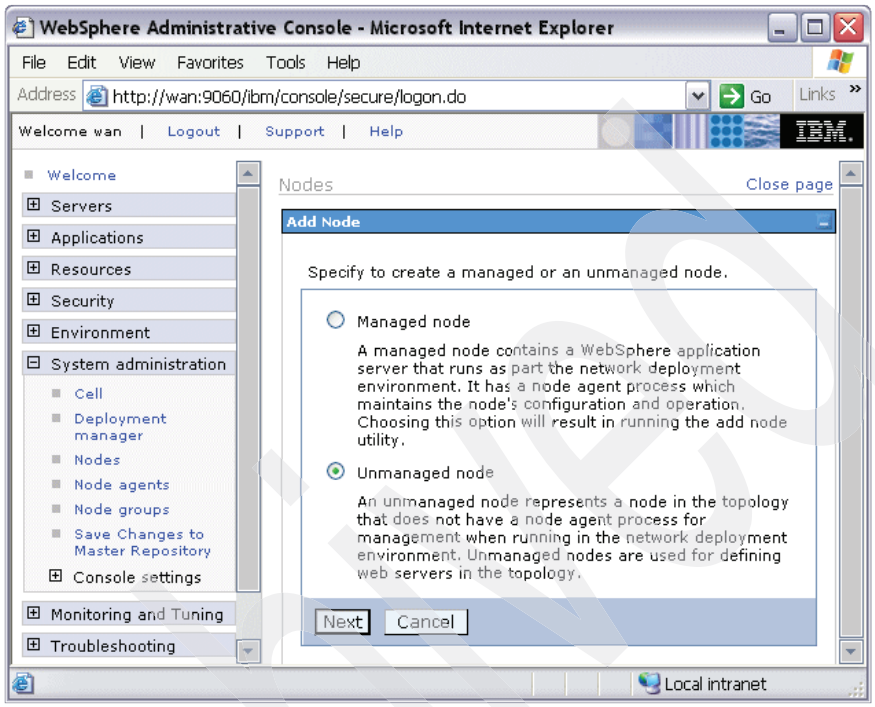


Figure 8-7 Add node page

4. Click **Next**.
5. Enter the following values in the General Properties page. See Figure 8-8 on page 391.
   a. Name

      Type a logical name for the node. The name must be unique within the cell. A node name usually is identical to the host name for the computer. However, you can make the node name different than the host name.

   b. Host name

      Enter the host name of the unmanaged node that is added to the configuration.

Archive

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

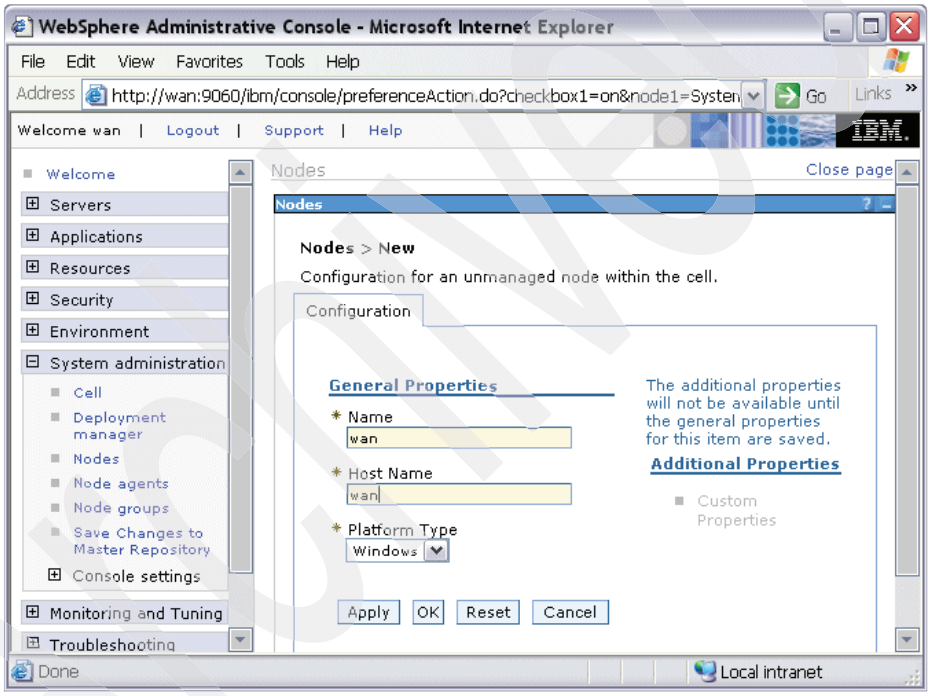


Figure 8-8 General properties for an unmanaged node

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.

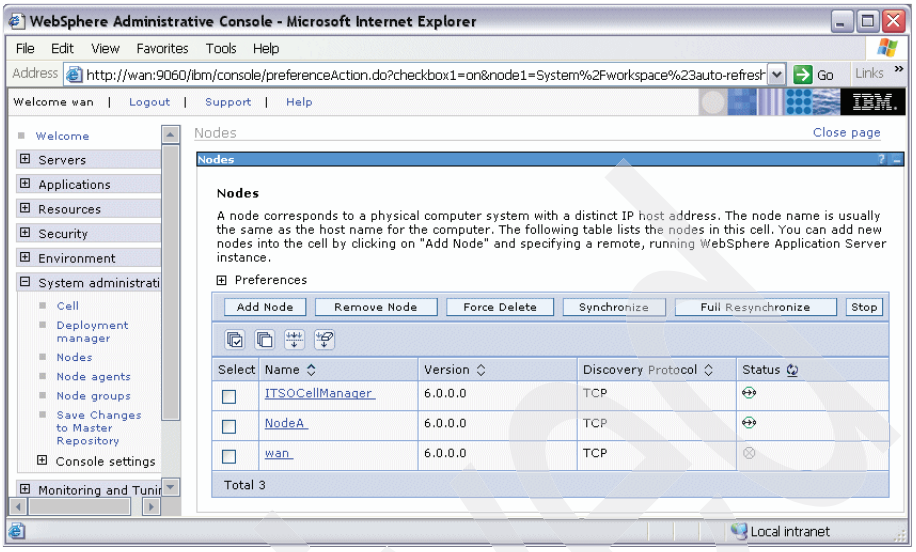


Figure 8-9 Nodes in a cell

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.

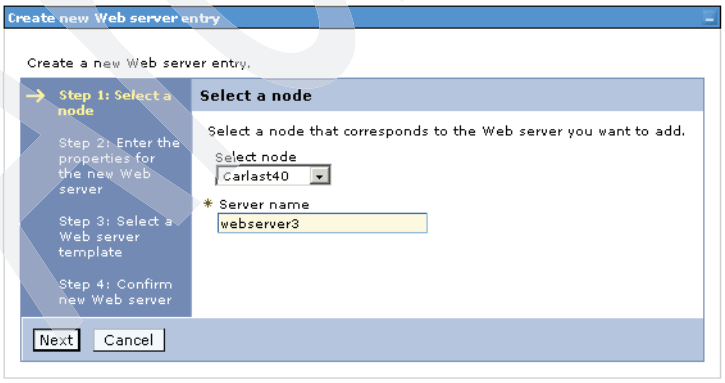


Figure 8-10 Defining a Web server: Step 1

4. Enter the properties for the Web server. See Figure 8-11 on page 393.

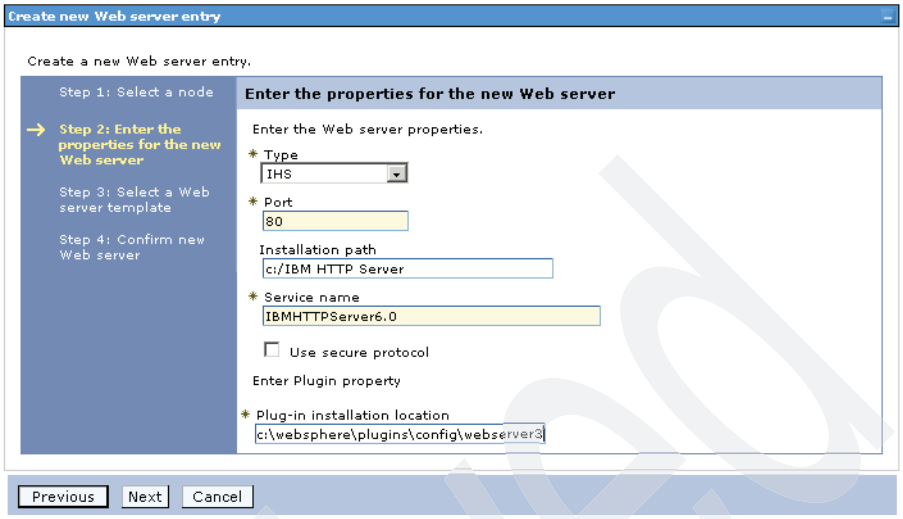


Figure 8-11 Defining a Web server: Step 2

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.

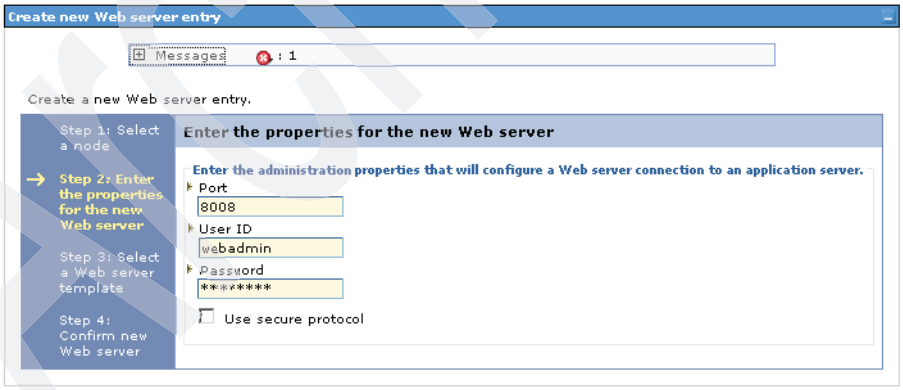


Figure 8-12 Defining a Web server: Step 3

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.

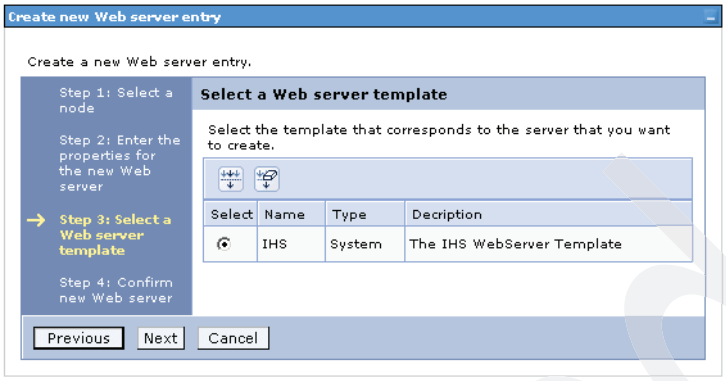


Figure 8-13 Defining a Web server: Step 4

7. Review the options as in Figure 8-14, and click **Finish**.

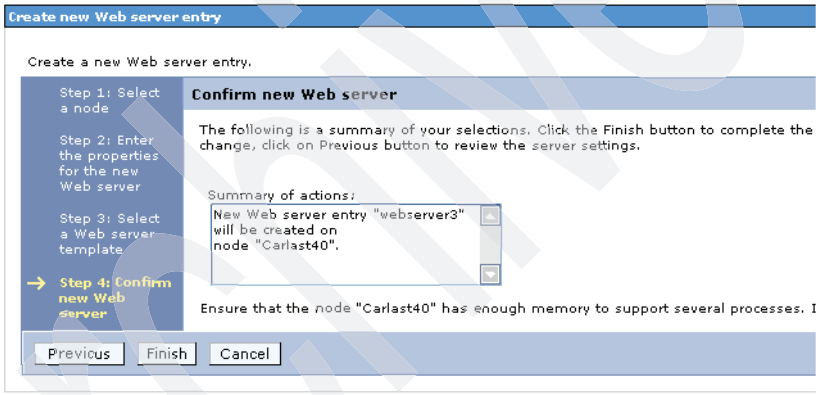


Figure 8-14 Defining a Web server: Step 5

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.

Archive

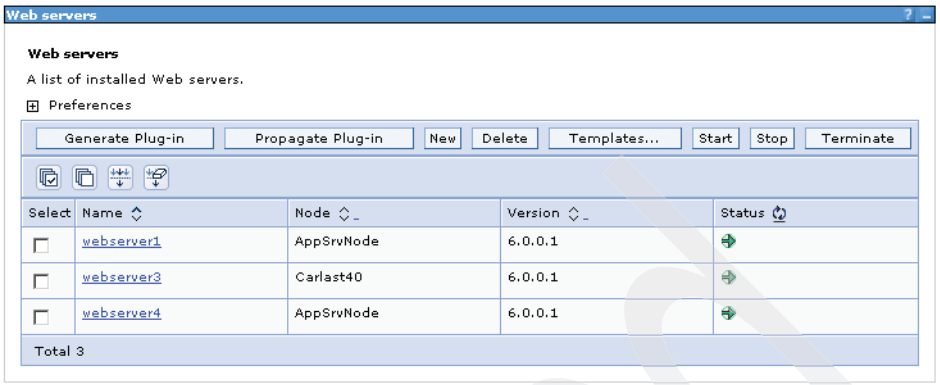


Figure 8-15 Web server status

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.**

Archive

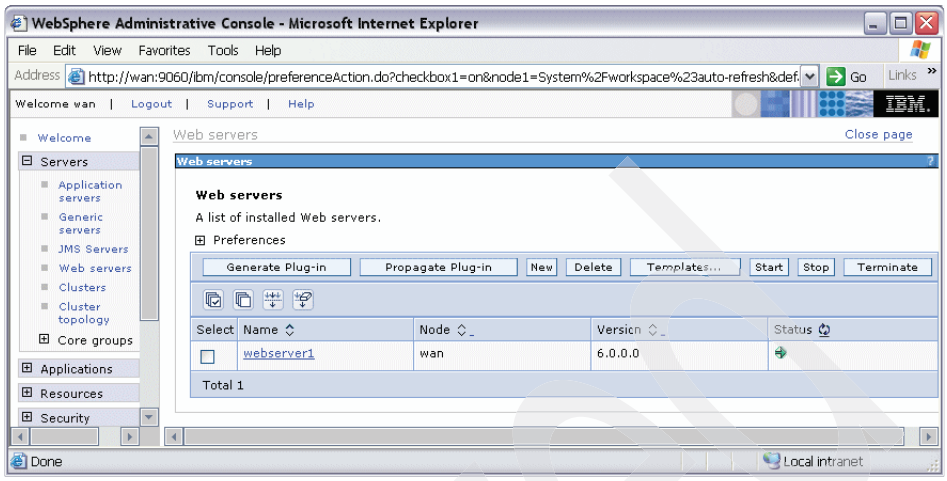


Figure 8-16 Web server definitions

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 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.

Archive

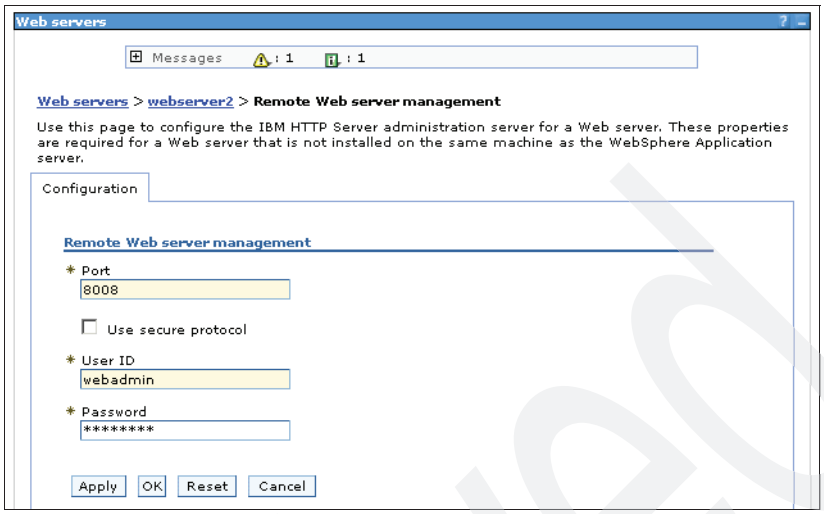


Figure 8-17 IHS remote management properties

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.

Archived

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.

Archived

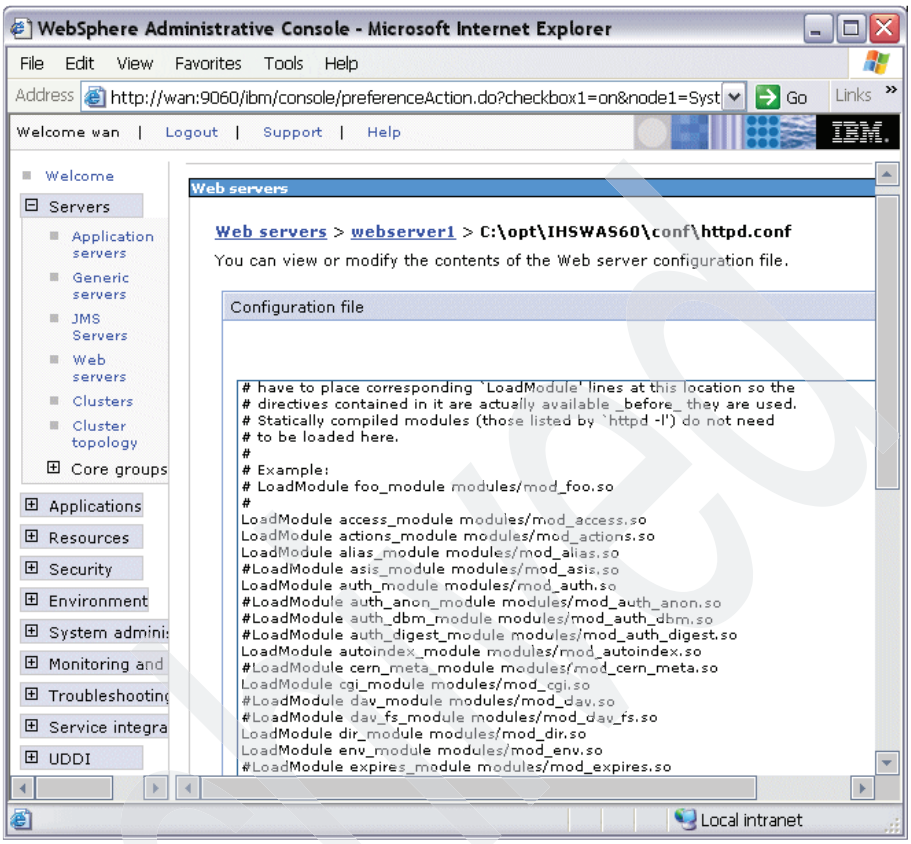


Figure 8-18 IBM HTTP Server configuration file httpd.conf

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.

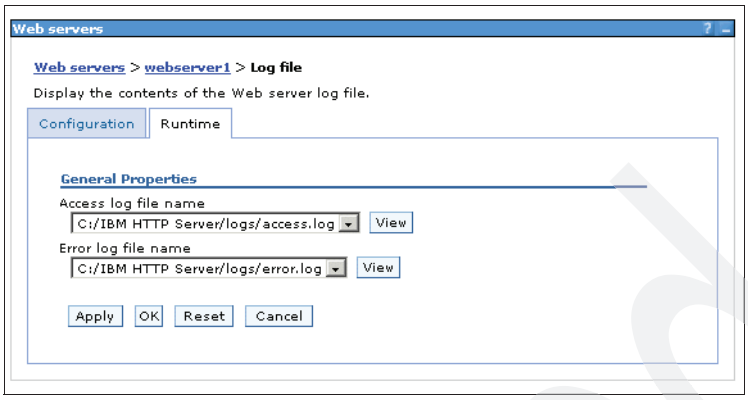


Figure 8-19 Web server Runtime page for logs

5. Click **View** beside the log you want to view. See Figure 8-20.

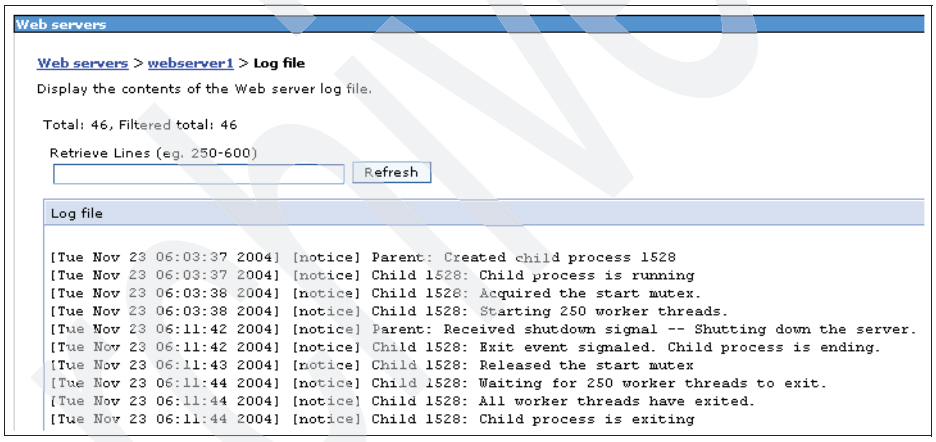


Figure 8-20 Viewing the error log

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.

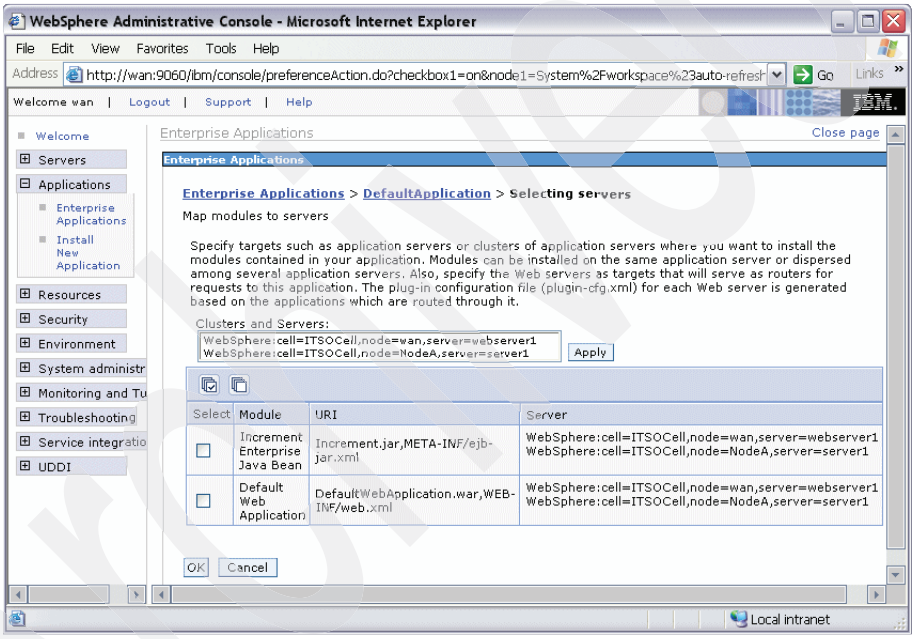


Figure 8-21 Map modules to selected servers

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 except 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.

Archived

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.

Archived

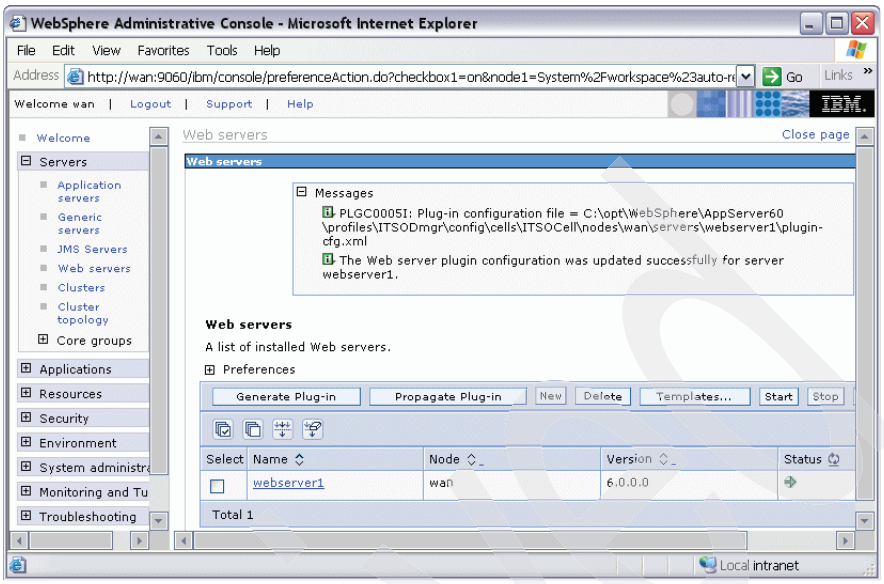


Figure 8-22 Web server definitions

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.

Archive

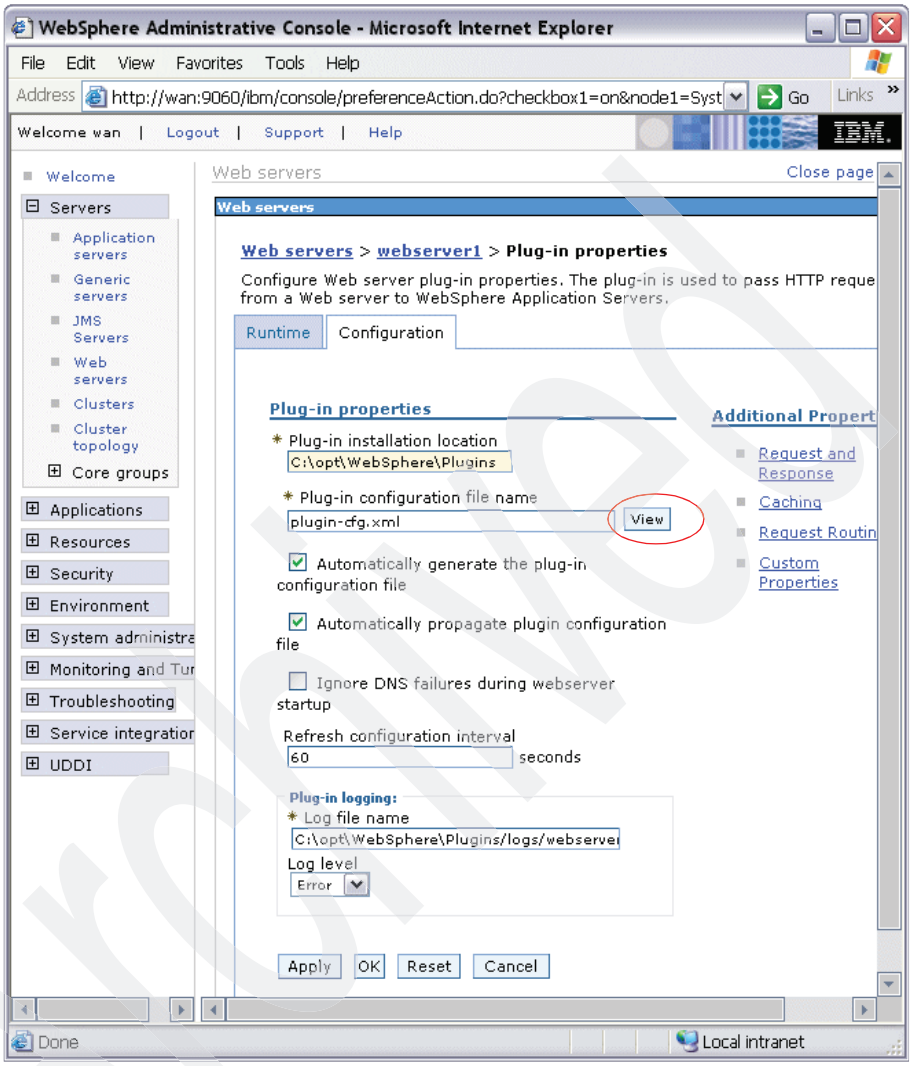


Figure 8-23 Plug-in properties

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.

Archived

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 webservrName, 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.

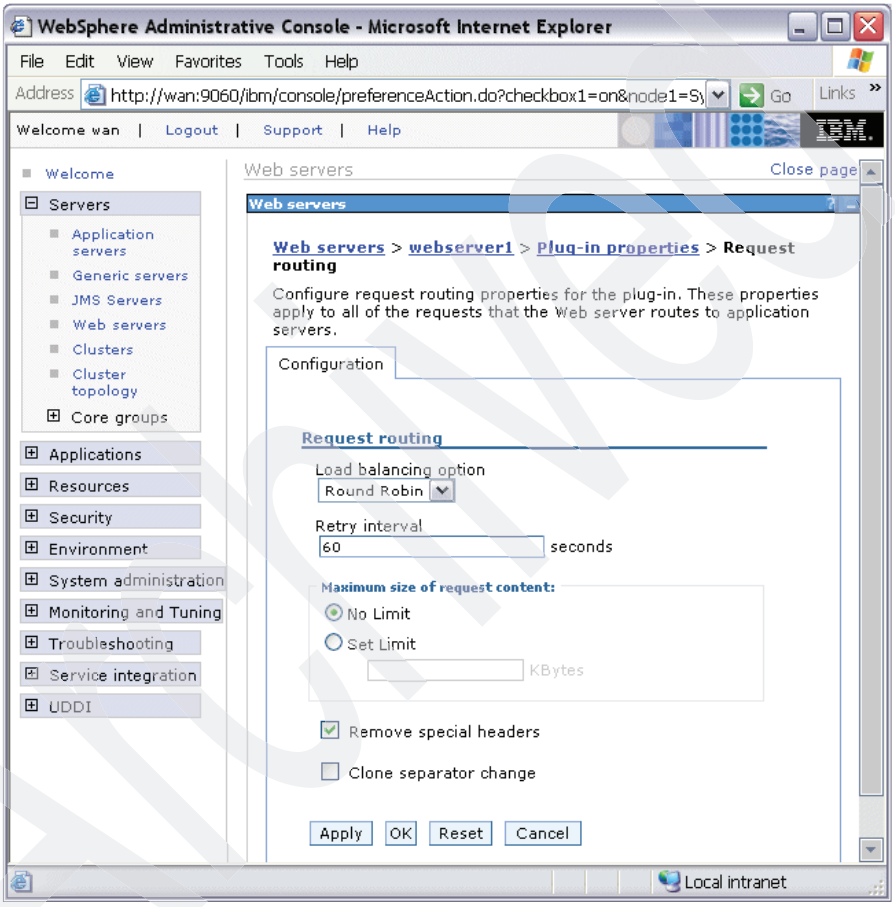


Figure 8-24 Request routing properties

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.

Archived

Archived

9

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

© Copyright IBM Corp. 2005. All rights reserved.

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.

Archived

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.

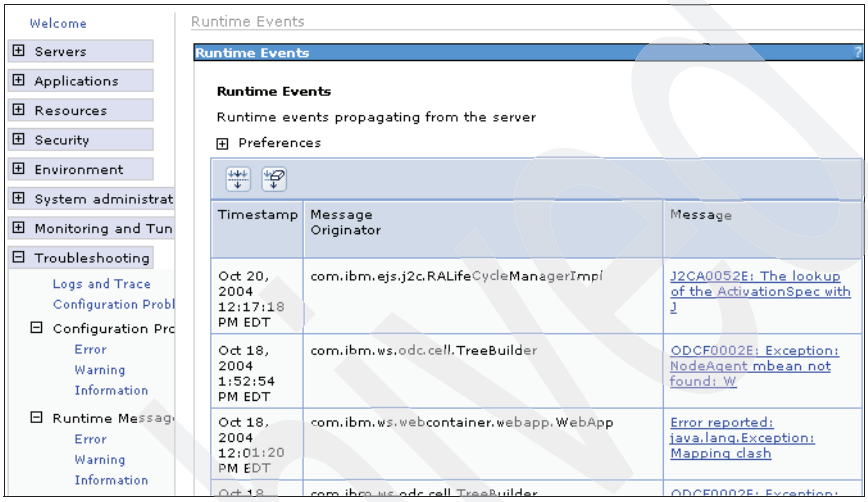


Figure 9-1 Runtime message errors view

View a message in detail by clicking the message text in the Message column. Figure 9-2 on page 420 shows sample message details.

Archive

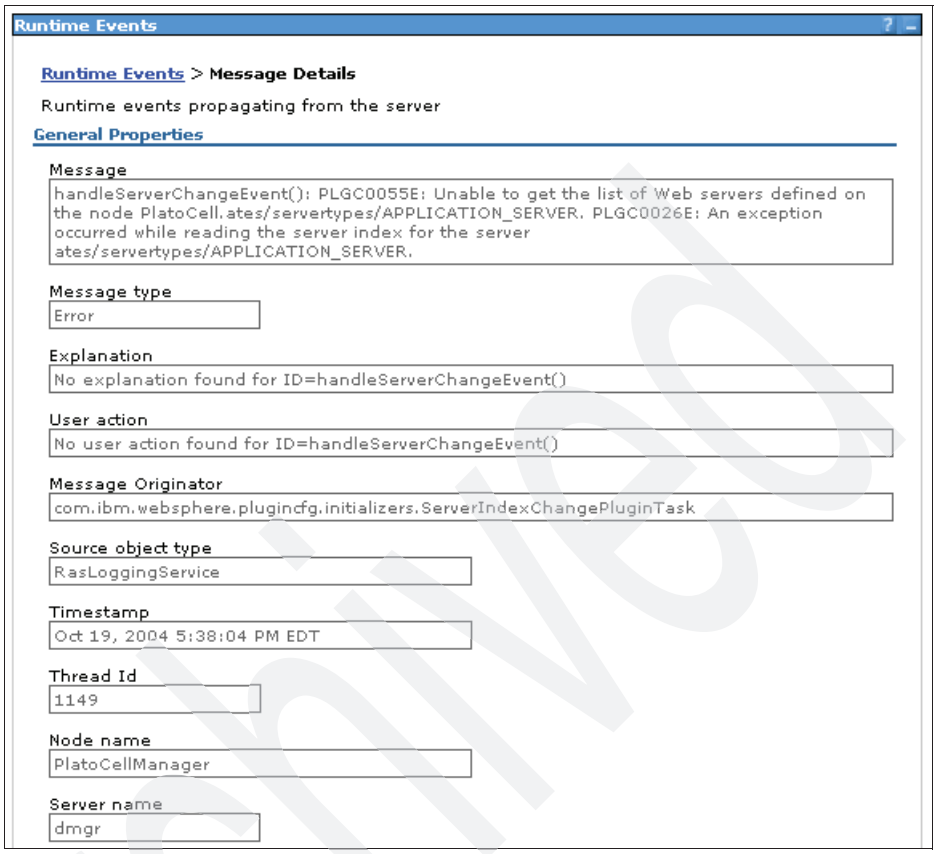


Figure 9-2 Message details

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

Archived

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 printing() 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.

Archived

5. Scroll through the window to display the attributes. See Figure 9-3.

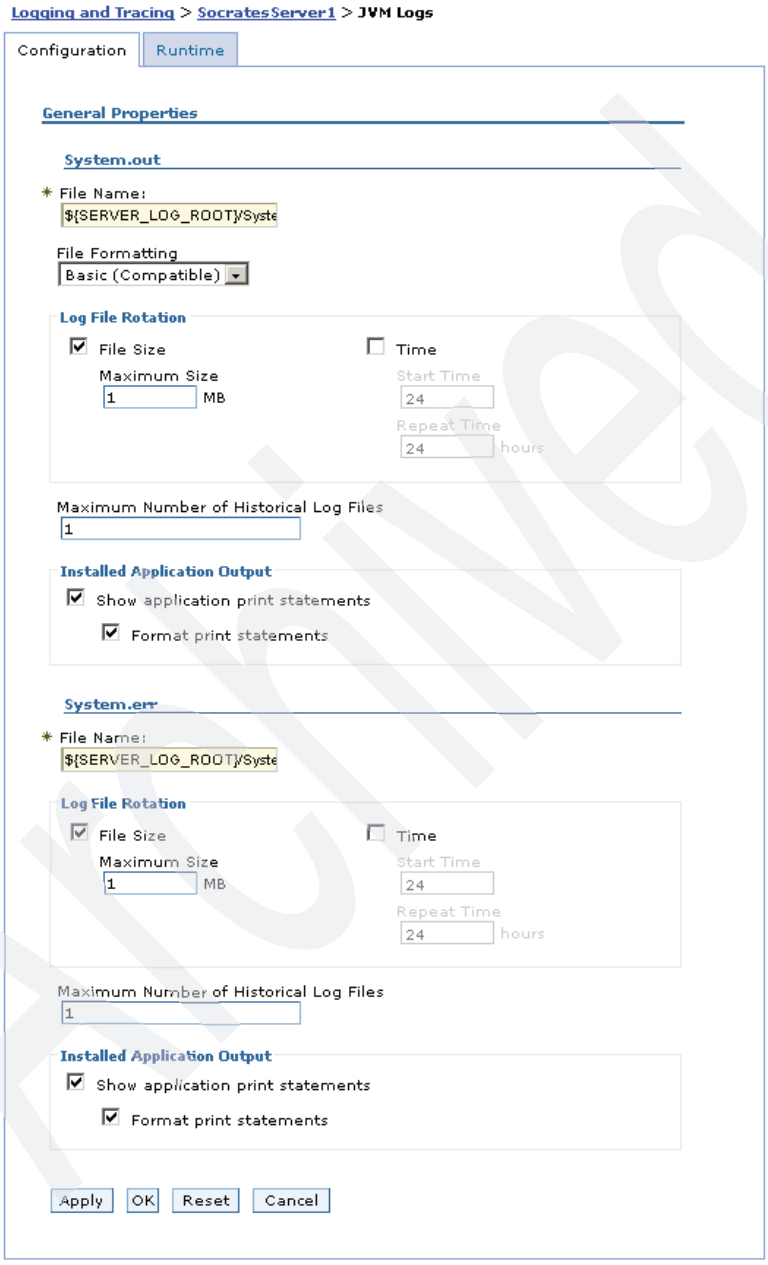


Figure 9-3 Configuring the JVM logs setting

Archived

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.

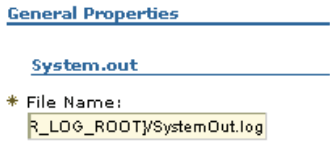


Figure 9-4 Enter the file name

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

Archived

– `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.

Archived

- **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.

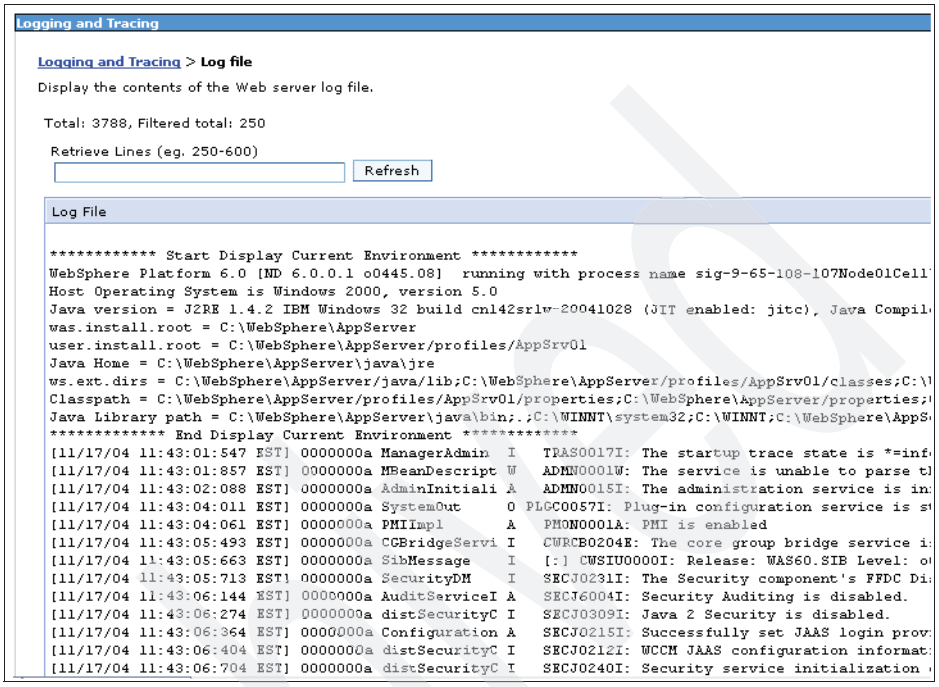


Figure 9-7 System.out log

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`	The type of the message or trace event, of which possible values are: **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.

Archive

- 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.

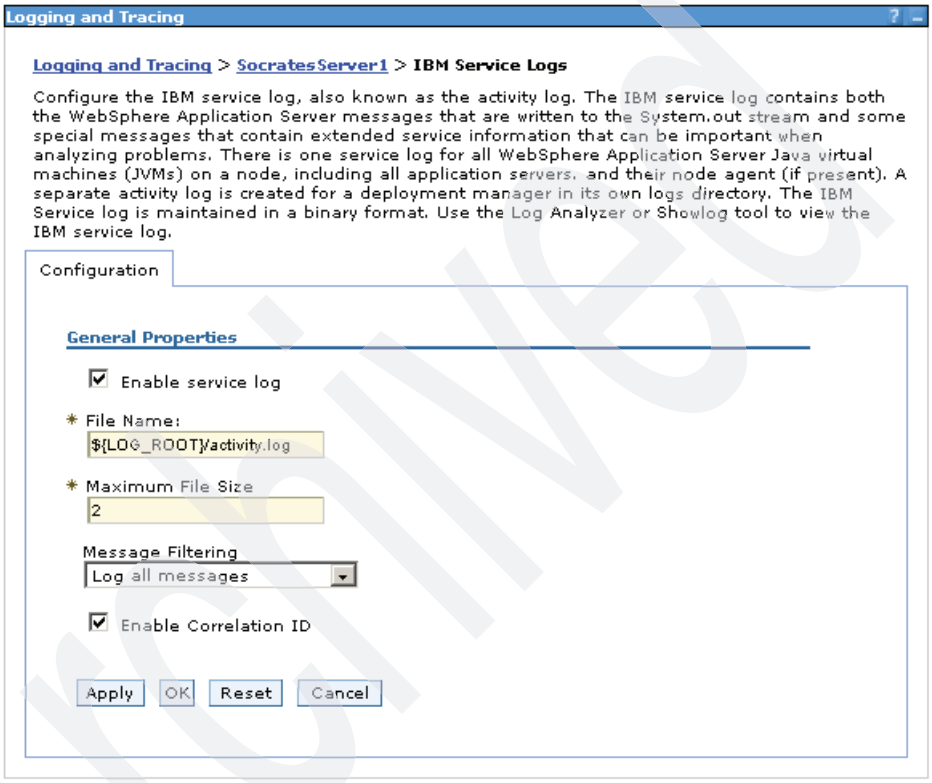


Figure 9-8 IBM Service log

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.

Archived

- **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.

Archive

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.

Archived

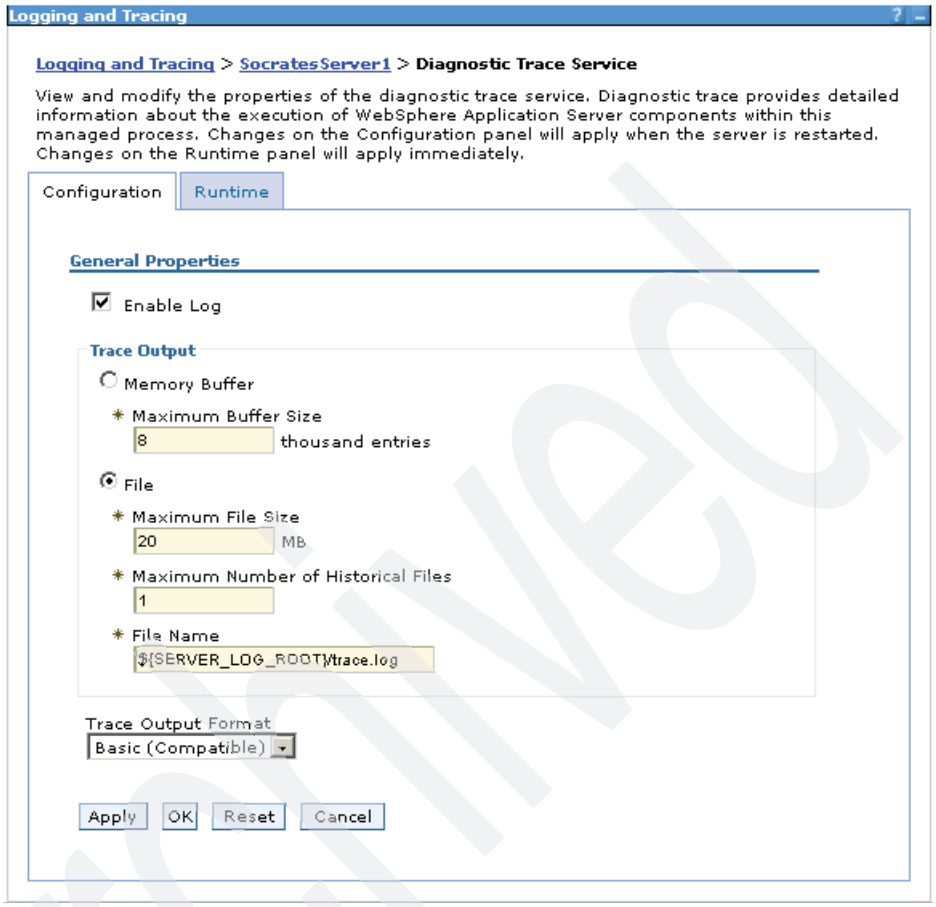


Figure 9-9 Trace service configuration

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.

Archived

- 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.

Archive



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.

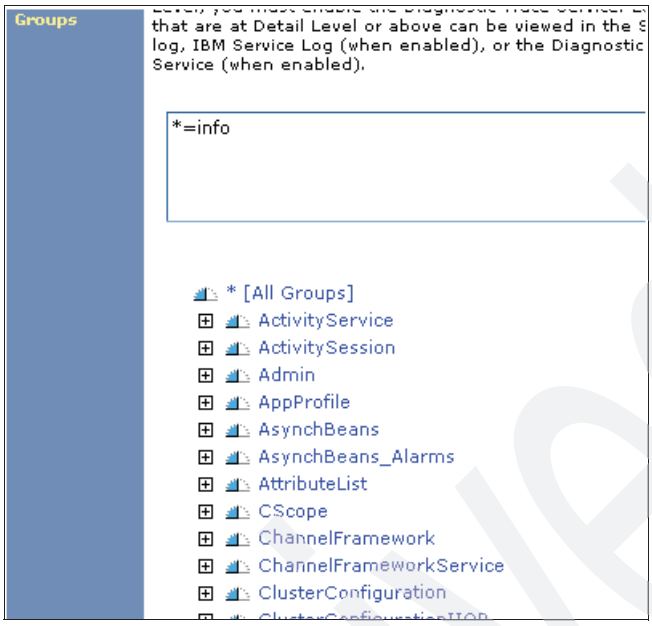


Figure 9-11 Log level details by Group

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.

Archived

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*	Logging is turned off. *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.
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

Archive

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

Archived

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.

Archived

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:

Archived

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.

Archive

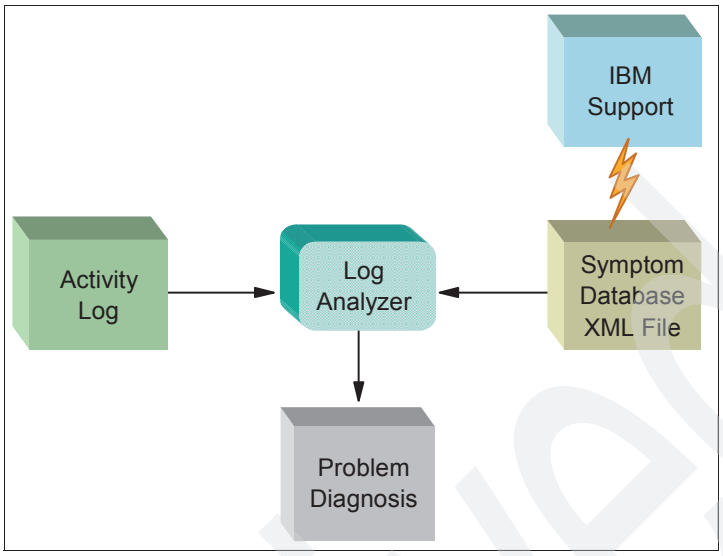


Figure 9-12 Log Analyzer concept

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.

Archived

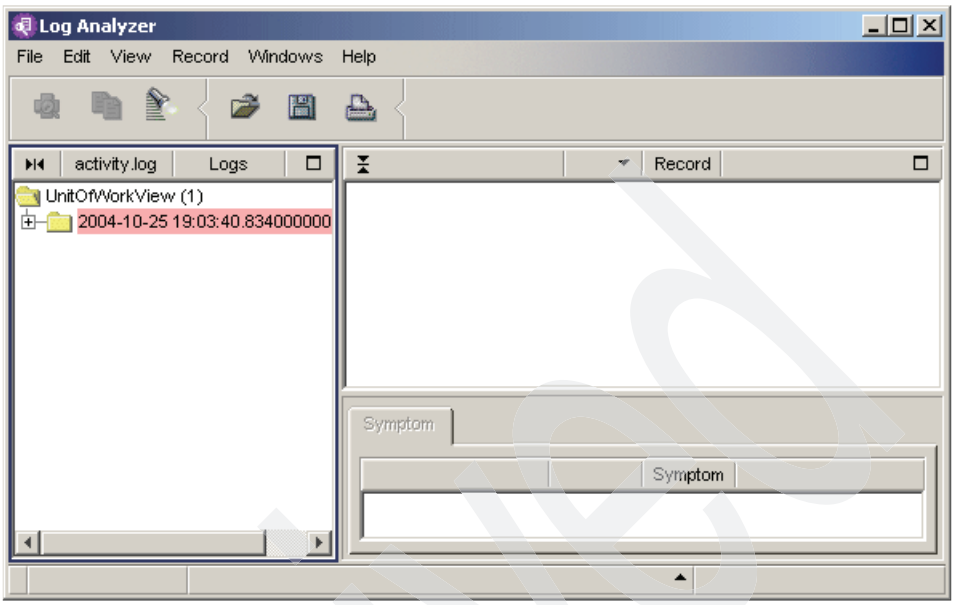


Figure 9-13 Log Analyzer

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.

Archived

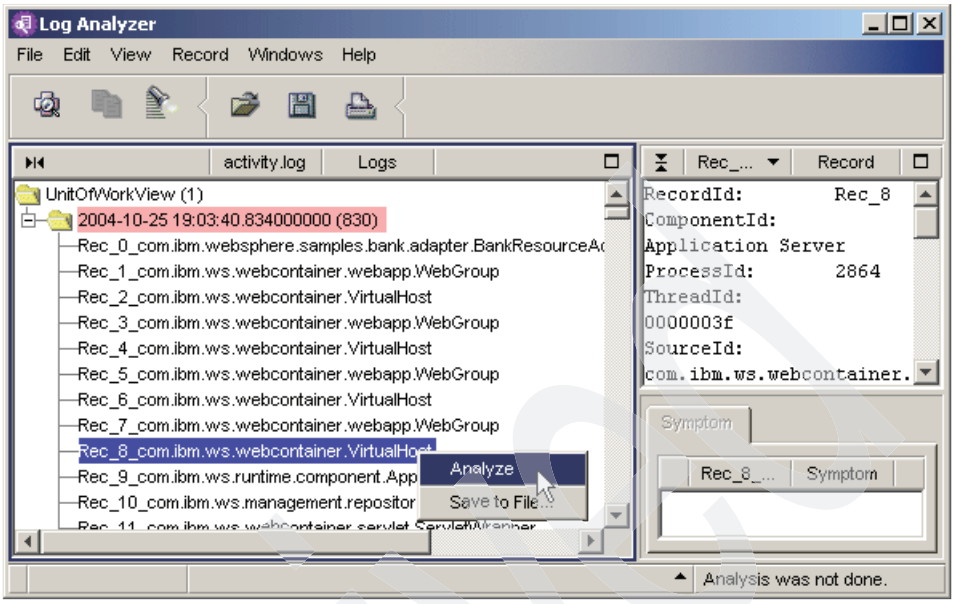


Figure 9-14 Selecting an entry to analyze

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.

Archive

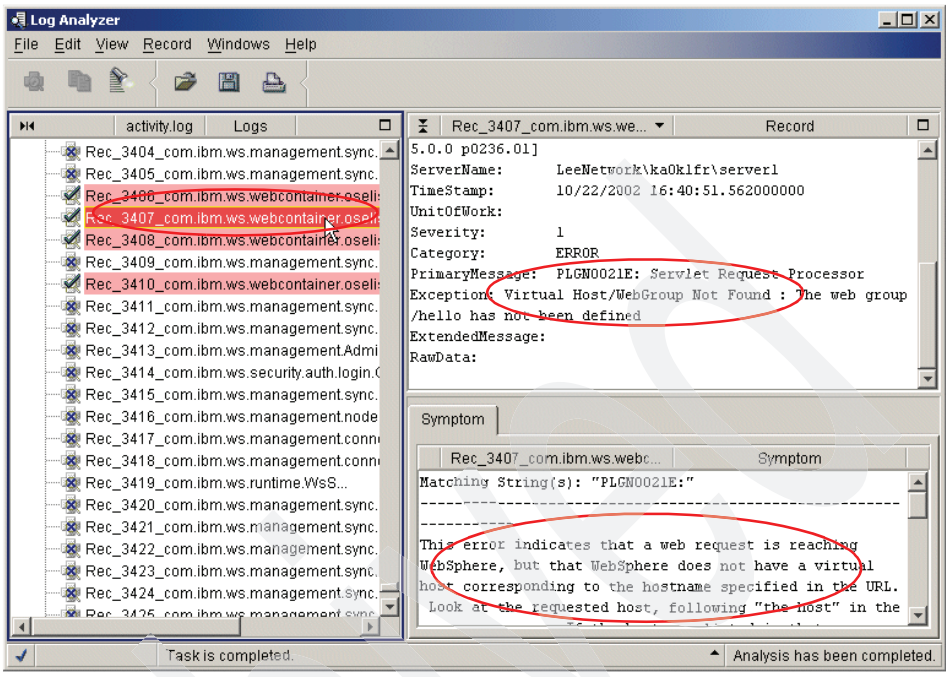


Figure 9-15 Log Analyzer information

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.OSEListenerDispatcher
```

In this example:

- `Rec_3407` is the entry number.
- `com.ibm.ws.webcontainer.oselistener.OSEListenerDispatcher` 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.

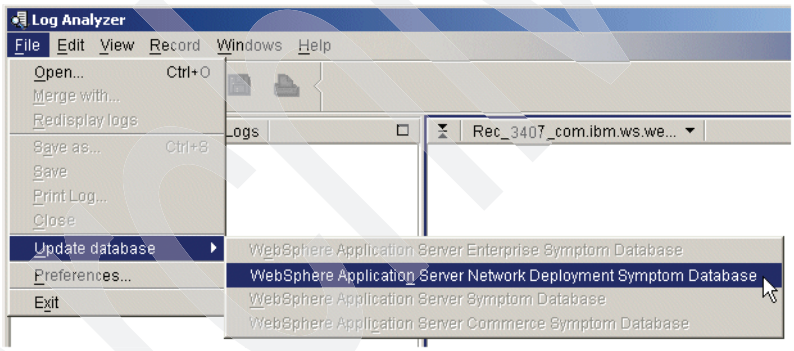


Figure 9-16 Updating the symptom database

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:

Archive

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.

Archived

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)
    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
```

Archived

```
    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
...
   72 (top)/domain                                       javax.naming.Context
   72    Linked to context: PericlesCell
   73 (top)/cellname                                     java.lang.String
   74 (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.IBM
TrackerDebug
```

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
Invalidated Sessions Count:0
Invalidated 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 wriiten 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)

Archived

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.

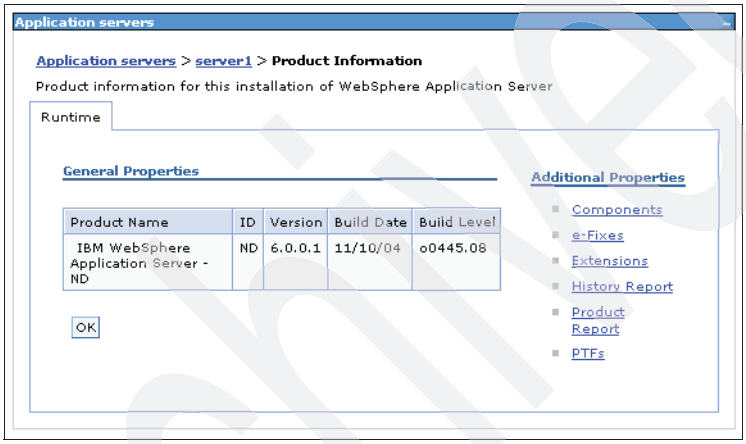


Figure 9-17 Product information

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>
```

Archived

```
  <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
dmgr01cell\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:
jitc), Java Compiler = jitc, 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
--------------------------------------------------------------------------------
```

Archived

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 linem 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

Archived

- *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/

Archived

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

© Copyright IBM Corp. 2005. All rights reserved.

Archived

10

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

Archived

© Copyright IBM Corp. 2005. All rights reserved.

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.

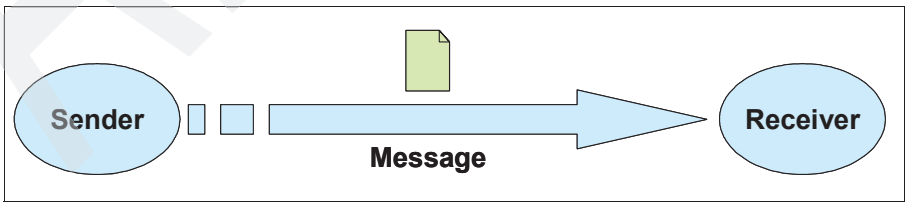


Figure 10-1 Direct communication using synchronous messaging

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.

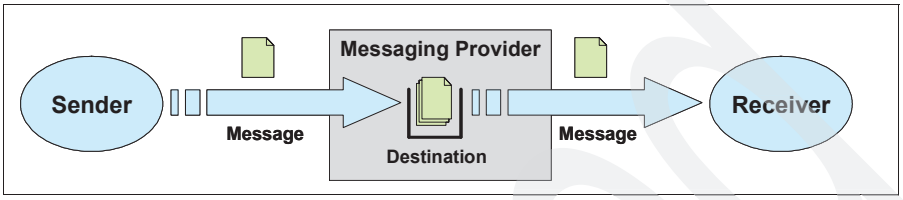


Figure 10-2 Indirect communication via a destination using asynchronous messaging

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.

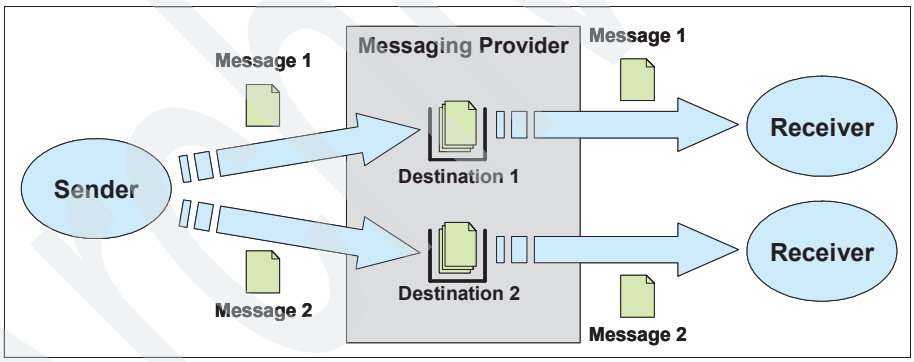


Figure 10-3 Communicating with multiple receivers using asynchronous messaging

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.

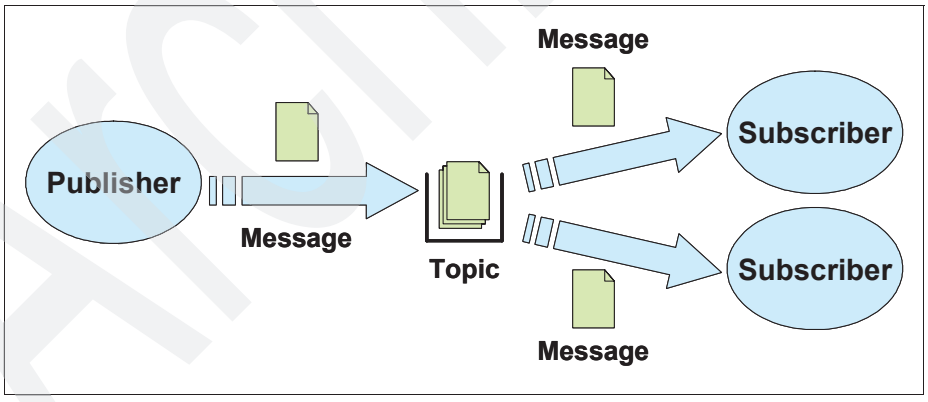


Figure 10-4 Publish/Subscribe messaging model

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

Archived

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.

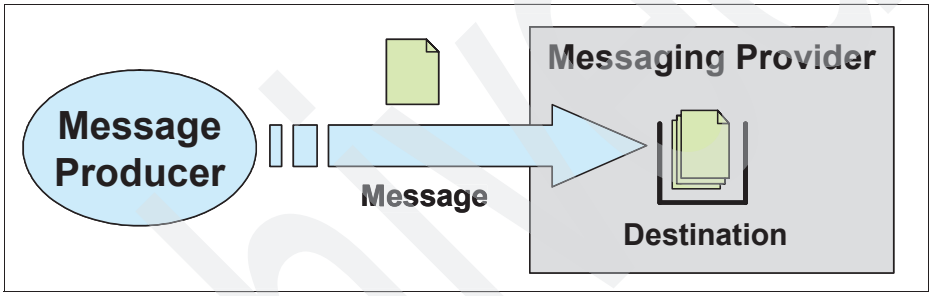


Figure 10-5 Message producer pattern

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.

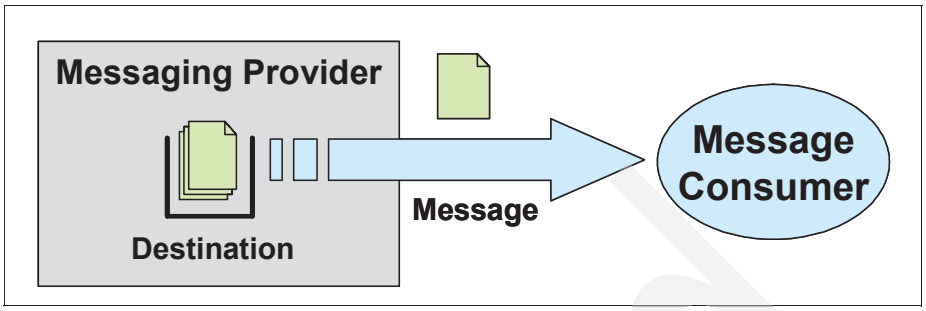


Figure 10-6 Message consumer pattern

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.

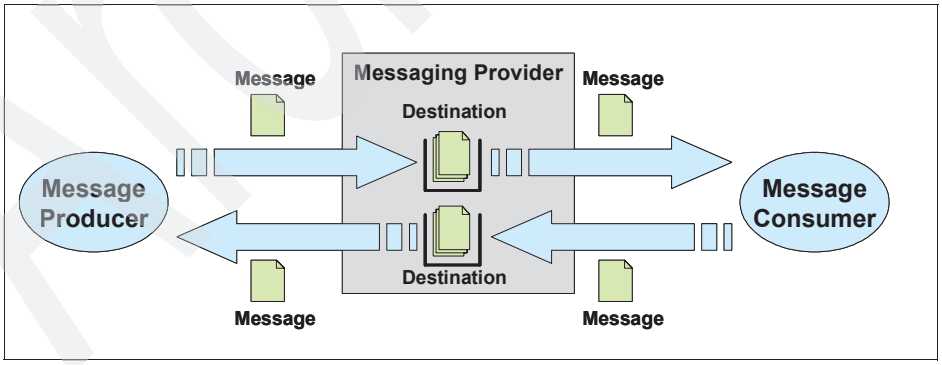


Figure 10-7 Request-reply pattern

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.

Archive

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);
```

Archived

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.

Archive

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.

Archived

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 are modes are discussed in the following sections.

Archive

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.

Archived

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

Archived

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.JMSException. The JMSException class is the root class of all JMS API exceptions.

A JMSException contains the following information:

- A provider-specific string describing the error
- A provider-specific string error code
- A reference to another exception

  The JMSException is usually caused by another exception being thrown in the underlying JMS provider. The JMSException 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 the implements the javax.jms.ExceptionListener interface is shown in Example 10-12.

Archived

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.

Archived

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.

Archived

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.

Archived

- 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.

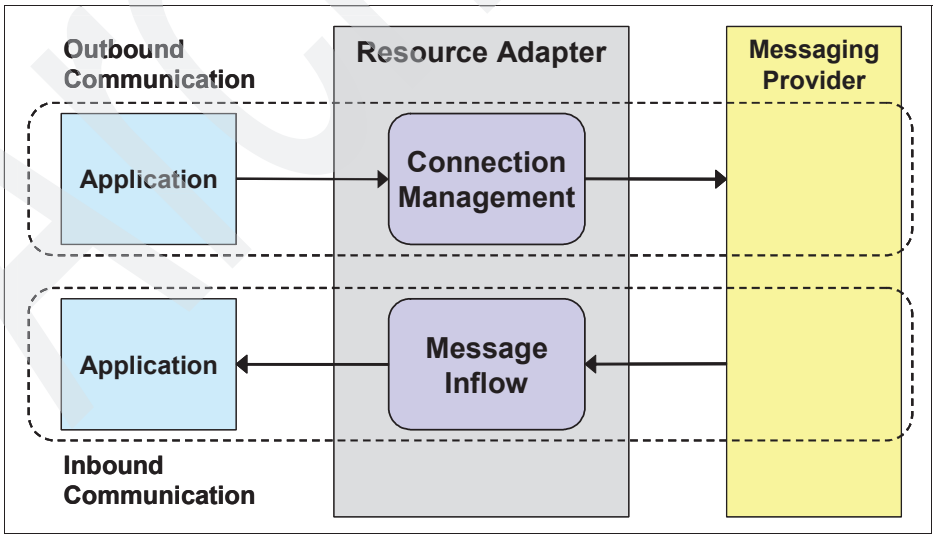


Figure 10-9 Inbound and outbound communication using a resource adapter

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 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

Archived

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,

Archived

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

Archived

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.

Archive

- 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.

Archived

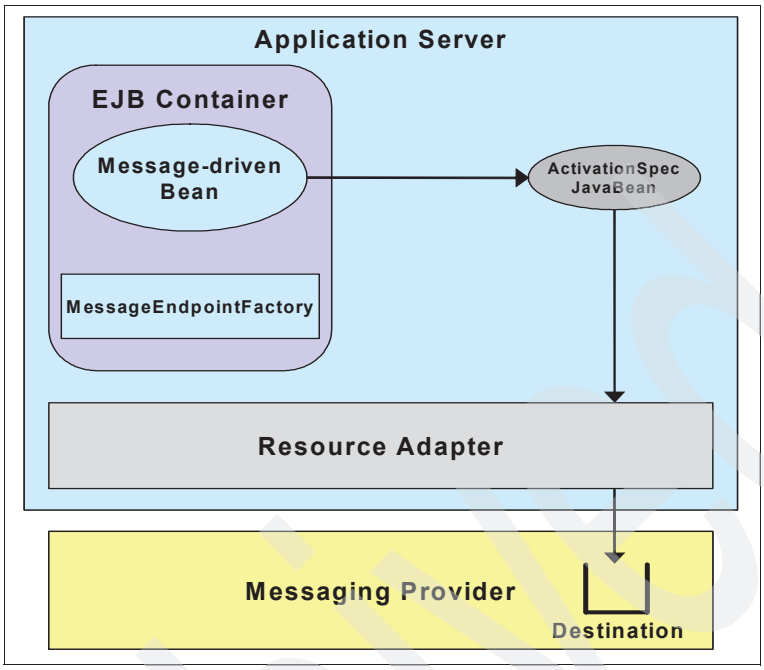


Figure 10-10 Associating an MDB with a destination using a ActivationSpec JavaBean

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.

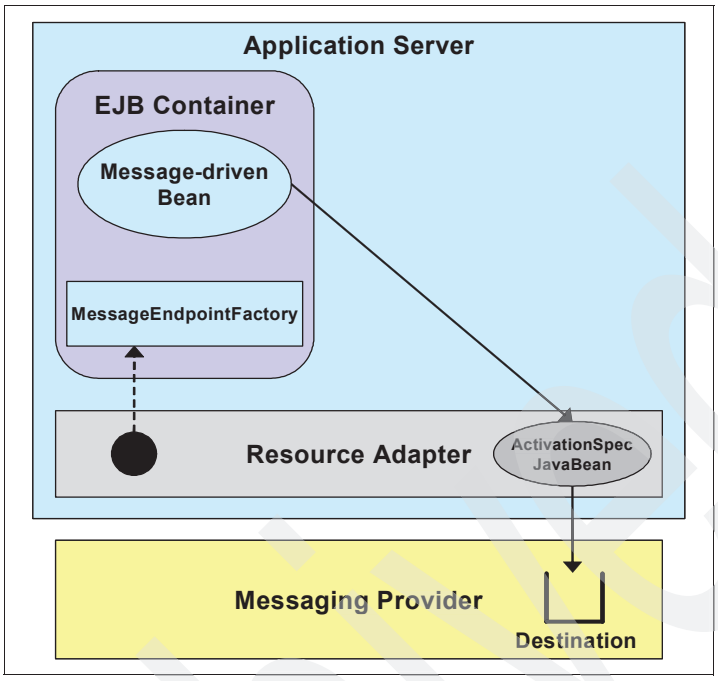


Figure 10-11 Activating a message endpoint

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.

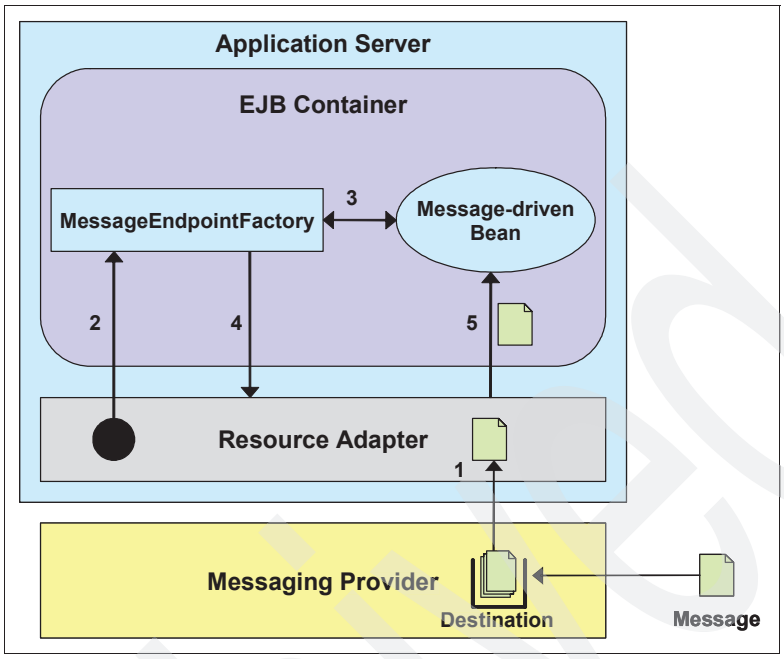


Figure 10-12 Delivering a message to a message endpoint

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 a 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.

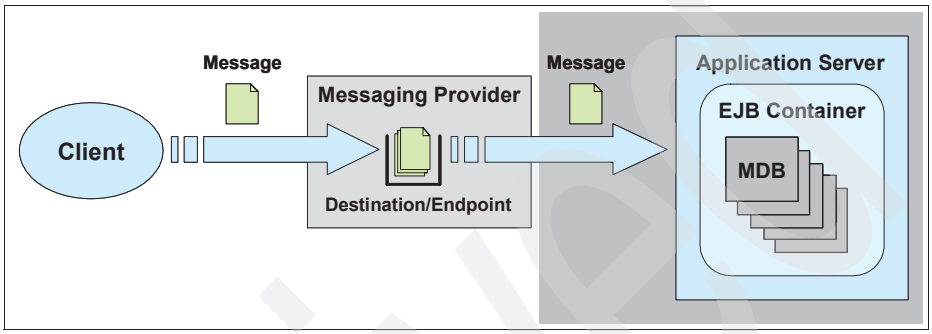


Figure 10-13 Client view of a message-driven bean

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.

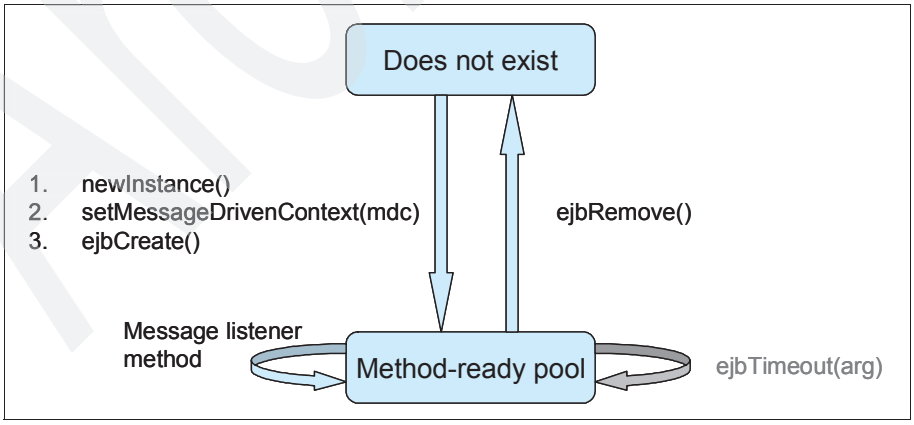


Figure 10-14 Message-driven bean life cycle

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);
}
```

Archived

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 1‘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 occured, 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.

Archived

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 occured, rollback the transaction
            userTx.rollback();
         }
         catch (SystemException e2)
         {
            e2.printStackTrace();
         }
      }
   }
}
```

Archived

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.

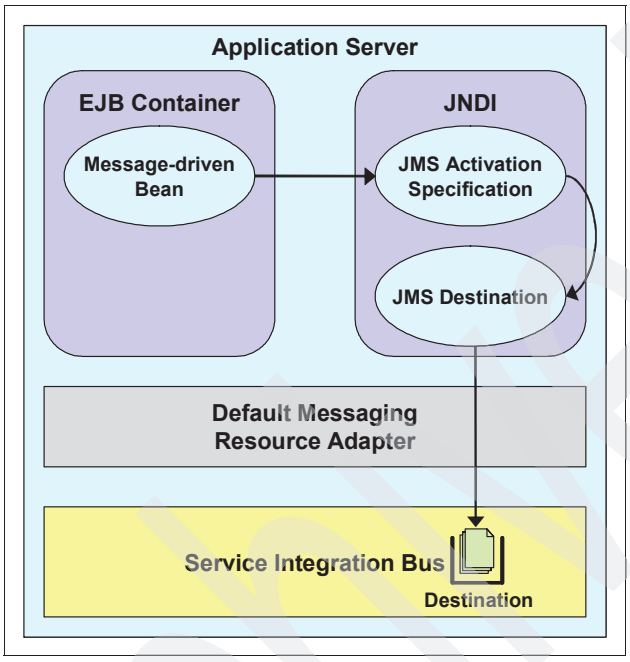


Figure 10-15 Associating an MDB with a destination using a JMS activation specification

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.

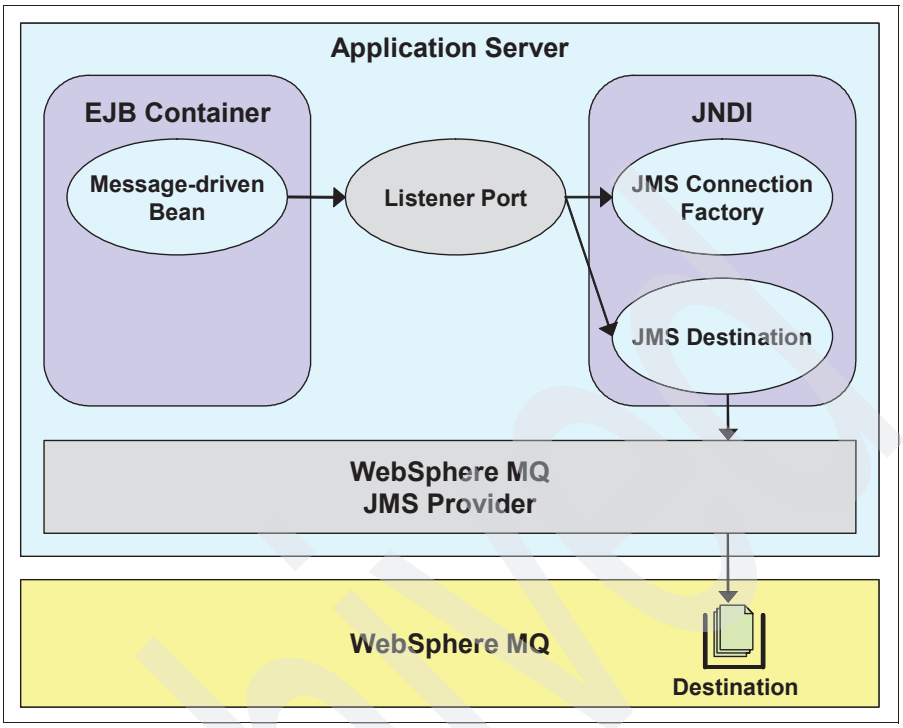


Figure 10-16 Associating an MDB with a destination using a listener port

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.

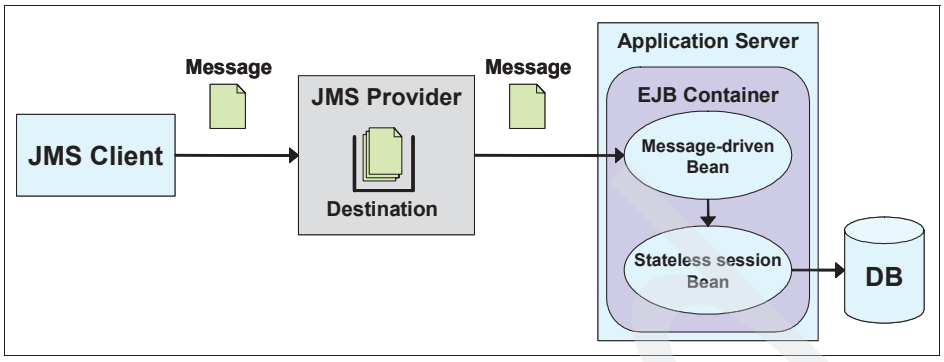


Figure 10-17 Delegating business logic to a stateless session bean

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.

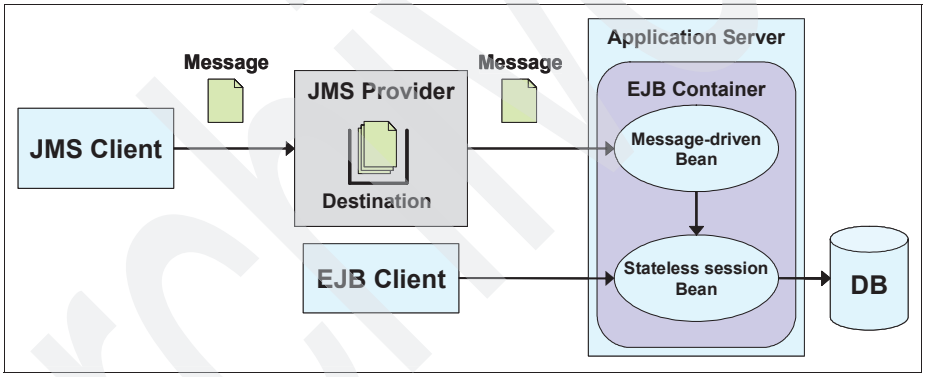


Figure 10-18 Business logic reuse

- 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 a 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 be 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 the 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 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.

Archive

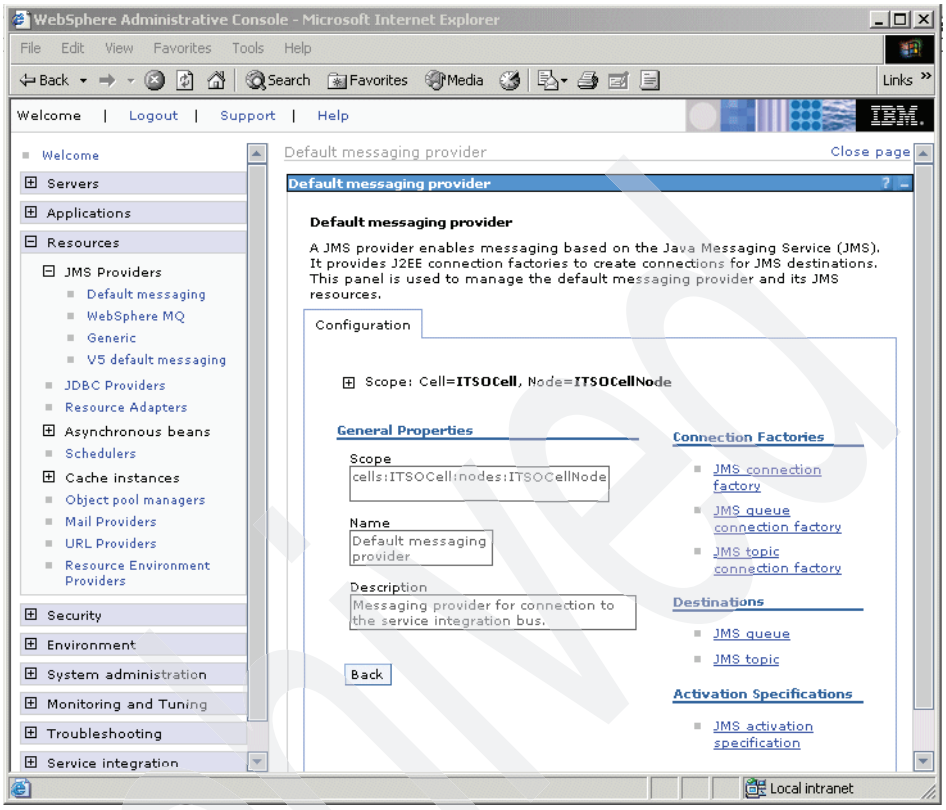


Figure 10-19 Default messaging provider configuration properties

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.

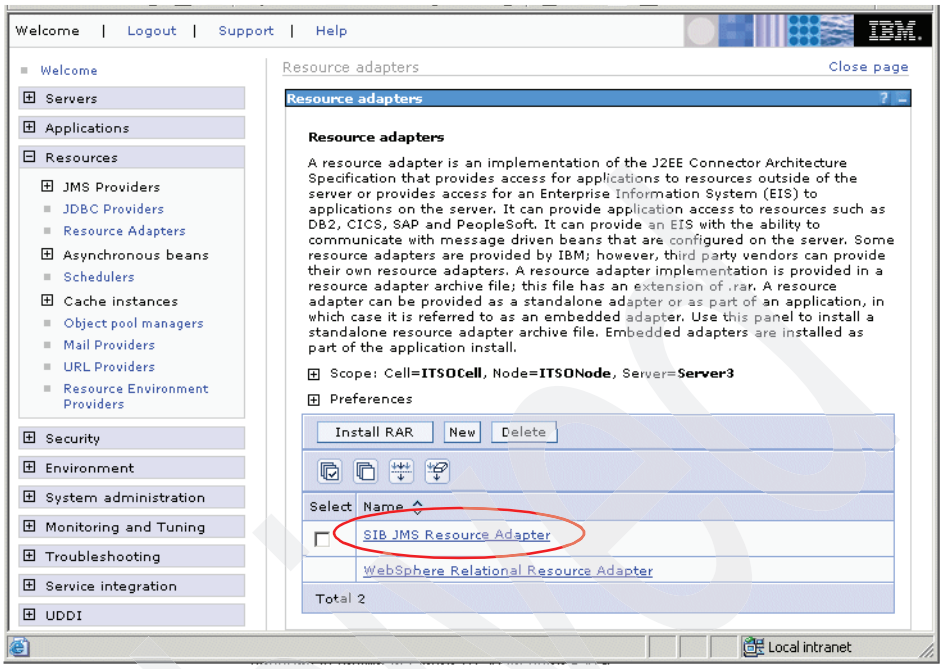


Figure 10-20 Resource adapters

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.

Archive

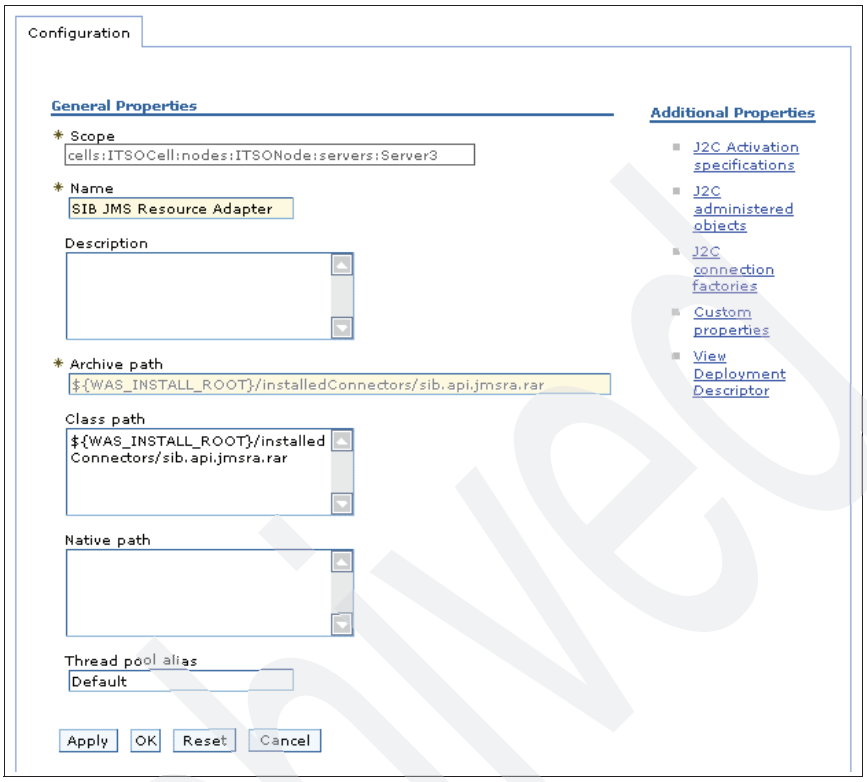


Figure 10-21 SIB JMS Resource Adapter properties

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

Archive

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.

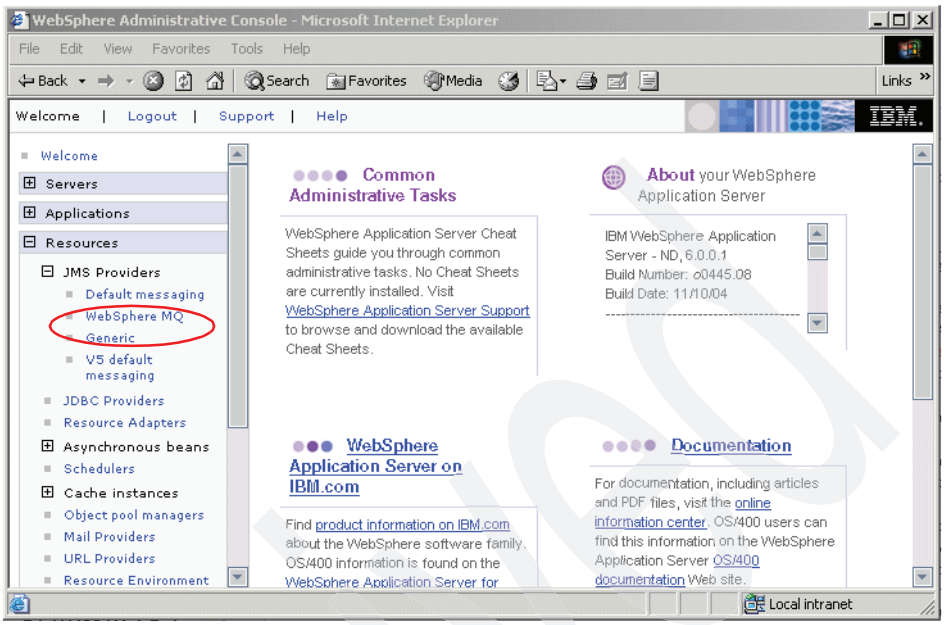


Figure 10-22 Finding the WebSphere MQ JMS provider in the navigation tree

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.

Archive

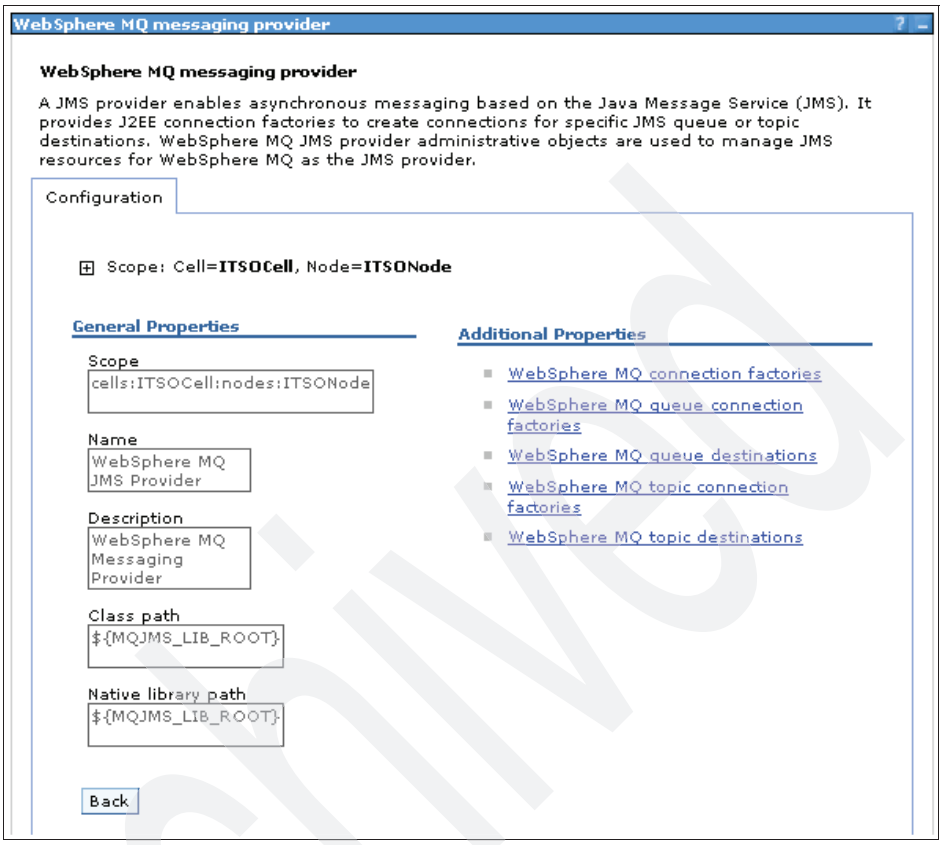


Figure 10-23 WebSphere MQ JMS provider configuration properties

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	► 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.

Archived

Property	Description
Native library path	► 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.

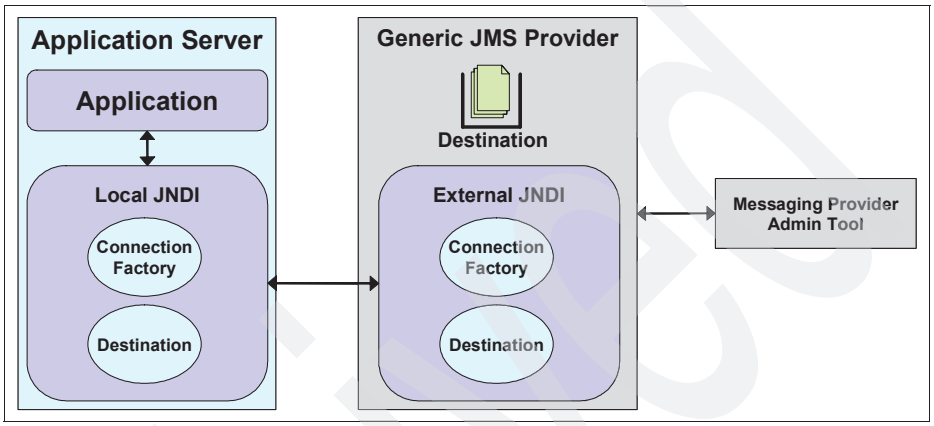


Figure 10-24 Generic JMS provider components

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.

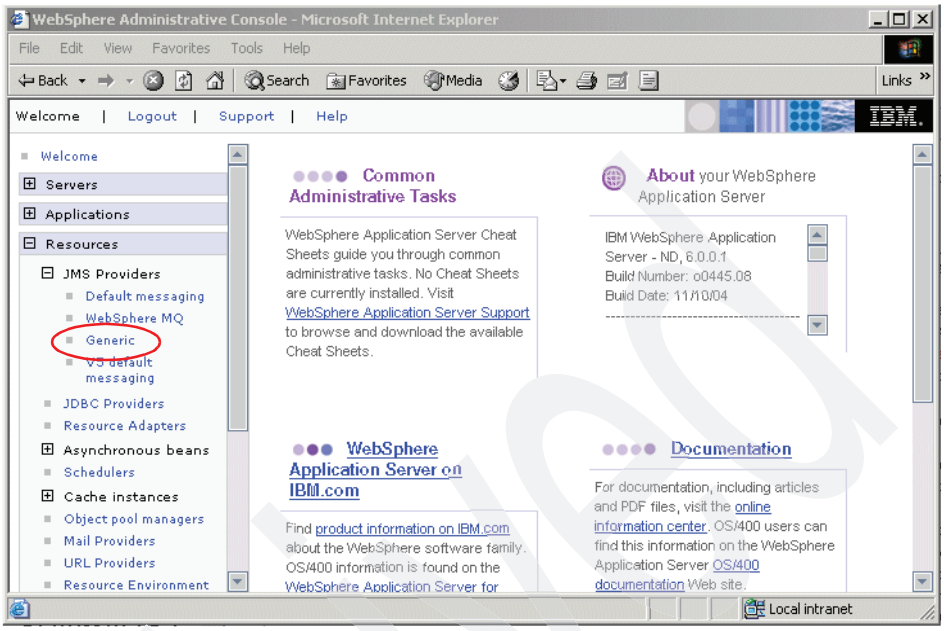


Figure 10-25 Finding the generic JMS provider in the navigation tree

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.

Author

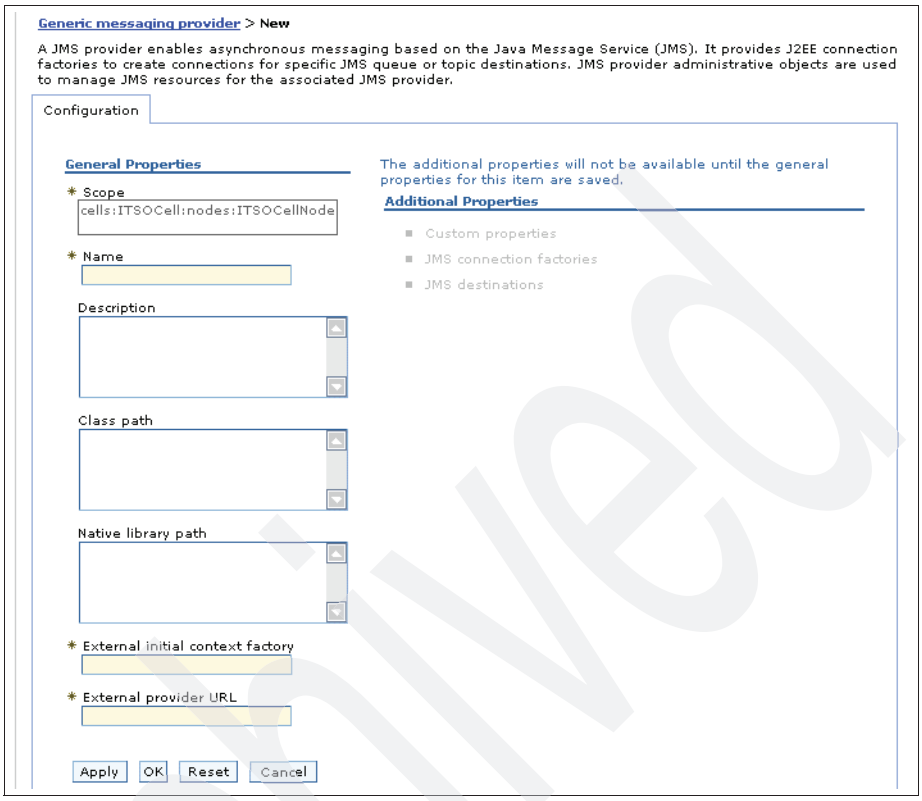


Figure 10-26 Generic JMS provider general properties

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

Archived

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 connection factory defaults to using 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 programatically by a JMS client running inside the J2EE Client Container, or administratively by the connection factory. The client identifier property enables an administrator to specify the identifier that should be assigned to connections created by the connection factory. This identifier is then used if the JMS client attempts to register a durable subscription without programmatically providing a client identifier.
Durable subscription home	Messages that are published to a topic that has inactive durable subscribers registered, must be stored by the messaging provider and delivered to each subscriber as and when they become active. The durable subscription home property enables an administrator to specify which messaging engine is responsible for persisting such messages. A suitable messaging engine must be specified in order to enable JMS clients to use durable subscriptions.

Quality of service properties

The JMS specification supports two modes of delivery for JMS messages; persistent and non-persistent. However, the service integration bus defines several levels of reliability that can be applied to both persistent and non-persistent messages. The levels of reliability defined by the service integration bus are discussed in more detail in “Reliability” on page 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	Read ahead is an optimization technique used by the default messaging JMS provider to reduce the time taken to satisfy requests from message consumers. It works by preemptively assigning messages to message consumers. Messages assigned to message consumers are locked on the server and sent to a proxy destination on the client, prior to the message consumer requesting them. The message consumer running within the client is then able to consume the messages from the local proxy destination. Messages that are locked on the server cannot be consumed by any other message consumers for that destination. Messages that are assigned to a message consumer, but not consumed before it is closed, are subsequently unlocked on the server and are then available for receipt by other message consumers. Valid values for this property are: ► **Default** Read ahead is enabled in situations where there can only be a single message consumer. That is, read ahead is enabled for message consumers on non-durable subscriptions and unshared durable subscriptions. This is the default value for this property. ► **Enabled** Read ahead is enabled for all message consumers. ► **Disabled** Read ahead is disabled for all message consumers. The read ahead property for the connection factory can be overridden by specifying a value for the read ahead property on a specific JMS destination.
Temporary queue name prefix	Enter the prefix to be used when generating the names of temporary queues created within JMS clients using this connection factory. The prefix can be up to twelve characters long. By default, no value is specified for this property, which causes temporary queues to be generated without any prefix.

Archive

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	This property specifies whether multiple TopicSubscribers, created using this connection factory, can consume messages simultaneously from a single durable subscription. Normally, only one session at a time can have a TopicSubscriber for a particular durable subscription. This property enables you to override this behavior, to enable a durable subscription to have multiple simultaneous consumers. Valid values for this property are: ► **In cluster** Allow sharing of durable subscriptions when connections are made from within a server cluster. This is the default value for this property. ► **Always shared** Share durable subscriptions across connections. ► **Never shared** Never share durable subscriptions across connections.

Advanced administrative properties

The connection factory for the default messaging JMS provider also exposes a number of advanced properties that are used for administrative purposes. These properties are described in Table 10-16.

Table 10-16 JMS connection factory advanced administrative properties

Property	Description
Component-managed authentication alias	Specify the J2C authentication data entry alias to be used to authenticate the creation of a new connection to the JMS provider. The alias encapsulates the user ID and password that will be used to authenticate the creation of the connection. 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.

Archive

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	Use this property to enable the sharing of JDBC connections between the data store component of a messaging engine and container-managed persistence (CMP) entity beans. In order for this to provide a performance improvement, the data source used by the data store and the CMP entity bean must be the same. If this is the case, a JDBC connection can be shared within the context of a global transaction involving the messaging engine and the CMP entity bean. If no other resources are accessed as part of the global transaction, WebSphere is able to use local transaction optimization in an effort to improve performance. The default value for this property is false (the check box is not selected). Please refer to the WebSphere Information Center for a full description of this performance optimization.
XA recovery authentication alias	Specify the J2C authentication data entry alias to be used to authenticate the creation of a connection to the JMS provider during XA recovery processing. The alias encapsulates the user ID and password that will be used to authenticate the creation of the connection. During XA recovery processing, a connection might need to be made to a messaging engine within the service integration bus. If security is enabled for the bus, it might be necessary to authenticate the creation of the connection. The XA recovery authentication alias is used for this purpose.

Archive

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.

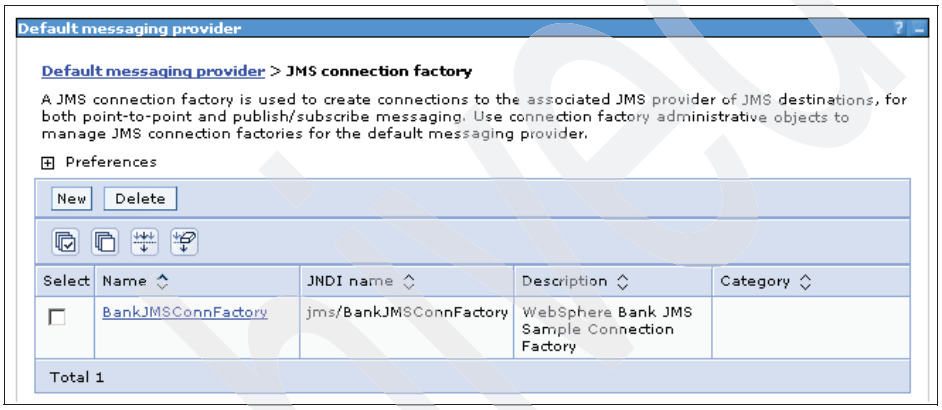


Figure 10-27 Default messaging JMS connection factory administered objects

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.

Archived

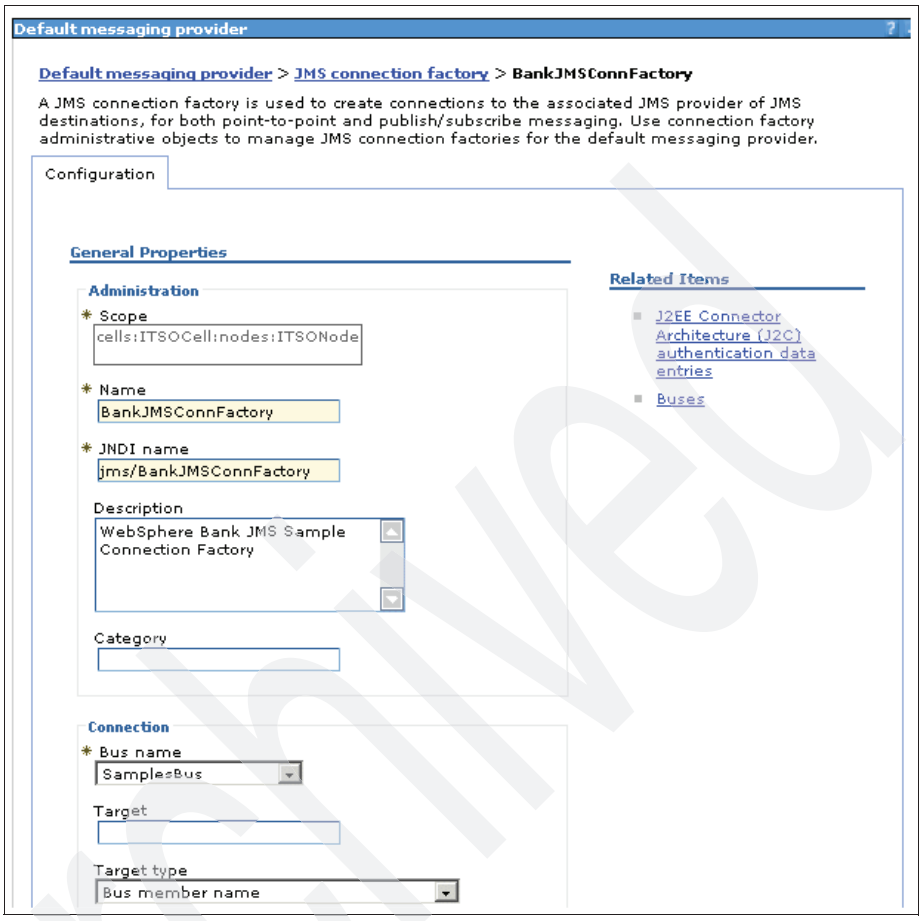


Figure 10-28 Default messaging JMS connection factory properties

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:

Archived

- 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	Use this property to specify the name of the service integration bus on which the destination is defined. The default behavior if, no value is specified for this property, is to assume that the destination is defined on the same service integration bus to which the application is connected. That is, the service integration bus will be determined from the connection factory that is used in conjunction with this JMS destination. The only situation in which a bus name must be specified is if the underlying destination that this JMS destination refers to is defined on a foreign bus. The foreign bus specified can refer to a service integration bus, or to WebSphere MQ. Please refer to section 11.1.7, “Foreign buses” on page 606 for more information.
Delivery mode	Us this property to specify the delivery mode to be used for messages that are sent to this destination. This property allows an administrator to override the delivery mode specified by the JMS client when sending a message. Valid values for this property are: ► **Application** The persistence of messages sent to this destination is determined by the JMS client application when sending a message. This is the default value for this property. ► **Nonpersistent** All messages that are sent to this destination are treated as non-persistent. ► **Persistent** All messages that are sent to this destination are treated as persistent.

Archived

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 descendents of A
A//.	Selects A and all descendents of A
//*	Selects everything
A/./B	Equivalent to A/B
A/*/B	Selects all B grandchildren of A
A//B	Selects all B descendents of A
//A	Selects all A elements at any level
*	Selects all first level elements

Note: The use of wildcards within a topic name for a JMS topic destination is only valid when the JMS topic destination is used by a message consumer. If a message producer attempts to use such a JMS topic destination, a JMS exception will be thrown to the JMS client application.

Refer to the WebSphere Information Center for a full description of using topic wildcards in topic expressions to retrieve topics provided by the default messaging provider and service integration bus.

The properties that are specific to JMS topic destinations are described in Table 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.

Archive

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	The read ahead property on a JMS destination enables an administrator to override the value of the read ahead property specified on the JMS connection factory. Valid values for this property are: ► **Enabled** Read ahead is enabled for all message consumers that are consuming messages from this destination. ► **Disabled** Read ahead is disabled for all message consumers that are consuming messages from this destination. ► **As connection factory** The value of the read ahead property specified on the JMS connection factory should be used. For information about the read ahead property, refer to Table 10-15 on page 531.

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.

Archived

4. 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.

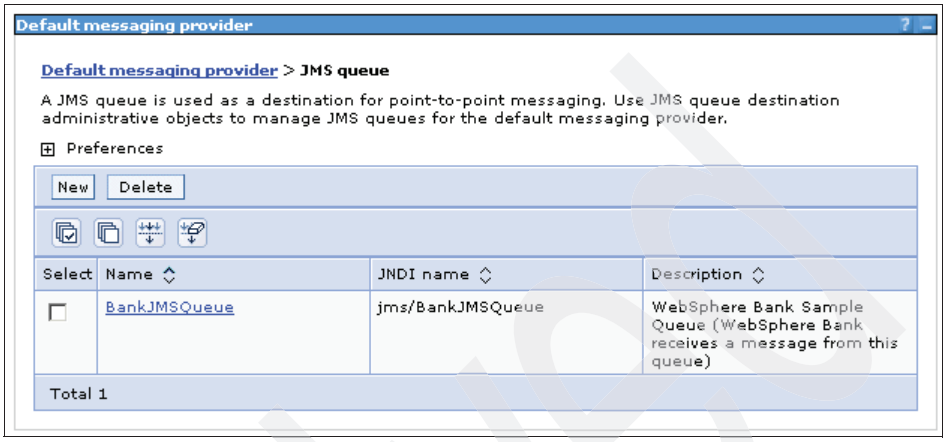


Figure 10-29 Default messaging queue destination administered objects

In this example, we already have one JMS queue destination object defined, called BankJMSQueue.

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 10-30 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.

Archive

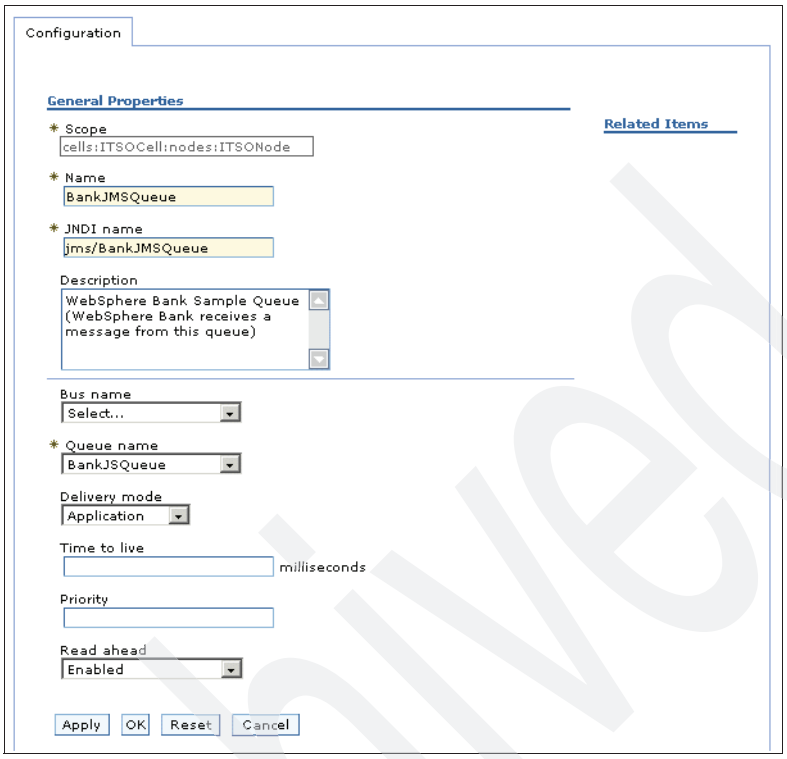


Figure 10-30 Default messaging queue destination properties

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.

Archived

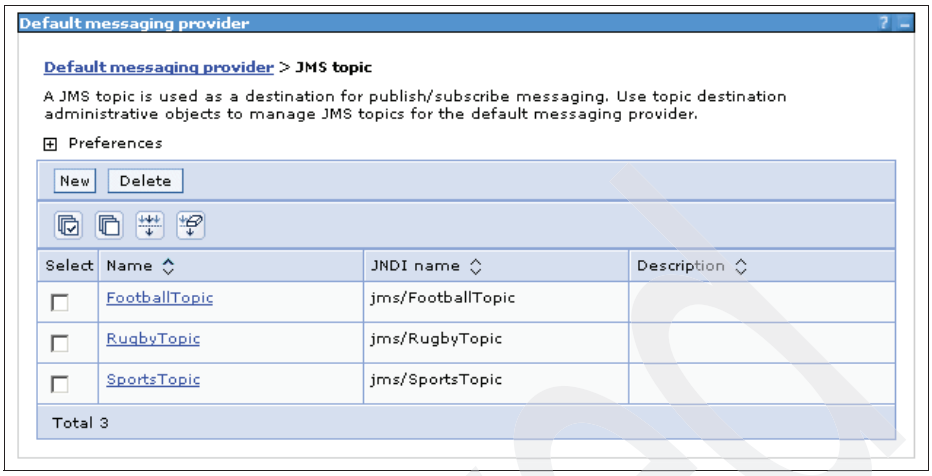


Figure 10-31 Default messaging topic destination administered objects

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.

Archived

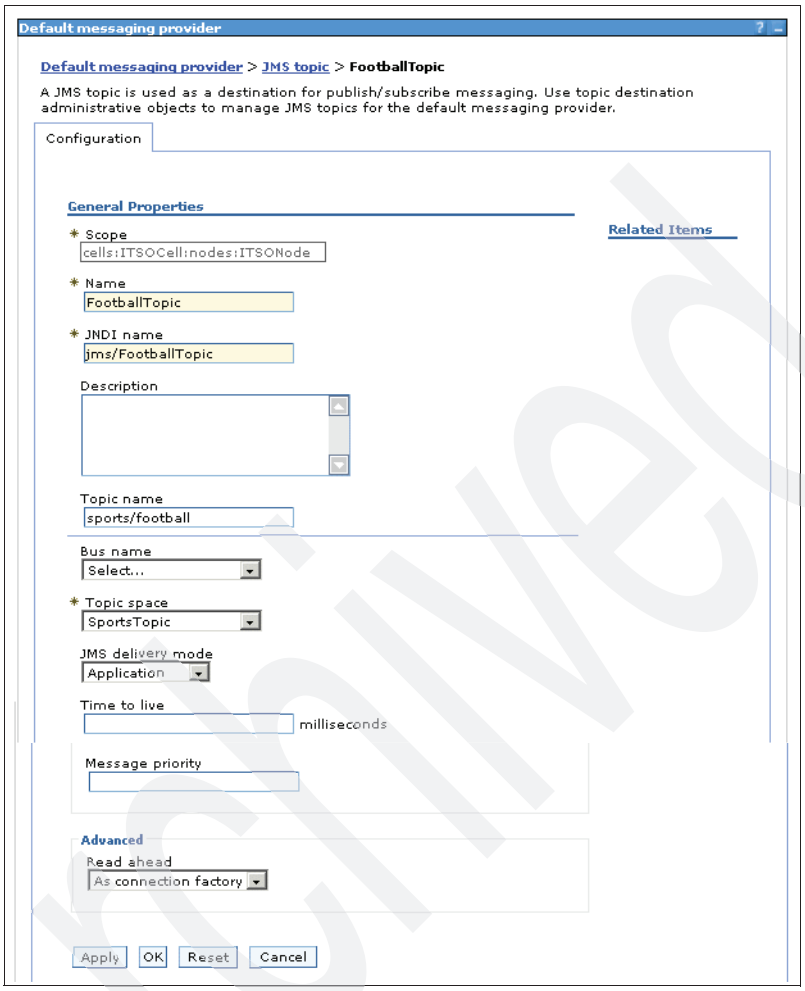


Figure 10-32 Default messaging topic destination properties

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

Archived

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.

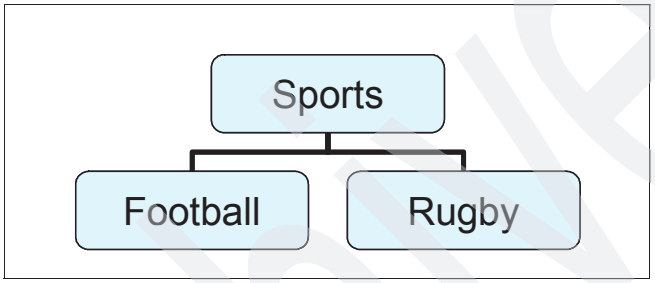


Figure 10-33 Sample sports topic hierarchy

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.

Archived

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.

Archived

Table 10-23 JMS activation specification destination properties

Property	Description
Destination type	Use this property to specify the type of the JMS destination with which a message-driven bean will be associated. Valid values for this property are: ► **Queue** The target destination is a queue destination. This is the default value for this property. ► **Topic** The target destination is a topic destination
Destination JNDI name	You must specify a JNDI name for the target destination.
Message selector	This property specifies a JMS message selector that should be applied to the target JMS destination. Only messages that match this message selector will be delivered to the message-driven bean. By default, no message selector is specified for a JMS activation specification. Refer to “Message selectors” on page 478 for more information.
Bus name	This property is the name of the service integration bus on which the target destination is defined. This bus must exist within the same cell as the application server on which the message-driven bean is running, but this application server is not required to be a member of the bus. However, the best performance will be obtained if the application server on which the message-driven bean is running is a member of the bus specified. A value must be specified for this property.
Acknowledge mode	Use this property to specify how the EJB container acknowledges the receipt of a message by a message-driven bean instance that is using bean managed transactions. Valid values for this property are: ► **Auto-acknowledge** The EJB container automatically acknowledges the delivery of a message when the onMessage method of the message-driven bean successfully returns. ► **Duplicates-ok auto-acknowledge** The EJB container lazily acknowledges the delivery of messages to message-driven beans. This can improve performance, but can lead to a message-driven bean receiving a message more than once.

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 programatically configure a durable subscription. The subscription durability properties on a JMS activation specification enable the configuration properties for a durable subscription to be specified administratively. These properties are described in Table 10-25 on page 548.

Table 10-25 JMS activation specification subscription durability properties

Property	Description
Subscription durability	Use this property to specify whether a JMS topic subscription is durable or nondurable. Valid values for this property are: ► **Durable** The messaging provider stores messages while the message-driven bean is not available, and delivers the messages when the message-driven bean becomes available again. ► **Nondurable** The messaging provider does not store and redeliver messages if a message-driven bean is not available. This is the default value for this property.
Subscription name	JMS clients must provide a subscription name when attempting to register a durable subscription. Because a JMS client can create several durable subscriptions, the subscription name must be unique within the context of a particular client identifier (described within this table). A message-driven bean is not able to programatically specify a subscription name when it creates a durable subscription. A suitable subscription name must be specified in order to enable message-driven beans associated with this JMS activation specification to use durable subscriptions.
Client identifier	JMS clients must provider a unique identifier when attempting to register a durable subscription. This identifier is used by the messaging provider to associate messages with a JMS client while it is inactive. When the JMS client becomes active again, it subscribes to the durable subscription, passing the same unique identifier. The messaging provider is then able to deliver persisted messages to the correct client. A message-driven bean is not able to programatically specify a client identifier when it creates a durable subscription. A suitable client identifier must be specified in order to enable message-driven beans associated with this JMS activation specification to use durable subscriptions.

Archive

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 the 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.

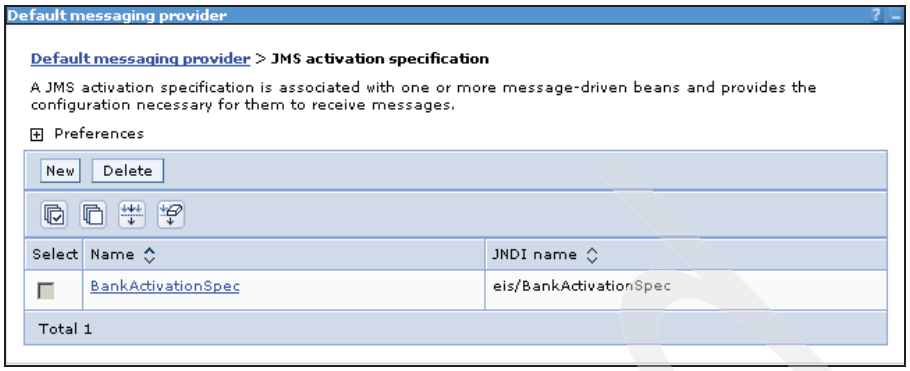


Figure 10-34 Default messaging JMS activation specifications

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